The Catalog Problem:

Deep Learning Methods for Transforming Sets into Sequences of Clusters

by

Mateusz Jurewicz

under the supervision of Professor Leon Derczynski

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

IT UNIVERSITY OF COPENHAGEN

with Tjek A/S as Industrial Partner

Tjek

mateusz.jurewicz@gmail.com

Abstract

The titular Catalog Problem refers to predicting a varying number of ordered clusters from sets of any cardinality. This task arises in many diverse areas, ranging from medical triage, through multi-channel signal analysis for petroleum exploration to product catalog structure prediction. This thesis focuses on the latter, which exemplifies a number of challenges inherent to ordered clustering. These include learning variable cluster constraints, exhibiting relational reasoning and managing combinatorial complexity. All of which present unique challenges for neural networks, combining elements of set representation, neural clustering and permutation learning.

In order to approach the Catalog Problem, a curated dataset of over ten thousand real-world product catalogs consisting of more than one million product offers is provided. Additionally, a library for generating simpler, synthetic catalog structures is presented. These and other datasets form the foundation of the included work, allowing for a quantitative comparison of the proposed methods' ability to address the underlying challenge. In particular, synthetic datasets enable the assessment of the models' capacity to learn higher order compositional and structural rules.

Two novel neural methods are proposed to tackle the Catalog Problem, a set encoding module designed to enhance the network's ability to condition the prediction on the entirety of the input set, and a larger architecture for inferring an input-dependent number of diverse, ordered partitional clusters with an added cardinality prediction module. Both result in an improved performance on the presented datasets, with the latter being the only neural method fulfilling all requirements inherent to addressing the Catalog Problem.

Abstrakt

Det titulære Catalog Problem henviser til at forudsige varierende antal ordnede klyngedannelse fra sæt af enhver kardinalitet. Denne opgave opstår på mange forskellige områder, fra behandling af patienter til multikanal signal analyse i petroleum udforskning, til forudsigelse af produkt katalog struktur. Denne tese fokusere på det sidstnævnte, som er et eksempel på en række udfordringer, der er forbundet med ordnet klyngedannelse. Disse udfordring omfatter indlæring af varierende klyngebegrænsninger, relationel ræsonnementer, og håndtering af kombinatorisk kompleksitet. Alle disse områder udøgr unikke udfordringer for neurale netværk, der sammensætter elementer af sætrepresentation, klyngedannelse, og indlæring af ordne.

For at kunne løse Catalog Problemet, et kureret datasæt er vedlagt med over 10.000 produktkataloger der består af mere end en million produkter. Desuden præsenteres et library til generering af enklere, syntetiske katalogstrukturer. Disse og andre datasæt danner grundlaget for det følgende forskning, og giver mulighed for en kvantitativ sammenligning af de forslåede metoders evne til at løse den underliggende udfordring. Især gør syntetiske datasæt det muligt at vurdere modelens evne til at lære højere ordensammensætnings- og strukturelle regler.

Der foreslås to nye metoder til at løse Catalog Problemet: et sætkodningsmodul der er udformet til at forbedre netværkets evne til at hærde forudsigelsen på hele inputmængden, og en større arkitektur til at udlede en variabel antal forskellige, delvis ordnede klynger med et ekstra kardinalitetsforudsigelsesmodul. Begge resultere i en forbedret ydeevne på de presenterede datasæt, idet sidstenævnte er den eneste neurale metode, der opfylder alle de krav, der er forbundet me Catalog Problemet.

To my loved ones - my fiancée Karolina, my family and my friends

Acknowledgements

I would like to thank Professor Leon Derczynski for being a stalwart supporter of this work throughout the entirety of my PhD education, professor Barbara Plank for the help in navigating an academic environment that was largely new to me, Professor Graham Taylor for giving me a chance to collaborate with his research groups at both the Vector Institute and University of Guelph during a remote stay abroad. I would also like to thank Professor Juho Lee for being a treasure trove of ideas for new experimental methods and Professor Rasmus Pagh for his unique perspective on the challenge at hand. Furthermore I would like to express appreciation of fellow students and postdoctoral researchers whose willingness to share their experiences, readiness to help and overall sense of humor made this process that much more rewarding. These include Manuel Ciosici, Daniel Varab, Marija Stepanović, Philip Lemaitre and Rasmus Lehmann, among many others. I am also deeply grateful to the entire PhD Support Team at ITU.

As this is an Industrial PhD project, I would like to deeply thank the entire team at Tjek for their continued support in a myriad of different ways throughout these last three years. Thanks go to Morten Bo Rønsholdt and Jack Tolboe for giving this project the initial green light, to Sarah Vang Nøhr and Tariq Siddique for the support during the process of applying for funding, to Peter Ebbesen and Jeppe Bårris for being the industrial supervisors, to Malene Egesø Kjerrumgaard for helping with everything related to the actuarial aspects of the project. I would also like to thank the entire Support and Services Department - Andreas, Joanna, Sirin, Thomas and others, for carefully, painstakingly annotating the data without which this project

wouldn't be at all possible. My thanks go to the Software Engineering Team, among them Iasonas Zotos, Bjarne Petersen, Angese Bussone, Marten Sievers, Simon Jensen, Joakim Karlsson, Laurie Hufford, Mazvydas Katinas, Io Klarstrup, Henrik Tudborg, Martin Gausby and Jesper Louis Andersen for their patience, guidance and professionalism. I would also like to thank Anders Hansen for inheriting the project in its later stages and showing continued support nonetheless.

I would like to thank Innovations Fund Denmark for providing a significant amount of funding to this project and extending further help during the pandemic and corresponding lock-downs. I am also deeply grateful for the hard work of the academic reviewers, whose insightful and supportive feedback regarding the included articles helped shaped them into the form presented here. Last but not least, I would like to thank my family and friends, without whose support none of this would have been possible and to whom this thesis is dedicated. My deepest thanks to you all.
Words truly cannot express how grateful I am.

Table of Contents

List of Figures

Notes on Notation

Scalars, Vectors, Matrices and Sets

a	A scalar (integer or real). Also used to denote a single element of a set or of an ordered sequence when dimensionality is irrelevant
$\mathbf{a}$	A vector
$\mathbf{A}$	A matrix
$\mathbb{A}$	A set
$\lvert\mathbb{A}\rvert$	The cardinality of a set
$(0, 1, 2)$	A sequence containing 0, 1 and 2 in sorted order
$\{0, 1, 2\}$	A set containing 0, 1 and 2
$\{0, 1, \ldots, n\}$	A set of all integers between 0 and n
π^n	A permutation of length n
$\mathbf{A}_\pi$	A matrix whose columns are permuted according to π

Neural Networks

ρ, ϕ	Used as shorthand for fully connected neural networks
$\mathbf{X}_\pi$	An arbitrarily ordered matrix representing the set $\mathbb{X}$ that forms the input to the neural network
$\hat{\mathbf{y}}$	A vector output predicted by the neural network
$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}; \theta)$	A loss function parameterized by θ, the weights of a neural network

Other

$\mathbb{R}$	The set of real numbers
$\mathbb{Z}$	The set of all integers
$\mathcal{P}(\mathbb{X})$	The powerset of set $\mathbb{X}$
$\mathcal{N}$	The Gaussian distribution

Abbreviations

ACP Attentive Clustering Process.

CCP Clusterwise Clustering Process.

CoS Compositional Score.

EPC Electronic Product Catalog.

EPN Enhanced Pointer Network.

GNN Graph Neural Network.

HN Hopfield Network.

IC Input Composition.

ISAB Induced Set Attention Block.

LN Layer Normalization.

LSTM Long Short-Term Memory.

MAB Multihead Attention Block.

MLP Multi-layer Perceptron.

NCP Neural Clustering Process.

NN Neural Network.

NOC Neural Ordered Clusters.

PMA Pooling by Multihead Attention.

PN Pointer Network.

PPC Paper Product Catalog.

S2S Set-to-sequence.

SAB Set Attention Block.

SIT Set Interdependence Transformer.

SME Subject Matter Expert.

ST Set Transformer.

StS Structural Score.

TSP Travelling Salesman Problem.

Glossary of Terms

Catalog Problem An umbrella term for problems that require taking as input sets of varying cardinality, and predicting an input-dependent number of ordered, *partitional clusters* in accordance with target preference. "Partitional" refers to the requirement that every cluster contains at least one element and that each element is assigned to exactly one cluster. An example of such a problem is the prediction of product catalog structures [1], from which this problem takes its name. For a visual explanation, the reader is referred to Figure 2.1.

Catalog Structure Either a permutation or an ordered, partitional clustering of the available elements of the input set, representing a catalog. The former relates to the *Incomplete Approach* (set-to-sequence) to the Catalog Problem, the latter to the harder, *Complete Approach* (primarily through neural ordered clustering methods).

Complete Approach One of the two presented approaches to address *Catalog Problem*. Unlike the *Incomplete Approach*, it respects all three aspects of the Catalog Problem, in that it (1) predicts an input-dependent number of ordered, partitional clusters (2) from sets of any cardinality (3) in a supervised manner. A Complete Approach does not require the target number of clusters to be known to the model ahead of time and does not include the prediction of the in-cluster order of elements.

Electronic Product Catalog A type of *Product Catalog*, specifically a digital one, displayed on the screen of an electronic device. These include the *Incito* catalogs generated by presented models and rendered by the Incito service. The presented, general neural network methods learn from historic *Paper Product Catalogs* how to compose good Electronic Product Catalogs.

Incito A catalog-rendering software service developed by Tjek A/S, capable of taking a nested list of product offers grouped into ordered sections and rendering

them into a readable *Electronic Product Catalog* on most modern devices (PCs, smartphones, tablets). Described in more detail in Chapter 1.3.2.

Incomplete Approach One of the two presented approaches to address the *Catalog Problem*. The Incomplete Approach applies *set-to-sequence* methods to the Catalog Problem. It includes predicting a permutation of all elements of the input set. It does not predict an input-dependent number of sections, instead requiring this number to be known a priori and fed to the model as special section-break tokens. Additionally, it involves predicting the in-section order of elements, which is ignored by the *Incito* rendering service. These shortcomings are addressed in the *Complete Approach* to the Catalog Problem.

Neural Clustering Neural-network-based methods that jointly cluster and learn representations, identifying clusters directly within the forward pass of the network, in a supervised manner. Whilst neural clustering could encompass other types of methods, for the purposes of simplicity within this work the term is used to refer specifically to supervised neural clustering methods that predict clusters in the forward pass and don't require a separate process to learn element representations. This line of research is also framed as *amortized clustering* [20–22], where the term "amortization" referred to the investment of large computational resources to train a model that is then used for fast posterior inference [20, 23]. The term covers both element-wise and cluster-wise methods, as elaborated on in Chapter 2.5.

Paper Product Catalog A type of *Product Catalog*, specifically a paper-based, printed catalog, sometimes referred to as a brochure or flyer.

Partitional Clustering Partitional clustering decomposes a set of elements into a set of disjoint clusters. Given a set of n elements, a partitioning method constructs k ($k \leq n$) partitions of the data, with each partition representing a cluster. It splits the input set into k clusters by satisfying the following requirements: (1) each cluster contains at least one element, and (2) each element belongs to exactly one cluster. Contrast with fuzzy partitioning, where an element can belong to more than one group.

Product Offers The basic building blocks of a *Product Catalog*. These consist of their textual description and heading, often associated with an image and price tag. In the context of the *Catalog Problem*, product offers are the elements of the input set, which then get clustered into "good" sections, where goodness is

learned from the training data rather than constrained to a metric space. Such sections need to be ordered into a full *Catalog Structure*.

Product Catalog Within the context of this work, a product catalog is a readable, visual presentation of a set of *Product Offers*. Individual product offers are grouped into sections, such as the pages of a printed *Paper Product Catalog*, which are then ordered into a complete *Catalog Structure*. Examples of sequential sections from a sample product catalog can be seen in Figure 1.2.

Sequence-to-sequence A family of problems (and the methods to address them) that require taking as input a sequence and outputting a sequence, or in other words a mapping of sequences to sequences. Within the field of neural networks, this most commonly involves an encoder-decoder architecture [24, 205]. Contrast with *Set-to-sequence*.

Set-to-sequence A family of problems (and the methods to address them) that require taking as input a varying-cardinality set of elements, and outputting a permutation of these elements. Set-to-sequence (S2S) methods commonly consist of a set encoder and a permutation learning module, as described in the first included article in Chapter 4. The *Incomplete Approach* to the *Catalog Problem* is a set-to-sequence method.

Synthetic Catalogs Simplified *Catalog Structures* consisting of atomic elements grouped into ordered sections according to flexible, customizable rules of section composition and catalog structuring. These synthetic catalogs are generated in an algorithmic as opposed to manual way. They are used as a simplification of the real-world *Paper Product Catalogs* included in the provided PROCAT dataset [1]. Described in more detail in Chapter 3.1.

Chapter 1

Introduction

This chapter includes an overview of the thesis in its entirety, a clear statement of the research objectives as well as the project's scope and industrial background. The introduction concludes with a list of authored articles with brief initial descriptions, their publication status at the time of thesis submission and a short summary of main contributions.

1.1 Thesis Structure

This Industrial PhD thesis consists of a kappa and four research articles. The kappa is comprised of the following chapters:

- Chapter 1: Introduction
- Chapter 2: Background and Related Work
- Chapter 3: Proposed Methods

After the kappa, each article is included in its entirety, with the addition of a short prologue. The thesis concludes with a final chapter summarizing the scientific findings, related challenges and proposed directions for future research.

1.1.1 What Will You Read?

In the next section I present the research objectives, followed by a section devoted to the industrial context, where the project's first key deliverable is discussed. This deliverable comes in the form of an open-sourced dataset of over 10,000 human-made product catalogs[1], consisting of more than 1.5 million individual product offers grouped into a quarter million of catalog pages. This data set is referred to as PROCAT. Given this context, an initial summary of included publications, their status and main contributions follows.

Chapter 2 is devoted to the theoretical background and related lines of research within neural-networks-based machine learning. A simplified abstraction of the industrial problem is defined in order to clearly highlight the aspects which this work addresses. Focus is given to the difficulties of set representation learning, permutation prediction, neural clustering and cluster ordering approaches. Each facet of the underlying challenge is discussed within the larger context of the scientific state-of-the-art. Subsequently, in Chapter 3 the proposed contributions are introduced in more detail, marking the end of the kappa.

With this framing firmly established, the four publications constituting the main body of the thesis are included in subsequent chapters, each preceded by a short prologue providing additional, updated context. Chapter 4 opens with a literature review outlining the neural network approaches to learning set representations and set-to-sequence mappings. Chapter 5 contains the article introducing the PROCAT dataset and early benchmarks thereon. The third publication, included in Chapter 6, expands upon these early benchmarks and proposes the *Set Interdependence Transformer* (SIT), a neural network module designed to more effectively encode higher order interaction between elements of input sets. This module is then incorporated into larger set-to-sequence model architectures and tested on both synthetic and real-

[1]Product catalogs are also sometimes referred to as product *catalogues*, brochures of flyers.

world datasets, including a simplified version of PROCAT, which does not require the prediction of an input-dependent number of product offer clusters.

In Chapter 7 the fourth and final article is included, which tackles the Catalog Problem by combining set encoding, neural clustering and set-to-sequence permutation learning. The proposed, combined model architecture, named *Neural Ordered Clusters* (or NOC for short) is demonstrated to be capable of learning higher-order compositional and structural rulesets and exhibits state-of-the-art performance on a number of datasets, including PROCAT. The thesis concludes in Chapter 8, with a summary of presented work, related challenges and future research directions.

1.2 Research Objectives and Scope

This work aims to answer two interconnected research questions:

1. How can we use neural networks to predict ordered, partitional clusters from sets of elements in a supervised manner?

2. How can we apply these methods to the problem of predicting such product catalog structure from sets of available product offers?

The work focuses primarily on deep learning, neural-network-based methods. The "catalog structure" mentioned in the second research question is defined in full detail in Chapter 2. The term refers to either a permuted order of a given set of product offers or its ordered, partitional clustering [2]. When in doubt about the exact meaning of a term, please refer to the Glossary of Terms (on page xvii). The second research question translates the general methods, developed through the first research question, into the domain of product catalog structure prediction. Emphasis is placed on being able to handle sets of varying cardinality, both with regards to the initial input set and when predicting a varying, input-dependent number of partitional clusters, which are themselves sets.

1.3 Industrial Context

This section introduces the company which co-funded the PhD project, along with related business-domain aspects, available data and other industrial considerations of relevance. Significant attention is given to the PROCAT dataset, which underpins the majority of presented research and provides the supervision targets.

1.3.1 Tjek A/S

Tjek A/S, previously known in the Scandinavian markets as either eTilbudsavis or ShopGun, is an e-commerce company that aggregates and presents retail product offers to consumers. It has over a million active monthly users in the Nordic markets and is present in 5 other EU states. This presents the company with significant scale-related automation challenges while attempting to stay true to its core value of putting people first.

Tjek's business model allows people to search for available product offers from any sector and view the corresponding catalogs, composed by the retailers in-house. There is no additional in-app advertising and all retailers are enabled to have their content available in the app on equal terms. The business is also environmentally focused - according to the Danish Ministry of Environment and Food between 2010 and 2016 fewer paper catalogs resulted in the savings of 86,533.3 tons of paper [3]. Given that in that period Tjek's app has had over 2 million downloads, which is $\frac{2}{3}$ of the overall market presence in Denmark and assuming even the highest standard of paper recycling, this could amount to a life-cycle carbon footprint reduction of 470 million kg [4], which would in turn make up 2% of Denmark's initial commitment to the Kyoto protocol [5]. The presented work can further help foster the transition from print to digital catalogues, reducing paper waste [6].

The creation of a catalog's structure in the context of e-commerce product catalogs is a time- and resource-consuming process, involving marketing and branding experts

and requiring the retailers to balance inter-department interests, overall sales profits and end-user preferences. Being able to generate structured catalogs from any subset of available product offers democratizes the process of browsing for products, potentially enabling people to have access to the product offers because they are relevant to them and not because they maximize retail sales. This is particularly valuable to businesses that have products to sell, but cannot invest in the manual design of readable, visually appealing catalogs. Thus, Tjek A/S has great interest in developing software capable of democratizing and accelerating the process of creating product catalogs that provide relevant information in an engaging, sequential reading format from provided sets of product offers.

1.3.2 Incito

In order to provide this functionality, the Engineering Department of Tjek A/S has developed a software service named Incito, which defined the input and output of our neural networks and guided the research in the initial set-to-sequence direction. Bridging the gap between software-as-a-service (SaaS) and a novel, simpler data-serialization format designed specifically for product presentation, Incito is capable of rendering catalog structures on a wide range of devices and screen sizes, including smartphones, tablets, and desktop computers. This affords it great flexibility over existing PDF viewers, which constituted the majority of the previous generation of on-device product catalog viewers in Denmark.

Incito, in its most basic form, takes as input a simple nested list of product offer identifiers. The top-level list represents the catalog in its entirety as an ordered sequence of sections. These sections are the Incito equivalent of the pages of paper-based catalogs. Each section sub-list consists of a number of product offer IDs, linked to their textual, tabular and visual properties. In its most rudimentary version, Incito's internal logic calculates the appropriate in-section placement of product offers depending on their corresponding images and the amount of screen space available,

whilst preserving the specified order of sections. Thus, the challenge for this project is to take a set of product offers, group them into sections and predict the order of these sections to provide a viewing experience (which is to be learned from the training data). A visual explanation of Incito's input and output is given in Figure 1.1.

This simplified abstraction of the underlying industrial challenge is further referred to as the eponymous **Catalog Problem**, defined fully in Chapter 2.1. It requires our models to take as input variable-cardinality sets of elements and output a partitional clustering of these elements, where each cluster is assigned an order within their final sequence. The proposed neural methods must be able to learn to predict ordered clusters from available data in a *supervised* manner. Predominant difficulties inherent to the Catalog Problem relate to learning permutation invariant representations of sets that encode higher-order interactions between the sets' elements, predicting a varying number of partitional clusters that abide by implicit constraints and managing combinatorial complexity.

1.3.3 PROCAT

PROCAT is a dataset of product catalogs and the main source of supervised data for this project [1]. It contains full product catalogs designed by human experts within internal departments of individual retailers and displayed within Tjek's apps. This dataset consists of 11,063 human-designed catalog structures, made up of 1,613,686 product offers with their text features, grouped into a total of 238,256 sections. The dataset's diversity stems from the catalogs covering 15 different GPC-GS1 commercial categories [7] and from their original composition being created by 2398 different retailers, including cross-border shops that have a significant following in Denmark and neighboring Scandinavian countries, particularly Sweden and Norway, as well as Germany. For an overview of the categories, see Table 1.1.

The data was collected within the full 4 year period between 2015 and 2019. The original structure of each catalog is preserved by retaining information about which

Figure 1.1: **Incito**. This proprietary rendering service defines the output of the models. Incito expects an ordered list of sub-lists, each sub-list consisting of product offers (top). Each product offer is represented here in the Incito input by a descriptive identifier (e.g. `broccoli_1`). Predicted nested lists are passed to the Incito service, which renders them into readable, visually appealing electronic catalog pages (bottom), here in the form of three consecutive screen captures. Order of sub-lists is respected, order of elements within a sub-list is ignored in favour of utilizing all available screen space. Colours and other visual elements are determined by the Incito service heuristically.

Category	Number of Catalogs	Percentage (total)
Food (FBT)	7,456	67.40%
Electronic	5,231	47.28%
Personal Care	5,113	46.22%
Tools	3,311	29.93%
Sports Equipment	2,147	19.41%
Lawn/Garden Supplies	2,039	18.43%
Home Appliances	2,028	18.33%
Baby Care	1,986	17.95%
Household Furniture	1,672	15.11%
Pet Care	1,522	13.76%
Footwear	1,324	11.97%
Toys and Games	1,293	11.69%
Fuels	548	4.95%

Table 1.1: GPC-GS1 commercial categories of the catalogs present in the PROCAT dataset. The given percentages do not add up to a hundred because most catalogs fall into multiple GPC categories. The predominant category is FBT, which stands for food, beverages and tobacco, covering the majority of grocery shopping.

product offers were presented together on which page, what the order of pages was and through a separate feature referred to as *priority class*, which represents the relative size of the corresponding product offer's image on the page in the original catalog. A visual representation is given in Figure 1.2. The data was acquired by Tjek A/S through its proprietary system for extracting product offers from any PDF. This system reads the feeds and scrapes a list of stores and PDF catalogs associated with said stores. Afterwards, the operations department performs a human curation step to make sure the obtained data is correct. The diversity of the dataset is limited due to the product offer text being in Danish. The intention was to provide a valuable resource for an underrepresented language.

The dataset consists of instances representing 3 types of entities. The most atomic entity is a *product offer*, which represents a specific product with a text heading and description, which in most cases includes its on-offer price. The individual product

Figure 1.2: **Paper Product Catalogs**. Examples of real-world catalogs designed by human experts, taken from the provided PROCAT dataset. Each row consists of three consecutive pages, each page sequence is selected from a different product catalog. Such page compositions and their order within the larger structure of catalogs forms the training data and supervision target for the presented models.

offers are then grouped into *sections*, which represent pages in a paper product catalog (PPC). Finally, an ordered list of sections comprises a single *catalog*. The original set of product offers that comprised such a catalog forms the input from which one must predict the original catalog's target structure, as per the Catalog Problem. In the simpler, **incomplete approach** to this task the target number of sections is included in the input as special section-break tokens. The task then takes the form of permuting the input set into an ordered sequence, with the tokens marking the start and end of a section. This incomplete approach is therefore a set-to-sequence approach, which includes prediction of in-section order of product offers that is ignored by the Incito service. In the **complete approach**, it requires clustering the input set of elements into a predicted number of partitional clusters and permuting them into the target order of sections. The complete approach combines clustering and permutation learning. Both approaches are explained in detail in Chapter 2.1.

Each product offer instance consists of its unique id, its related section and catalog ids, a text heading and description in both raw form and as lowercase word tokens obtained via the NLTK tokenizer [8], the total token count, and finally the full product offer text as a vector referencing a vocabulary of the most common 300 thousand word tokens. Additionally, each product offer is categorized into a priority class, representing how visually prominent it was in the original catalog in terms of relative image size (on a 1-3 integer scale). However, the prediction of the priority class is not tackled in this work. For examples of product offers in PROCAT, see Table 1.2.

Each catalog instance consists of its unique id, an ordered list of associated section ids, and a list of product offer ids that comprise the catalog in question. Additionally, each catalog instance also includes information in the form of ordered lists of sections, each containing a list of product offers as vectors, with their corresponding priority class and the catalog's length as the total number of product offers within it. Finally, a matrix of product offer vectors ($\mathbf{X}_\pi \approx \mathbb{X}$) is provided for each catalog, along with the target ($\mathbf{y}$) required to restore the original order and composition of sections.

section	heading	description
1	Lamb chops	Approx. 400 grams. Marinated chops with mushrooms, bacon. Best served with cream.
1	Ham roast	700-800 grams. Oriental. Mexico.
1	Melon	Organic piel de sapo or cantaloupe melon. Unit price 20.00. Spain, 1st class.
2	Hair spray	ELNETT. Extra strong. Strong hold. 400 ml.
2	Deodorant	Spray. Roll-on. 50-150 ml. REXONA

Table 1.2: Five examples of product offers from PROCAT, each with its raw text features consisting of a heading and description. These examples are translated from Danish into English. Left column specifies which section each offer was placed on within a single catalog. In this context a section corresponds to a single page of a paper product catalog that each offer was placed on.

Every catalog instance consists of both raw data and preprocessed features. The dataset is not a sample. It contains all catalog instances from the years 2015 - 2019 available for viewing in the Tjek A/S app. No other selection filter was used. The URL to access the dataset is provided at https://doi.org/10.6084/m9.figshare.14709507. The data is made publicly available under the Attribution-NonCommercial-ShareAlike 4.0 International license (CC BY-NC-SA 4.0). All explanations on how to read the dataset, with examples, are provided via jupyter notebooks as part of the code repository for repeated experiments at https://github.com/mateuszjurewicz/procat.

An important limitation of PROCAT and learning from human-made product catalogs in general is that we only have access to one canonical ordering of the product offer instances, whereas it is possible that other, equally valid catalogs can be constructed from the same input set of product offers. In order to mitigate this a synthetic dataset library is provided, where many valid targets are available for each input. These synthetic catalogs are introduced in more detail in Chapter 3.1.

1.3.4 Other Industrial Considerations

There are certain aspects of the industrial challenge, stemming from informal domain expertise, that guided the direction of the presented research. These include a focus on

relational reasoning, on handling inputs sets of varying cardinality and on prioritizing attention-based permutation learning approaches over ranking methods from the field of information retrieval, among others.

Firstly, the emphasis on relational reasoning stems from interviews with subject matter experts (SMEs) responsible for designing product catalogs. In these conversations, the SMEs highlighted multiple examples in which the presence or absence of multiple specific product offers determines the choice of section members and the order of sections. An illustrative case is given by a catalog that includes a product offer advertising a prime cut of Venison and another product offer for La Bonne Vie double-cream Brie cheese. In isolation, these two product offers do not necessarily make for a complementary pair that would fit on a well-composed section. However, with the addition of a product offer for a bottle of red wine, such as 2007 Cabernet Sauvignon, to the original set of available product offers, the triplet forms a popular French-cuisine-inspired catalog page. Other work in this area supports the assumption that there are many such pairwise and higher-order interactions, which any proposed model must be capable of encoding as they determine the target catalog structure [9].

Secondly, there is great variety among the available catalogs in terms of length. Certain DYI catalogs, often published only once per quarter, consist of almost a thousand individual product offers. By contrast, grocery-based catalogs, which form the majority of the provided PROCAT dataset, tend to consist of one to two hundred product offers. There are similar trends within each GS1-GPC category. This necessitates the ability to handle input sets of varying cardinalities.

Thirdly, the intent of the SMEs is to provide an engaging catalog *narrative* that inspires the reader to complete a purchase. What this means in practice is that product offers within a catalog tend not to conform to a section grouping by product category. A catalog may contain discount product offers on fruits and vegetables, but these will be spread across multiple, non-consecutive sections. This consideration puts traditional ranking approaches at a disadvantage, seeing how the proper catalog

structure is not determined by a single relevance score per product offer. Furthermore, learn-to-rank applications often require the existence of a query for which such per-element relevance is calculated, having originated in the field of information retrieval.

Finally, an important difficulty pertinent to the challenge offered by PROCAT is the existence of a higher number of reasonable substitutes for each product offer on a given section within the entire input set of initially available products. This difficulty becomes harder to surmount as the cardinality of the input sets increases to tens and hundreds. In other words there may exist multiple other equally valid catalogs that could have been constructed from the same underlying set of product offers but we only have access to a single target sequence of sections. As an interesting aside, in practice the composition and structure of catalogs is commonly defined prior to the exact product images being available.

1.4 Publications and Status

The articles included in this thesis are as follows:

1. Mateusz Jurewicz and Leon Derczynski. *"Set-to-sequence methods in machine learning: a review"*. Published in the Journal of Artificial Intelligence Research, Volume 71: 885-924, JAIR 2021.

2. Mateusz Jurewicz and Leon Derczynski. *"PROCAT: Product Catalogue Dataset for Implicit Clustering, Permutation Learning and Structure Prediction"*. Published in the Proceedings of the Thirty-Fifth Conference on Neural Information Processing Systems, Datasets and Benchmarks Track, NeurIPS 2021.

3. Mateusz Jurewicz and Leon Derczynski. *"Set Interdependence Transformer: Set-to-Sequence Neural Networks for Permutation Learning and Structure Prediction"*. Published in the Proceedings of the 31st International Joint Conference on Artificial Intelligence, IJCAI-ECAI 2022.

4. Mateusz Jurewicz, Graham Taylor and Leon Derczynski. *"Clustering and Ordering Variable-Sized Sets: The Catalog Problem"*. Under review as a conference paper at the Eleventh International Conference on Learning Representations, ICLR 2023.

The first published article is a literature review in the field of set-to-sequence deep learning. It introduces foundational concepts and expounds upon the most prominent neural network (NN) methods of encoding sets, ranging from the initial Deep Sets [10], through the still popular Set Transformer [11] to more recent Featurewise Sort Pooling [12], AttSets [13] and RepSet [14]. Many of these methods form the baselines for later experimental work included in the thesis. The first publication also provides an overview of model architectures created to map from sets onto permutations of the sets' elements, such as the original Pointer Network [15] and its enhanced adaptation by some of the same authors [16] which conditions its prediction on a permutation invariant representation of the input set: the Read-Process-and-Write (RPW). It also includes a longer discussion of other NN methods for predicting a permutation of set elements. These include permutation matrices and learn-to-rank approaches.

The second published paper introduces the PROCAT dataset and an initial approach to the Catalog Problem as a pure set-to-sequence task, without having to predict an input-dependent number of sections. Instead a varying number of section-break tokens is provided as part of each input set, the challenge being to place them in the proper place in the final permutation. Additionally, early benchmark performance is reported and a library for generating synthetic catalog structures in adherence to customizable, flexible rulesets is displayed. This synthetic data is used in later work to provide insights into the compared models' exhibited capacity to learn compositional and structural rules that determine the target shape of the synthetic catalogs.

The third published article introduces an enhanced modular method for learning representations of sets. Named the Set Interdependence Transformer (SIT), it takes

inspiration from the popular Set Transformer [11] and adds a mechanism for attending directly to an encoding of the entire set. This modification appears to improve the full model's ability to learn structural rules dependent on higher-order interactions between the set elements. Specifically, the SIT module is used as the set encoding method and combined with a permutation learning module in the form of an Enhanced Pointer Network [17]. This joint set-to-sequence model is then tested on a variety of tasks, ranging from the Travelling Salesman Problem on a 2D Euclidean Plane [18], through formal grammars and a version of the Van Dyck language [19], to sentence ordering and the incomplete, set-to-sequence formulation of PROCAT.

The fourth and final paper tackles the more challenging, complete approach to the PROCAT dataset. This requires the proposed model to be able to predict the target number of sections into which the available set elements should be split. Therefore, focus is given to neural clustering methods and the possible ways of adapting them to output not just a partitional clustering but also the order of predicted clusters. The paper proposes such an architecture, referred to as Neural Ordered Clusters (NOC) and enhances it with a mechanism for learning cluster cardinality constraints, which contributed to achieving good performance on both synthetic datasets and PROCAT.

1.5 Contributions

The main research contributions of this PhD project are as follows:

1. A **literature review** consolidating the field of modern set-to-sequence machine learning, providing a comprehensive entry point for computer scientists interested in this subdomain. The review covers set encoding and permutation learning methods as well as a qualitative comparison thereof, in order to enable researchers to be easily guided to the methods best suited to their purpose.

2. A large, curated **dataset of real-world product catalogs**, consisting of over 1.5 million individual product offers composed into 10,000 catalogs. PROCAT

is made freely available under a CC BY-NC-SA license, spans 15 GPC-GS1 product categories and to the best of my knowledge remains as the only publicly available dataset lending itself to the learning of the structure of electronic product catalogs. It also provides a resource in an underrepresented language.

3. A library for generating simplified, **synthetic catalog structures**, according to an adjustable set of rules. This enables procedural generation of datasets supplementing PROCAT, enabling faster training and more fine-grained evaluation through specific, per-rule, compositional and structural metrics. Additionally, this addresses the limitation of PROCAT by providing multiple valid catalogs for the same underlying input. Benchmarks are provided for both datasets.

4. The **Set Interdependence Transformer** (SIT) - a set encoding neural network module capable of effectively learning higher-order interactions among elements of sets of varying cardinality. SIT can be easily plugged into a modularized set-to-sequence architecture and its performance is demonstrated in such a setting on PROCAT and other datasets, both real and synthetic.

5. A complete **set-to-sequence model** outperforming state-of-the-art methods on established datasets and within the application domain of catalog structure prediction on both real-world and synthetic datasets. This model utilizes SIT as its set encoder and exhibits performance empirically supporting its ability to learn n^{th} order relational rules within fewer than n layers.

6. The introduction and **definition of the Catalog Problem**, a novel joint clustering and cluster ordering problem over sets of elements, which is a challenging variant of the set-to-sequence domain with multiple aspects that are not handled by existing neural methods. This problems is exemplified and tackled on multiple datasets, through a robust comparison of existing and proposed neural methods, providing insights into the models' capacity to learn higher-order

relational rules of cluster composition and ordered structure.

7. A combined neural network model capable of predicting ordered partitional clusters from sets of any cardinality, the **Neural Ordered Clusters** (NOC). Blending set encoding, neural clustering methods and pointer attention mechanisms, NOC learns to output an input-dependent number of partitional clusters from variable sets as well as their permutation, in a supervised manner. Additionally, it is further enhanced with a module for learning cluster cardinality constraints, which proved vital to tackling the Catalog Problem.

References

[1] M. Jurewicz and L. Derczynski, "PROCAT: Product catalogue dataset for implicit clustering, permutation learning and structure prediction," in *Thirty-fifth Conference on Neural Information Processing Systems. Datasets and Benchmarks Track*, 2021.

[2] X. Jin and J. Han, "Partitional clustering," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 766–766, ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_631. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_631.

[3] Miljøministeriet, *Danskernes papirforbug i kraftigt fald*, Miljøstyrelsen, Ed., Dec. 2018. [Online]. Available: https://mst.dk/service/nyheder/nyhedsarkiv/2018/sep/danskernes-papirforbrug-i-kraftigt-fald/.

[4] K. Dixon, *How much do you save in carbon emissions by being a good energy customer?* GoodEnergy, Ed., Sep. 2020. [Online]. Available: https://www.goodenergy.co.uk/good-stats-on-carbon-saving/.

[5] UN, "Kyoto protocol," *United Nations Framework Convention on Climate Change. Available online: http://unfccc. int/kyoto_protocol/items/2830.php*, 1997.

[6] S. Wirtz-Brückner and E.-M. Jakobs, "Product catalogs in the face of digitalization," in *2018 IEEE International Professional Communication Conference (ProComm)*, IEEE, 2018, pp. 98–106.

[7] H. M. Zahera and M. Sherif, "Probert: Product data classification with fine-tuning bert model.," in *MWPD@ ISWC*, 2020.

[8] S. Bird, "NLTK: The natural language toolkit," in *NLTK: The natural language toolkit*, Jan. 2006. DOI: 10.3115/1225403.1225421.

[9] Y. C. Xu, S. Cai, and H.-W. Kim, "Cue consistency and page value perception: Implications for web-based catalog design," *Information & Management*, vol. 50, no. 1, pp. 33–42, 2013.

[10] M. Zaheer, S. Kottur, S. Ravanbhakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 3394–3404.

[11] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh, "Set transformer: A framework for attention-based permutation-invariant neural networks," in *International conference on machine learning*, PMLR, 2019, pp. 3744–3753.

[12] Y. Zhang, J. Hare, and A. Prügel-Bennett, "Fspool: Learning set representations with featurewise sort pooling," in *International Conference on Learning Representations*, 2019.

[13] B. Yang, S. Wang, A. Markham, and N. Trigoni, "Robust attentional aggregation of deep feature sets for multi-view 3d reconstruction," *International Journal of Computer Vision*, vol. 128, no. 1, pp. 53–73, 2020.

[14] K. Skianis, G. Nikolentzos, S. Limnios, and M. Vazirgiannis, "Rep the set: Neural networks for learning set representations," in *International conference on artificial intelligence and statistics*, PMLR, 2020, pp. 1410–1420.

[15] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *Advances in neural information processing systems*, vol. 28, 2015.

[16] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," *arXiv:1511.06391*, 2016.

[17] Y. Yin *et al.*, "Enhancing pointer network for sentence ordering with pairwise ordering predictions," in *AAAI*, vol. 34, 2020, pp. 9482–9489.

[18] K. H. Hansen and J. Krarup, "Improvements of the held—karp algorithm for the symmetric traveling-salesman problem," *Mathematical Programming*, vol. 7, no. 1, pp. 87–96, 1974.

[19] X. Yu, N. T. Vu, and J. Kuhn, "Learning the dyck language with attention-based seq2seq models," in *ACL Workshop BlackboxNLP*, 2019, pp. 138–146.

Chapter 2

Background and Related Work

This chapter defines the eponymous Catalog Problem and describes in greater detail what is meant by *catalog structure*, as referenced in Section 1.2 and the Glossary of Terms. To facilitate this, the chapter introduces important concepts from the fields of set encoding, permutation learning and neural clustering. As a consequence, an initial synopsis of the rationale behind certain choices with regards to the directions of the scientific investigation is given (and expanded on in Chapter 3). This chapter also interweaves descriptions of related work, intending to frame the research presented in the next chapter within the state-of-the-art. Additionally, a wider discussion of similar challenges and related approaches is provided.

2.1 The Catalog Problem

This Chapter defines the Catalog Problem as an umbrella term for challenges that require:

1. taking as input varying-cardinality sets,
2. predicting an input-dependent number of ordered, partitional clusters,
3. in accordance with a supervision target.

Many important real-world challenges can be framed in this way, from supply chain management [25] to prioritization in medical triage [26]. Other areas of application

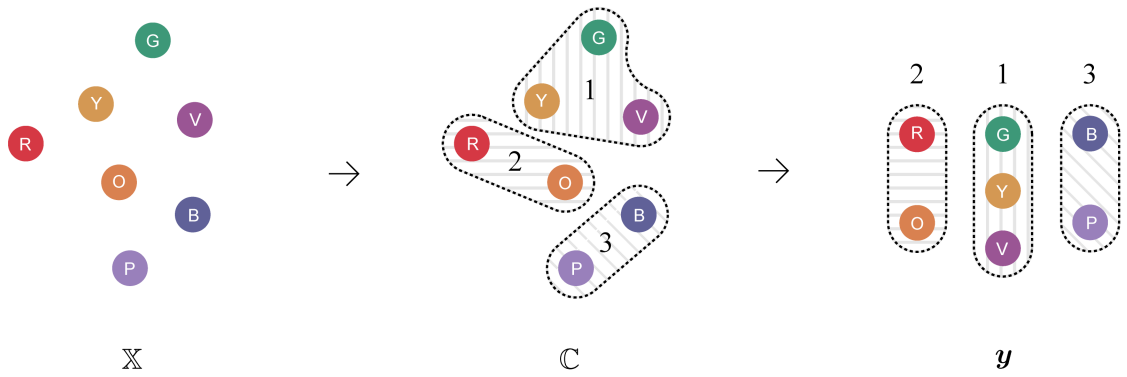


Figure 2.1: **The Catalog Problem**. From left to right: a set of input elements ($\mathbb{X}$); target partitional clustering of those elements ($\mathbb{C} = \{\mathbb{C}_1, \mathbb{C}_2, \mathbb{C}_3\}$); and a target ordering over those clusters ($\mathbf{y} = (\mathbb{C}_2, \mathbb{C}_1, \mathbb{C}_3)$), left to right. The targets presented here were chosen arbitrarily for the purposes of this demonstration, in used datasets they contain learnable patterns. A candidate model may perform all these tasks using information about inter-element relations and intra-cluster relations in order to characterise a cluster, and inter-cluster relations to generate the final, ordered clustering. Such a method, capable of predicting an input-dependent number of ordered partitional clusters, constitutes a **Complete Approach** to the Catalog Problem.

include petroleum exploration [27], business process analytics [28] and the domain of product catalog structuring [29], which was the focus of presented research and inspired the name. For a visual introduction, see Figure 2.1.

More formally, in the Catalog Problem the input is an unordered set of unique elements $\mathbb{X} = \{x_1, \ldots, x_n\}$, with a varying cardinality $n = |\mathbb{X}|$. The target output is a sequence of partitional clusters $\mathbf{y} = (\mathbb{C}_1, \ldots, \mathbb{C}_k)$, where k determines the number of clusters, and each cluster is itself a set defined by the elements assigned to it. The number of elements assigned to a cluster can differ per cluster. All elements must be assigned to a cluster and an element can only belong to one cluster (hence the use of the term partitional [2]). Empty clusters are not allowed.

For example, given an input set $\mathbb{X} = \{x_1, x_2, x_3, x_4, x_5\}$ the target can take the form of the ordered sequence $\mathbf{y} = (\{x_3, x_4\}, \{x_1\}, \{x_2, x_5\})$. This example requires a prediction matching the following target set of clusters: $\mathbb{C} = \{\mathbb{C}_1, \mathbb{C}_2, \mathbb{C}_3\}$, such that if $\mathbb{C}_1 = \{x_1\}$, $\mathbb{C}_2 = \{x_3, x_4\}$ and $\mathbb{C}_3 = \{x_2, x_5\}$ then $\mathbf{y} = (\mathbb{C}_2, \mathbb{C}_1, \mathbb{C}_3)$.

When implemented in code, the clustering target ($\mathbb{C}$) can be represented by a se-

quence of *cluster assignments*: $(c_1, \ldots, c_n)$, which are per-element integer identifiers that denote each element (x_i) belonging to one of k clusters via that cluster's arbitrarily assigned identifier between 1 and k ($x_i \in \mathbb{C}_j$ is equivalent to $c_i = j$).

For the example above, the target cluster assignments would be $\mathbf{c} = (1, 3, 2, 2, 3)$. This particular $\mathbf{c}$ indicates that the element x_i should be assigned to cluster $\mathbb{C}_1$, the element x_2 to cluster $\mathbb{C}_3$ and so forth. Note that the integers that identify clusters do not correspond to their target order. This convention becomes useful when discussing models that first iteratively predict clusters one at a time and then order them all into the final predicted sequence $\hat{\mathbf{y}}$. For such a model the predicted cluster $\hat{\mathbb{C}}_1$ is identified by the integer identifier 1 to signify that it was the first predicted cluster. The "hat" sign ($\hat{}$) is used to distinguish between targets $(\mathbf{c}, \mathbb{C}, \mathbf{y})$ and predictions $(\hat{\mathbf{c}}, \hat{\mathbb{C}}, \hat{\mathbf{y}})$.

In a neural network context the problem can be framed as learning a parameterized function ρ_θ, whose input is a set of varying cardinality and whose output is a sequence of k partitional clusters, such that $\rho_\theta(\mathbb{X}) = \hat{\mathbf{y}}$, with k being predicted indirectly. In practice, the input set is represented as the matrix $\mathbf{X}_\pi \approx \mathbb{X}$. The order of this matrix's column vectors, each corresponding to a single set element, is represented by a permutation π.

Neural approaches to the Catalog Problem can predict cluster assignments and cluster order separately, both in an iterative way. Predicting the cluster assignments takes the form of assigning a probability of an element x_i belonging to a cluster $\mathbb{C}_j$ as $p_\theta(\hat{c}_i = j)$. Given a vector of predicted probabilities for all unassigned elements at the j^{th} clustering step and a binary target vector obtained from $\mathbf{c}$ for the corresponding elements, with 1s marking that the element at that index should be assigned to the j^{th} cluster, a cross-entropy loss is calculated between these two probability distributions.

Predicting the order of obtained clusters as a sequence $(\hat{\mathbf{y}})$ takes the form of predicting an attention vector over all k clusters in $k - 1$ iterative steps, where at each step the highest attention points to the cluster to be placed next in the final predicted sequence $\hat{\mathbf{y}}$.

Predicted cluster assignments ($\hat{\mathbf{c}}$) and sequence of clusters ($\hat{\mathbf{y}}$) are derived from these model outputs. Given these and the targets, the quality of the clustering can be assessed through the V-measure [30] and of the cluster ordering via Kendall's Rank Correlation Coefficient (τ), as well as through certain custom metrics, as described in more detail in Chapter 3.

Challenges that epitomize the Catalog Problem often require the ability to learn higher-order interactions between input set elements [31], hence the focus on set encoding in Section 2.3. The predicted clusters must be ordered and the permutation stage must be capable of handling a varying number of input clusters, which themselves vary in sizes. In the proposed methods the focus is on pointer-style attention employed over fixed-length vector representations of predicted partitional clusters to permute them, in a supervised manner. This and other permutation methods are outlined in Section 2.4. The capacity to obtain meaningful embeddings of sets is also vital in the clustering stage, described in Section 2.5, where previously predicted clusters can influence the subsequent cluster assignments for remaining elements.

The product catalog context in particular requires learning variable cluster-level constraints, such as maximum cardinality thresholds[1], exhibiting relational reasoning and managing combinatorial complexity (product catalogs consist of hundreds and sometimes thousands of individual product offers). For the purposes of this work the product catalog structure, which we are trying to predict, takes the form of individual product offers (our input set elements) grouped into complementary sections (partitional clusters) ordered into a compelling catalog narrative (permutation of predicted clusters). However, the initial articles explore a simplified, **Incomplete Approach** to the Catalog Problem, requiring only set encoding and permutation learning, as the number of clusters is known to the model via the total count of special section-break tokens included in the input set. This is also referred to as the set-to-sequence

[1]Learned, per-cluster maximum cardinality thresholds help control the number of products presented per page, which in turn makes for a more evenly distributed catalog.

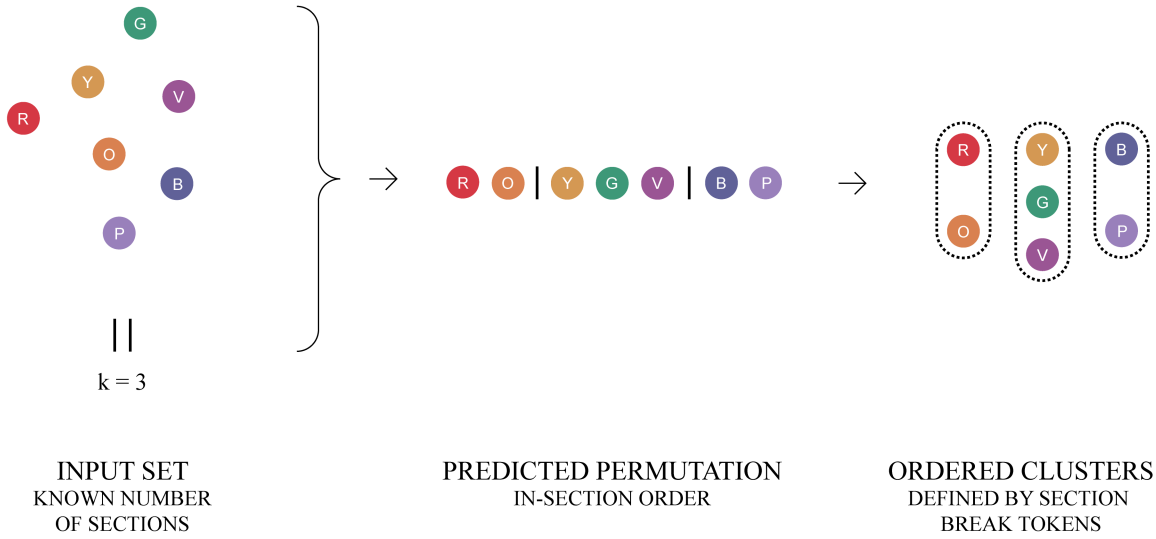


Figure 2.2: **Incomplete Approach**. In this work, two approaches to the Catalog Problem are explored. The earlier, simpler approach requires the target number of sections to be known to the model. This is shown above as $k = 3$, represented by the number of vertical section-break tokens - two in the example, one fewer than the number of sections. These are included in the input set (left). The incomplete, set-to-sequence approach also predicts meaningless in-section order as it outputs a permutation of all input set elements (middle). The in-section order is ignored by the Incito service. From this permutation, the final ordered clustering is derived (right).

approach. For a visual explanation of this distinction, see Figure 2.2.

As such, the provided sections outlining related work and important theoretical concepts begin with the topic of set encoding. As mentioned in Section 1.3.4, conversations with SMEs suggested that the composition of the input set of available product offers, the pairwise and higher-order interactions among its elements, can drastically influence the target section split. This determined the initial direction of presented research towards methods capable of taking sets of any cardinality and encoding them into fixed-length vectors. When combined with permutation learning methods such as Pointer Networks [15] into full set-to-sequence architectures, these constitute the incomplete approach to the Catalog Problem. In particular, pointer-style attention offers one specific advantage over learn-to-rank methods in that it allows for learning the kind of reading narrative SMEs defined as crucial to an engag-

ing and inspiring catalog viewing experience, as opposed to sorting products by their presumed relevance to a specific query.

However, a set-to-sequence approach cannot predict an input-dependent number of sections purely from the set of product offers (excluding section break tokens) without significant alterations. Some such modifications are explored in the fourth publication, in Chapter 7. This pointed the research in the direction of neural clustering methods, completing the list of the three main topics that are introduced in the following sections. First, however, I will discuss similar problems, the history of attempts to address them and briefly mention which related methods are considered out of scope.

2.2 Similar Problems and Related Approaches

The question of automatically generating online catalogs spans many domains, reflecting the width of the definition of the term "catalog". One example comes in the form of a significant body of research in the area of structuring and online presentation of library catalogs [32–35], where emphasis is placed on information retrieval. However, this work's focus lies specifically with electronic *product* catalogs also known as EPCs [36–40]. Ever since the advent of the online catalog, there has been significant business and scientific interest in understanding what makes for a satisfying user experience [41–44]. Substitutes of the readable, visual EPC have been developed in the form of pure-search interfaces [45] and recommendation systems [46]. However, the direction of this work was determined by the input and output specification of the Incito service outlined in Section 1.3.2, precluding the use of these alternatives.

As such, one can look to clustering methods with a notion of order, which are closer to the formulation of the Catalog Problem. Significant research has been devoted to explicitly preventing the effect of the arbitrary order of set elements in which they are presented to a clustering algorithm [47–49]. This mirrors the development of permutation-invariant NN set encoding methods described in Section 2.3. Although this setting requires *supervised* learning, one can also look to the more popular, *un-*

supervised setting, where clustering approaches with a notion of order range from hierarchical clustering [50, 51], through ordinal clustering [52] and incremental conceptual clustering [53] to Markov clustering [54] as well as other, more recent methods [55–57]. Certain unsupervised clustering methods without the ordering element, like affinity propagation [58, 59], are also capable of outputting an input-dependent number of partitional clusters.

An alternative to the set and cluster sequence formulation of the addressed catalog prediction challenge would be to represent the output structure in the form of a graph. There has been significant progress in the field of *Graph Neural Networks* (GNNs), capable of mapping graphs in the form of an adjacency matrix and node features to various target outputs [60–63]. This includes neural methods capable of encoding graphs in a permutation-invariant way [64]. Unsupervised graph representation setting has also been explored [64, 65], with applications in link prediction, node community detection and other areas [66–69]. Of particular interest is the topic of amortized probabilistic detection of communities in graphs [22], which is a supervised method taking inspiration from neural clustering models. However, in the Catalog Problem the input does not have the structure of a graph.

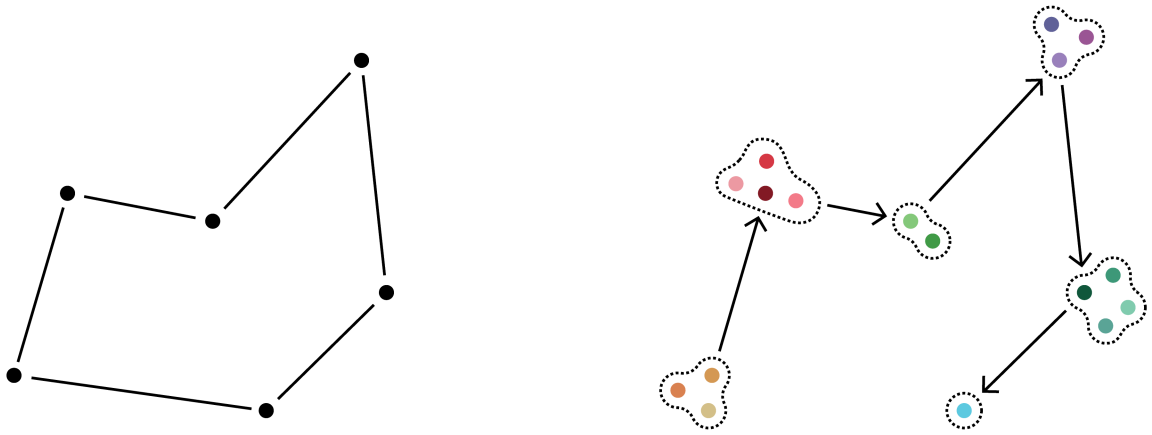
On the other hand, an unexplored but potentially viable alternative would be to represent the target output (a sequence of partitional clusters) as a graph, with some nodes representing actual input set elements and perhaps a special, secondary type of node representing a new cluster to which these elements can be linked, together forming a complete catalog section. Other possible reformulations could utilize different types of edges to the same end. Similarly the notion of the order of such node communities could be represented through directed edges. This possibility has not been granted an in-depth assessment due to time constraints, but existing work within deep generative NNs could provide the foundation for future research along these lines [70, 71]. In particular Graph Recurrent Attention Networks could provide a viable avenue of investigation for predicting ordered communities, given the

sequential nature in which they predict blocks of nodes and edges [72–74].

Another branch of DL methods that could lend itself to approaching the Catalog Problem comes in the form of Reinforcement Learning [75–77], or RL for short. Significant progress has been made in both value- [78–80] and policy-based methods [81–83], as well as combinations thereof in the form of the popular actor-critic framework [84–87]. Reinforcement learning methods have also been used in combination with permutation learning techniques such as pointer networks, which are discussed in Section 2.4. A notable example of this comes in the form of the AlphaStar agent, where set-to-sequence methods were employed to help the agent manage the structured, combinatorial action space in conjunction with an auto-regressive policy [88].

The Catalog Problem itself can be seen through the lens of combinatorial optimization problems [89, 90], of which a notable example is the *Travelling Salesman Problem* (TSP) in 2D Euclidean [91] (as shown in Figure 2.3). Combinatorial optimization challenges can be formalized as a 3-tuple $(\mathbb{X}, \psi, \omega)$, where $\mathbb{X}$ is a set of instances, from which given an instance $x_i \in \mathbb{X}$ the $\psi(x_i)$ is a finite set of feasible solutions $\hat{y}_j \in \mathbb{Y}_i$ and $\omega(\hat{y}_j)$ is a measure of such solutions, such that $\omega(\hat{y}_j) \in \mathbb{R}^+$ and the goal is to either minimize or maximize the value of ω.

RL solutions have been successfully applied to many combinatorial optimization problems formalized in this way, including the TSP, which I have also used as a demonstrative example of an NP-hard [92] challenge in set-to-sequence experiments in the second publication, in Chapter 6. TSP mirrors one aspect of the Catalog Problem, in that part of the objective is to find a target order of elements according to some measure function. In case of the TSP in a 2D Euclidean space these elements are coordinate points and, in the simplest formulation, the goal is to minimize the length of the total route travelled between all points. In case of the Catalog Problem the elements are composed clusters (catalog sections) and the measure function relates to creating a compelling, well-structured catalog sequence. Thus ω is not directly available.



TRAVELLING SALESMAN PROBLEM

THE CATALOG PROBLEM

Figure 2.3: **Travelling Salesman Problem**. The TSP bears similarities to the Catalog Problem. On the left, an example TSP tour in 2D Euclidean is given, where a full Hamiltonian Cycle must be found - the tour finishes at its starting point. To the right, an analogous depiction of a single solution to an example of a Catalog Problem is given. Here, the most optimal Hamiltonian Path between predicted clusters of elements must be found to represent their order, not returning to the initial cluster.

An initial example of an RL method outperforming set-to-sequence methods (specifically pointer networks) on the TSP was proposed by Kool *et al.* (2018). In this work, an impressive starting point for learning heuristics for combinatorial optimization problems defined on graphs has been developed, provided the solutions can be represented as sequential decisions with corresponding reward signals. Whilst the framework proposed by Kool *et al.* is policy-based and utilizes the REINFORCE algorithm [81], other works approach it from a value-based, Q-learning angle [94]. Specifically, Khalil *et al.* (2017) use a 1-step Deep Q Network [95] to tackle TSP based on graph embeddings, with a reward measure providing a learning signal after each node (as opposed to at the end of the entire tour).

Progress in applying RL to combinatorial optimization problems continues through many active branches of research [96–99], including further work by Kool *et al.* (2022) in the area of the *Vehicle Routing Problem*[2] and its variants [100, 101]. Further

[2]VRP can itself be seen as a generalization of TSP.

progress through RL has been achieved in such areas as transportation systems [102], traffic signal control [103] and robotics [104], among many others. The deciding factor behind not applying RL methods to the Catalog Problem in the scope of this work is the lack of a clear, fine-grained reward measure that could provide the learning signal at each decision step. This relates to the challenge outlined in Section 1.3.4, devoted to other industrial considerations. Namely, in PROCAT there is only one canonical order of catalog sections available as the supervision target per example, despite the possibility of there existing multiple equally valid alternatives. Evaluation of generated structures is more generally a challenging problem across many domains [105]. However, RL methods remain an area of interest for potential future work.

Thus, the most pertinent NN approach to the Catalog Problem, satisfying all technical and industrial considerations, appeared to combine elements of set representation, permutation learning and neural clustering. Each of these is given separate focus in the following sections, providing necessary theoretical background and outlining prominent related methods.

2.3 Set Encoding

For the purposes of this thesis, a set can be intuitively defined as a collection of distinct elements, without a canonical order between them [106]. The first of three main aspects of the presented work revolves around learning meaningful, fixed-length vector representations of sets of varying cardinality. These representations' meaningfulness is defined by how useful they are to solving any number of downstream tasks. Set-input ML applications tackle regression [10], classification [107], segmentation [108, 109], meta-learning [110, 111] and even set-to-set tasks within the domain of recommendation [112], image search [113] and person re-identification [114], among many others. Unsurprisingly, the encoding of the input set is also vital in both the incomplete, set-to-sequence and complete, neural ordered clustering approaches to the Catalog Problem.

There are several properties that a set encoding method should ideally possess:

1. Firstly, take as input a **set of any cardinality** and return its fixed-length representation. This condition is not met by classical feed-forward neural networks [115], requiring either a recurrent or attention-based approach.

2. Secondly, we require this encoding to be **permutation invariant**, in that it should remain identical under any arbitrary permutation π of the input sets' elements. This condition is not met by *Recurrent Neural Networks* (RNNs), which are inherently sensitive to the order of their input [116].

3. Thirdly, the vector representation of the set should encode relational information. An encoding that includes information about **higher-order interactions** among set elements is of particular importance to solving the Catalog Problem, as shown in the third included publication, in Chapter 6.

4. Additionally, one would prefer this encoding method to be computationally inexpensive, ideally maintaining $O(n)$ time complexity, given that many set-input ML challenges require taking in sets whose cardinality is in the thousands.

Formally, a set encoding function f_{se} can be specified to handle sets of any cardinality in the following way:

$$f_{\text{se}} : \mathcal{P}(\mathbb{X} \in \mathbb{R}^d) \rightarrow \mathbb{Y} \in \mathbb{R}^e, \tag{2.1}$$

where $\mathcal{P}$ is the powerset of set $\mathbb{X}$ of real-valued vectors with d dimensions, mapped by the set encoding function f_{se} onto an e-dimensional, real valued vector $\hat{\mathbf{y}} \in \mathbb{Y}$. An input set $\mathbb{X}$ of d-dimensional elements can be passed to this function in any arbitrary order π, in accordance with its cardinality $n = |\mathbb{X}|$. In practical terms, this takes the form of the order of its elements within the matrix $\mathbf{X}_\pi$ representing the actual set as

it is fed into a neural network. This set encoding function should return the same fixed-length vector given the same underlying set regardless of its permutation:

$$f_{\text{se}}(\mathbb{X}) = f_{\text{se}}(\mathbf{X}_\pi) = f_{\text{se}}(\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}) = f_{\text{se}}(\{\mathbf{x}_{\pi(1)}, \ldots, \mathbf{x}_{\pi(n)}\}) = \hat{\mathbf{y}} \quad \forall\ \pi \in \Pi^n. \tag{2.2}$$

Another property of interest to set encoding is *permutation equivariance*. In tasks where each set element has an associated target label, such that these individual labels depend on the entirety of the set, one would expect the predicted labels to remain the same per element, regardless of how the original input set is permuted. That property is referred to as permutation equivariance and it occurs often in the intermediate stages of learning per-element representations of sets, before some form of a pooling operation is applied, as discussed in the next paragraph outlining related NN methods.

Now that a better understanding of the desired properties has been established, I will introduce the progression of neural methods towards achieving them. For a deeper look into these and other set encoding methods, the reader is referred to the first included publication, in the form of a journal review in Chapter 4. The model commonly considered to be the first NN set encoder that achieves three of the four desired properties (1, 2 and 4) is the aptly named *DeepSets* by Zaheer *et al.* (2017). The permutation invariance is achieved by transforming each set element x_i in an independent and identical way into its embedded representation via a stack of feed-forward neural networks $\phi_e(x_i)$, followed by summing these representations together and further processing them using another stack of fully-connected layers ρ_s, with nonlinearities:

$$\text{DeepSets}(\{x_1, \ldots, x_n\}) = \rho_s\left(\sum_{i=1}^{n} \phi_e(x_i)\right). \tag{2.3}$$

This method can be applied to sets of any cardinality whilst maintaining $O(n)$ time complexity. Its main disadvantage stems from its inability to explicitly encode pairwise and higher-order interactions between set elements, which are lost during the summation [11]. Additionally, significant doubts have been raised regarding the limits of the representational power of the DeepSets method [117]. More precisely, the $O(n)$ computational complexity comes at the cost of the dimensionality of the latent space having to be at least equal to the cardinality of the input set n to ensure universal function approximation.

A significant improvement with regards to encoding relational information came in the form of the *Set Transformer*, proposed by Lee *et al.* (2019). The Set Transformer consists of the expected stacked multi-head self-attention layers as seen in the classic Transformer [118]. It introduces a permutation equivariant operation, the *Multihead Attention Block* (MAB), which performs the self-attention transformations over sets of any cardinality, enabling it to explicitly encode pairwise and higher order interactions between set elements (depending on the number of stacked MAB layers). It also proposes a different approach to the set-pooling operation, replacing the summation seen in DeepSets with *Pooling by Multihead Attention* (PMA), which performs the transformer-style attention between the obtained permutation equivariant representations of elements $\mathbf{E}_\pi \in \mathbb{R}^{n\times d}$ and k learned seed vectors as the matrix $\mathbf{R} \in \mathbb{R}^{k\times d}$, preserving the relational information. Whilst MAB is usually used to transform a single set ($\mathbb{X} \approx \mathbf{X}_\pi$) to get the embeddings of all its elements ($\mathbf{E}_\pi$), it actually takes two arguments. In most cases these will be two copies of the matrix representing the same set, but for clarity in the equations below a distinction is made between them as $\mathbf{X_1}$ and $\mathbf{X_2}$:

$$\mathbf{E}_\pi = \mathrm{MAB}(\mathbf{X}_1, \mathbf{X}_2) = \mathrm{LN}(\mathbf{H} + \rho_s(\mathbf{H})), \tag{2.4}$$

$$\text{where } \mathbf{H} = \mathrm{LN}(\mathbf{X}_1 + \mathrm{MHA}(\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_2)), \tag{2.5}$$

$$\text{where MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{concat}(\mathbf{O}_1, \ldots, \mathbf{O}_h), \tag{2.6}$$

$$\text{where } \mathbf{O}_j = \text{ATT}(\mathbf{Q}\mathbf{W}_j^Q, \mathbf{K}\mathbf{W}_j^K, \mathbf{V}\mathbf{W}_j^V), \tag{2.7}$$

$$\text{where ATT}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{Q}\mathbf{K}^\top)\mathbf{V}, \tag{2.8}$$

where the transformer-style attention transformation (ATT) takes advantage of three learned weights matrices ($\mathbf{W}_j^Q$, $\mathbf{W}_j^K$, $\mathbf{W}_j^V$) to cast the matrices $\mathbf{X}_1 = \mathbf{X}_2$ representing the input set into its query, key and value representations ($\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$). This operation is repeated for each j^{th} head ($j \in \{1, \ldots, h\}$) as part of multi-head intra-set attention (MHA), as defined by Vaswani *et al.* (2017), without positional encoding. These operations are incorporated into the multihead attention block (MAB) with the inclusion of a row-wise feed-forward neural network (ρ_s), with layer normalization (LN) after each block [119], resulting in permutation equivariant[3] per-element representations $\mathbf{E}_\pi$. These are then aggregated into a permutation invariant representation $\mathbf{s}$ of the entire set by performing pooling via the PMA:

$$\mathbf{s} = \text{PMA}(\mathbf{E}_\pi) = \text{MAB}(\mathbf{R}, \mathbf{E}_\pi), \tag{2.9}$$

resulting in an encoding of the entire set $\mathbf{s} \in \mathbb{R}^{k \times d}$. In most applications the chosen number of learned seed vectors that comprise $\mathbf{R}$ is one, hence $\mathbf{s}$ becomes a fixed-length vector representation of the entire set. The combination of a MAB followed by PMA is referred to as a *Set Attention Block* (SAB), as shown in Equation 2.10.

$$\text{SAB}(\mathbb{X}) = \text{PMA}(\text{MAB}(\mathbf{X}_\pi, \mathbf{X}_\pi)). \tag{2.10}$$

[3]For a formal proof, see section 3.1 and supplementary material of Lee *et al.* (2019).

Formulated in this way, the Set Transformer satisfies 3 of the 4 desired properties (1, 2 and 3), with the SAB operations requiring quadratic time $O(n^2)$ due to the self-attention mechanism. However, the authors also propose a way to mitigate this challenge and achieve all four properties, namely the *Induced Set Attention Block*. ISAB requires defining a set of m d-dimensional vectors $\mathbf{k}_{\text{in}} \in \mathbb{R}^d$, referred to as the inducing points. These vectors are trainable parameters of the ISAB layers, performing the following sequence of operations:

$$\text{ISAB}_m(\mathbf{X}_\pi) = \text{MAB}(\mathbf{X}_\pi, \mathbf{H}) \in \mathbb{R}^{n \times d}, \tag{2.11}$$

$$\text{where } \mathbf{H} = \text{MAB}(\mathbf{K}_{\text{in}}, \mathbf{X}_\pi) \in \mathbb{R}^{m \times d}. \tag{2.12}$$

As seen above, ISAB first transforms the matrix $\mathbf{K}_{\text{in}} \in \mathbb{R}^{m \times d}$ (containing m inducing point vectors) into an intermediate representation $\mathbf{H}$ by attending to the input set $\mathbb{X} \approx \mathbf{X}_\pi$. These transformed inducing points are again attended to by the input set to produce a permutation equivariant matrix of n vectors. This operation has been compared to low-rank projection and to autoencoder models. It allows the time complexity to be reduced to $O(nm)$, with m serving as a hyperparameter balancing the model's representational power and training speed.

The Set Transformer's effectiveness resulted in numerous adaptations and variants, such as *Latent Variable Sequential Set Transformers* for multi-agent motion prediction by Girgis *et al.* (2021), the *Point Transformer* for semantic scene segmentation on point clouds with positional encoding by Zhao *et al.* (2021), and the Set Interdependence Transformer (SIT), introduced in the third included publication in Chapter 6, among others [122–125]. Other notable NN set encoding methods, partially elaborated on in the journal review in Chapter 4, include *RepSet* and its approximate version *ApproxRepSet* (which obtain permutation invariance through bipartite graph matching) by Skianis *et al.* (2020), *AttSets* by Yang *et al.* (2020), and *Generalized*

Sliced-Wasserstein Embedding by Naderializadeh *et al.* (2021). The importance of enabling neural networks to handle set-structured input data continues to fuel this active branch of research with more methods being proposed on an ongoing basis.

2.4 Permutation Learning

Once a meaningful, fixed-length encoding of the varying-cardinality input set has been obtained, one can proceed to condition the predicted output structure on it. In the incomplete, set-to-sequence approach to the task, this structure takes the form of the permutation of the input sets' elements. Whilst this does not constitute a complete approach to the Catalog Problem, as outlined in Section 2.1, that approach itself also requires the prediction of a permutation of the obtained clusters. Thus it is reasonable to discuss permutation learning, set-to-sequence methods before proceeding to the topic of neural clustering.

The earliest neural set-to-sequence model was proposed by Hopfield and Tank (1985). It introduced a constrained version of the set-to-sequence challenge, in which the input set must be of a known, fixed cardinality. However, sets of varying cardinality require different neural architectures, making it limited in application. Nonetheless, Hopfield Networks (HNs) and their adaptations still find active use in research [128–130], more recently suggesting that the attention mechanism of transformers is equivalent to the update rule of continuous-state HNs, as seen in the work of Widrich *et al.* (2020), in the field of immune repertoire classification.

Although the primary focus of this Section is on pointer-style attention mechanisms, one would be amiss if other formulations of the permutation task in NN settings had not been mentioned. An example comes in the form of *ordinal classification*[4]. Within neural ordinal classification by Cheng *et al.* (2008), a fixed number of ordinal classes is stipulated, each of which receives a predicted score. Given a single

[4]Ordinal classification is sometimes referred to as ordinal regression [132, 133]. To avoid confusion, the term *ordinal classification* is used to refer specifically to the described formulation of the prediction target, which I consider to be more reminiscent of classification tasks.

example with five input set elements and therefore five corresponding ordinal classes (as is the case, for example, in the popular sentence ordering dataset ROCStory [135]), the prediction target would be expressed in the following way:

$$\mathbf{Y} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{y}_1 & \mathbf{y_2} & \mathbf{y_3} & \mathbf{y_4} & \mathbf{y_5} \end{bmatrix}, \tag{2.13}$$

$$\mathbf{y}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{2.14}$$

where each column of the example's corresponding target matrix $\mathbf{Y}$ consists of one of the five column vectors $\mathbf{y}_{1:5}$. The i-th value in each vector $\mathbf{y}_j$ represents the NN model's target confidence (y_j^i) that this vector's corresponding input set element x_j should be placed in the i-th position or later in the output sequence. The element that should be last in the output sequence has a target vector $\mathbf{y}_j$ consisting of all 1s. This formulation requires input sets of fixed cardinality and by default allows for predictions that are self-contradictory, e.g.:

$$\hat{\mathbf{y}}_j = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \end{pmatrix}, \tag{2.15}$$

where the j-th element of the input set is predicted to be simultaneously in the first, middle and last position of the output sequence (but not the second and penultimate one). Nonetheless, neural ordinal classification methods continue to receive research interest in the settings of multi-instance learning [136] and facial age estimation [137], with various adaptations aimed at helping them preserve the global ordinal relationships between set elements, mitigating the issue exemplified in Equation 2.15. For a general overview the work of Gutiérrez *et al.* (2015) is recommended.

Another formulation of the prediction target comes in the form of *permutation matrices*. A permutation matrix is a square, binary matrix $\mathbf{M}$ that has exactly one entry $m_j^i = 1$ in each row i and column j [139]. Every other entry in $\mathbf{M}$ is equal to 0. The dimensions of a permutation matrix are defined by the cardinality of the input set or the length of the target output sequence[5]. This matrix is unimodal, in that each of its rows has a single highest value, and doubly-stochastic, in that it consists solely of nonnegative numbers, with each row and column adding up to 1.

The key property of interest in permutation matrices is that given an arbitrarily ordered sequence $\mathbf{x}_\pi$, representing an inherently unordered set $\mathbb{X}$, we can left-multiply it by a permutation matrix to obtain a reordered sequence. In the example below, the arbitrary initial permutation π takes the form of the vector $(1, 5, 3, 4, 2)$, resulting in the input sequence $\mathbf{x}_\pi$ being equal to $(x_1, x_5, x_3, x_4, x_2)$. Assuming the goal is to restore the sorted order $(x_1, x_2, x_3, x_4, x_5)$, one would wish to predict the following permutation matrix $\mathbf{M}$, such that:

$$\mathbf{M}\mathbf{x}_\pi^\top = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_5 \\ x_3 \\ x_4 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \tag{2.16}$$

To use a permutation matrix as the target output of a NN model that can be trained through gradient-based optimization, a relaxation of the concept must be utilized. Barring such relaxation, one is left with the equivalent of a sorting operator, which is non-differentiable [140]. There are many possible methods of obtaining this relaxation from the input after it has been transformed by a chosen neural network architecture, such as adding elementwise Gumbel perturbations [141], applying the Sinkhorn operator to directly sample matrices near the Birkhoff polytope [142], which

[5]Which will only differ in the case of permutations with repetition, which are considered out of scope in the context of the Catalog Problem.

is the convex hull whose points are doubly-stochastic matrices [143], or through the application of a softmax operator on a derived matrix of absolute pairwise distances between the individual input elements [140].

Depending on the specifics of the task at hand, the target matrix can be predicted in a single pass (if the length of the input and output is a constant, known prior to inference) or sequentially, row by row [144]. An interesting method employing permutation matrices has been proposed by Zhang *et al.* (2018), in which a trainable, pairwise ordering cost function is used to produce an anti-symmetric matrix $\mathbf{C}$, whose entry c_j^i represents the cost of placing the i-th element prior to the j-th. This function is parameterized by a neural network, which is then used to continuously adjust the learned permutation matrix. Referred to as a *Permutation-Optimisation* module, this has been demonstrated to perform well on number sorting, image mosaics reassembling and visual question answering, with the limiting feature of entailing cubic time complexity.

There remain two popular permutation learning NN methods that have received greater interest in the DL scientific community than the above-mentioned approaches. The first comes in the form of *listwise learn-to-rank* approaches [146], wherein a per-element relevance score is calculated conditioned on a list of available elements and a *query*, the final permutation being obtained by sorting elements by their corresponding score. Different queries should result in different permutations of the same underlying set. The terminology stems from applications in information retrieval, where the task is to rank the available *documents* (such as web pages) in order of relevance for a given query (e.g. a search term). For an overview of neural ranking methods, one can recommend the work of Mitra *et al.* (2018).

It isn't immediately obvious in the case of the Catalog Problem what should constitute the required query. One possible candidate is the permutation invariant representation of the entire set, as obtained via methods discussed in Section 2.1 and possibly learnt in a joint or sequential manner. Intuitively, the relative rank of each

element in the output sequence should depend on the entire available set. However, learn-to-rank approaches generally presuppose that two set elements have a canonical relative order between them, which is known not to be the case in the presented industrial setting, where it's the overall catalog narrative that dictates the grouping and order of elements (with multiple equally valid pairwise orderings, as outlined in Section 1.3). Listwise ranking NN methods of note include the *FATE* framework proposed by Pfannschmidt *et al.* (2018), the *Deep Listwise Context Model* (DLCM) by Ai *et al.* (2018) which combines two-stage ranking, sequential recurrent NNs and an attention-based loss function, and *SetRank*[6] by Pang *et al.* (2020), which employs the Set Transformer to obtain element representations that encode cross-document interactions and return a permutation invariant ranking by sorting the permutation equivariant relevance scores per each document.

Finally, a highly effective permutation learning NN method comes in the form of the Pointer Network (PN), originally introduced by Vinyals *et al.* (2015). PNs were initially developed to address certain limitations of earlier, *sequence-to-sequence* architectures [24], which have proven successful at NLP tasks such as translation [152] and parsing [153], image captioning [154] and short program evaluation [155]. Namely, PNs do not require the size of the output dictionary to be fixed *a priori*. The adjusted attention mechanism employed within PNs opened up the possibility of using RNN-based NN models on combinatorial optimization problems, such as the Travelling Salesman Problem, Delaunay triangulations [156] and the computation of planar convex hulls from finite sets, all of which the PN was tested on in the initial paper and performed very well.

In order to solve challenges where the dimensions of the output dictionary depend on the number of elements in the inputs sequence (and eventually the cardinality of the input set), the Pointer Network used a simplified modification of the attention

[6]This SetRank method shouldn't be confused with the Bayesian method for collaborative ranking from implicit feedback, known under the same name, developed by Wang *et al.* (2020).

mechanism introduced by Bahdanau *et al.* (2014). The PN follows an encoder-decoder framework [24], both originally in the form of RNNs. At each decoder step j a content-based attention vector $\mathbf{a}_j \in \mathbb{R}^n$ is produced. Each entry a_j^i in $\mathbf{a}_j$ represents the conditional probability of its corresponding input element x_i being the correct one to be placed next (in the j-th position) in the predicted sequence. This probability is conditioned on the entire input sequence $\mathbf{x} = (x_1, \ldots, x_n)$ in the form of all encoder hidden states $\mathbf{E} = (\mathbf{e}_1, \ldots, \mathbf{e}_n)$ as well as on the preceding decoder hidden states $\mathbf{D_j} = (\mathbf{d}_1, \ldots, \mathbf{d}_j)$, representing the output sequence predicted thus far, in an autoregressive manner.

Assuming a prediction target in the form of a permutation without repetition, this target becomes a sequence of non-negative integer indices $\mathbf{y} = (y_1, \ldots, y_n) \in \mathbb{Z}^n$, representing a valid permutation π, such that a target output $\mathbf{y}_\pi = (0, 2, 1)$ refers to the reordered sequence $\hat{\mathbf{x}}_\pi = (x_1, x_3, x_2)$ for a sample input $\mathbf{x} = (x_1, x_2, x_3)$. These discrete pointers are obtained from the continuous attention vector $(\mathbf{a}_j)$ at each j^{th} step via arg max. More formally, the pointer attention mechanism performs the following series of operations at each decoder step $j \in \{1, \ldots, n\}$:

$$z_j^i = \mathbf{v}^\top \tanh(\mathbf{W}_1 \mathbf{e}_i + \mathbf{W}_2 \mathbf{d}_j) \qquad \text{for } i \in (1, \ldots, n), \tag{2.17}$$

$$a_j^i = \frac{e^{z_j^i}}{\sum_{k=1}^{n} e^{z_j^k}} \qquad \text{for } i \in (1, \ldots, n), \tag{2.18}$$

$$p_\theta(y_j = i \mid y_1, \ldots, y_{j-1}, \mathbf{x}) = a_j^i \qquad \text{for } i \in (1, \ldots, n), \tag{2.19}$$

where $\mathbf{W}_1$, $\mathbf{W}_2$ and $\mathbf{v}$ are all trainable parameters (θ) of the pointer network, tanh is the hyperbolic tangent function, $\mathbf{d_j}$ is the decoder's hidden state at the j-th output element, $\mathbf{e_i}$ is the encoder hidden state corresponding to the i-th input element and finally the $\mathbf{z_j} = (z_j^1, \ldots, z_j^n)$ vector represents an output distribution over the dictionary of input elements. After the application of the softmax nonlinear activation

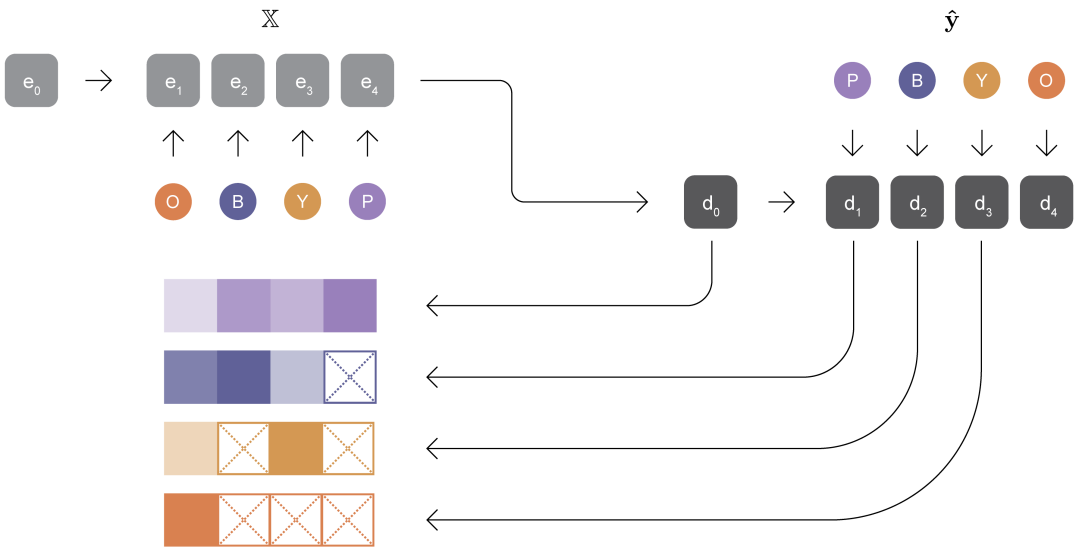


Figure 2.4: **Pointer Network**. A simplified illustration of pointer-style attention. Starting at the top-left corner, each element (x_i) of the input set ($\mathbb{X}$) is fed to the encoder, originally in the form of an RNN. This results in sequential encoder hidden states ($e_{1:4}$). The decoder network utilizes pointer attention to output a softmax distribution with dictionary size equal to the length of the input (bottom-left). The highest attention value points to the next element to be placed in the output permutation, which is used to obtain the decoder hidden state and fed to the model in the next recurrent step until all elements have been placed within the predicted sequence (top-right). Attention values are represented through colour opacity. Crossed out squares in the attention vectors represent a constraint in the form of disallowing repetitions.

function it becomes an attention vector $\mathbf{a}_j$. The element whose index is pointed to by the highest value in this attention vector becomes the element placed next in the output sequence and the loss is calculated from the sequence of predicted, continuous attention vectors. As shown in Equation 2.19, the probability of the i^{th} element being placed in the j^{th} position is conditioned on the entire available input set and the output sequence predicted up to the j^{th} step. For a visual explanation, please see Figure 2.4.

The main limitation of the original Pointer Network stems from the sequential nature of the encoder, making it sensitive to the arbitrary way in which the input set's elements are permuted into the input sequence. The authors of the PN address

this limitation in a follow-up paper [16], which introduces a method called *Read-Process-and-Write* (RPW). RPW obtains a permutation-invariant representation of the input and conditions its prediction entirely on this representation. For a visual introduction to pointer-based set-to-sequence architectures, see Figure 2.5.

RPW achieves this permutation invariance through a modified LSTM, in the form of a *process block*, which performs a preset number k of computation steps over the embedded set elements. Each step t_i consists of an attention transformation over the permutation invariant dot product of the the embedded elements and the preceding hidden state of the process block. At the final step t_k the hidden state of the process block is used to condition the probability of output pointers, in the way described in the previous paragraph outlining pointer networks. This forms the *write block* of the RPW model. RPW was an important step towards effective, permutation invariant set-to-sequence models, but the bottleneck incurred by a preset number of steps over the process block's hidden state resulted in slightly underwhelming performance and limited ability to generalize to unseen input set cardinalities, as tested on floating-point number sorting and constituency parsing in the original paper [16] and TSP and other combinatorial optimization problems in later works [1].

In consequence, a clear direction for successive set-to-sequence methods was to increase the representational capacity of the permutation invariant set representation. In the *ATTOrderNet* model developed by Cui *et al.* (2018) this took the form of a scaled dot-product self-attention of the Transformer [118] followed by an average pooling operation to ensure the invariance and a pointer-style attention for the permutation prediction. The first two of these modifications are reminiscent of the Set Transformer [11] and DeepSets [10] respectively, with the former using transformer-style self-attention to encode higher order interactions and the latter using a simple symmetric operator (such as sum or mean) to achieve the desired permutation invariance, as described in Section 2.3.

The ATTOrderNet was successfully applied to sentence and paragraph ordering

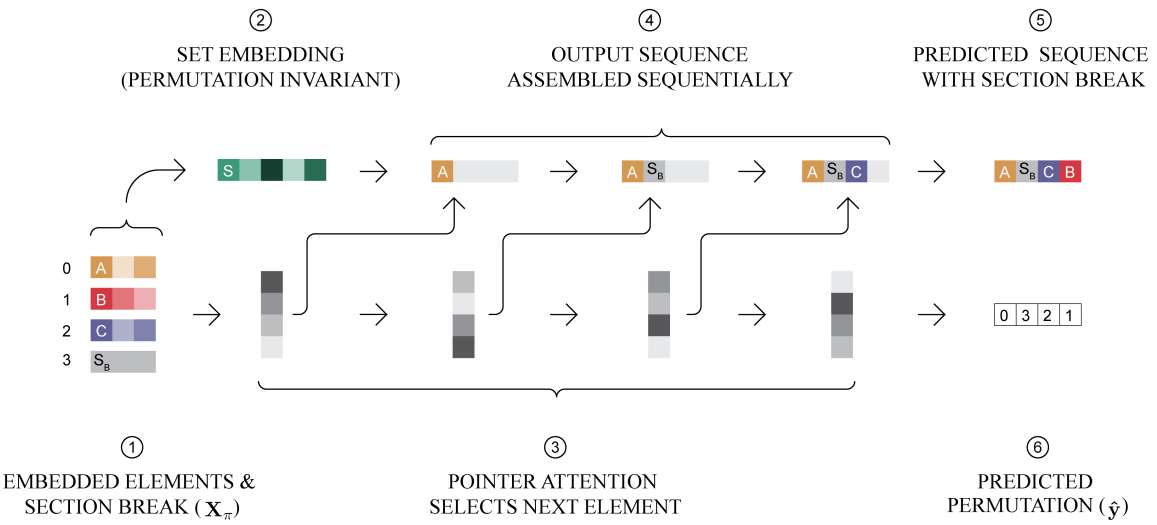


Figure 2.5: **Set-to-sequence Model Overview**. Starting on the left with an embedding of input set elements $\mathbf{X}_\pi \approx \mathbb{X}$ (1), including a predefined number $k-1$ of special section-break tokens ($\mathbf{S_B}$), the permutation invariant representation of the entire set is obtained through one of the set encoding methods (2). Then, a loop over the cardinality $n = |\mathbb{X}|$ of the input set begins (3), which utilizes pointer attention over all elements to successively select the next element to be placed in the output sequence (4). Depending on the model, the value of the attention vector can be conditioned purely on the set embedding (as is the case with RPW) or more commonly includes the per-element representations. The final obtained sequence of elements (5) represents the predicted permutation $\hat{\mathbf{y}}$ (6) of the original input matrix $\mathbf{X}_\pi$'s columns.

tasks within the domain of natural language processing [157]. Other variations of pointer networks have been utilized as parts of larger reinforcement learning models [88], within coherence modeling [158], text summarization [159] and graph-input ML [160]. A notable improvement over the canonical formulation of pointer-style attention as outlined in equations 2.17, 2.18 and 2.19, came in the form of an *Enhanced Pointer Network* (EPN), sometimes referred to as a *Future-History Pointer Network*, which introduced pairwise ordering predictions to improve both local and global coherence of the output sequence [17].

The EPN achieves this performance improvement by adding two pairwise ordering modules. The *FUTURE* module is trained on predicting the relative orientation of currently unordered set elements, while the *HISTORY* module reflects local orien-

tation between a chosen number (two in the original paper) of previously pointed-to elements and the candidates at current step, without the influence of the entire left-hand context in the form of the decoder's hidden state ($\mathbf{d}_i$). This left-hand context is considered to be "noisy" by the authors of the EPN because the order of elements predicted up to the i^{th} decoder step can contain incorrect placements. Formally, the conditional probability of a predicted order $\hat{\mathbf{y}}$ is calculated as:

$$p_\theta(\hat{\mathbf{y}} \mid \mathbb{X}) = \prod_{i=1}^{n} p_\theta(\hat{y_i} \mid \hat{\mathbf{y}}_{<i}, \mathbf{X}_\pi), \tag{2.20}$$

$$p_\theta(\hat{y_i} \mid \hat{\mathbf{y}}_{<i}, \mathbf{X}_\pi) = \text{softmax}(\mathbf{v}^\top \tanh(\mathbf{W}_1 \mathbf{d}_i + \mathbf{W}_2 \mathbf{M}_i)), \tag{2.21}$$

where $\mathbf{v}$, $\mathbf{W}_1$ and $\mathbf{W}_2$ are model parameters familiar from the classic pointer network equations (θ denotes the set of all trainable parameters), $\mathbf{X}_\pi$ is the matrix of arbitrarily ordered set elements, and $\mathbf{d}_i$ is the hidden state of the permutation module at current step i. The first hidden state $\mathbf{d}_0$ of this decoder is initialized from the permutation invariant set representation obtained via transformer-style multihead attention and average pooling. The $\mathbf{M}_i$ matrix is what differentiates EPN from PN, providing additional context consisting of two types of information. The first is global orientation relating all remaining unordered set elements to each other and the second is local coherence between two previously selected elements and remaining candidates. This context is obtained through the HISTORY and FUTURE sub-modules from $\mathbf{X}_\pi$ at each decoder step, essentially turning the permutation learning objective into a multi-task challenge. For further details regarding these two sub-modules, the reader is referred to Appendix A.2 of the fourth included article in Chapter 7.

During training of an EPN, given a batch $\mathbb{B}$ of m examples of the form $(\mathbb{X}, \mathbf{y})$, the loss function given in Equation 2.22 is minimized, where λ is a hyperparameter that balances the first term of the loss with $\mathcal{L}_{FH}$, a cross-entropy loss calculated from the pairwise predictions made by these FUTURE and HISTORY sub-modules:

$$\mathcal{L}(\mathbf{y};\theta) = -\frac{1}{m} \sum_{(\mathbb{X},\ \mathbf{y})\ \in\ \mathbb{B}} (\log\ p_\theta(\mathbf{y} \mid \mathbf{X}_\pi) + \lambda\ \mathcal{L}_{FH}). \tag{2.22}$$

For a visual explanation of the EPN, the reader is referred to Figure 2 from the original paper by Yin *et al.* (2020) or Figure 2 from the third included article in Chapter 6.

Despite all of these modifications resulting in greater representational power of the permutation invariant set encoding and increased performance in terms of the accuracy of the predicted permutation via pointer attention, the above-mentioned set-to-sequence models are incapable of predicting ordered clusters directly, without further alterations. In the incomplete approach to the Catalog Problem this limitation is circumvented by providing the model with a known number of special section-break tokens, that can be placed in the predicted sequence at various intervals, representing the end of a catalog section. In addition to requiring knowledge of the optimal number of clusters (and therefore sections) to predict ahead of time, this also forces the models to predict noisy, meaningless in-section order of input set elements[7]. In order to mitigate these limitations, two modifications of EPN-based set-to-sequence models are proposed, each with its own advantages and disadvantages, explained in more detail in the Appendix of the fourth included publication in Chapter 7. Nonetheless, the approach most closely addressing the requirements of the Catalog Problem necessitated the inclusion of neural clustering frameworks, capable of predicting an input-dependent, varying number of clusters from sets of any cardinality, as introduced in the next section.

[7]The in-section order is considered to be noise because the Incito service disregards it in order to be able to render the predicted catalog on any device, as described in Chapter 1.3.2.

2.5 Neural Clustering

Recent years have seen a growing interest in the development of NN-based clustering methods for complex data [161–165]. Commonly, the focus of such frameworks is placed on learning representations of the input set's elements that then lend themselves to some formulation of the clustering task via another NN, the embedding and cluster assignments being trained in an alternating fashion, with a shown tendency to overfit smaller datasets [166]. Another NN approach to clustering takes the form of first learning a similarity metric that predicts whether a pair of set elements belong to the same class and then training a NN to output assignments aligned with this similarity metric, as seen in the work of Hsu *et al.* (2018) in the form of *Constrained Clustering Networks* [167, 168] and *Attention Based Clustering* by Coward *et al.* (2020). Alternatively, *Centroid Networks* learn an element embedding that is then used for clustering purposes via a proposed *Sinkhorn K-means* algorithm [170]. The aforementioned methods employ NNs as parts of larger frameworks, but a separate line of research within the supervised setting (closer to this context) has developed methods for identifiying clusters directly within the forward pass of the network.

This line of research is often referred to as *amortized clustering* [20–22]. Within it, the term "amortization" referred to the investment of large computational resources to train a model that is then utilized for fast posterior inference [20, 23]. For the purposes of this thesis, this branch will be further referred to as *neural clustering*, denoting NN methods that are capable of jointly clustering and learning representations in a supervised manner. Under this definition, the foundational neural clustering method took the form of the *Neural Clustering Process* (NCP) proposed by Pakman *et al.* (2020). NCP is an elementwise method, iterating over every element of the input set, which can be of varying cardinality, and either assigning the current element to one of the previously opened clusters or a new one. In the latter case, the current element becomes the only member of a newly opened cluster, which can then be considered

as another candidate cluster for all remaining, unassigned set elements until there are none left. For a visual introduction to the ways in which the elementwise NCP and subsequent clusterwise CCP algorithms make cluster assignments, see Figure 2.6.

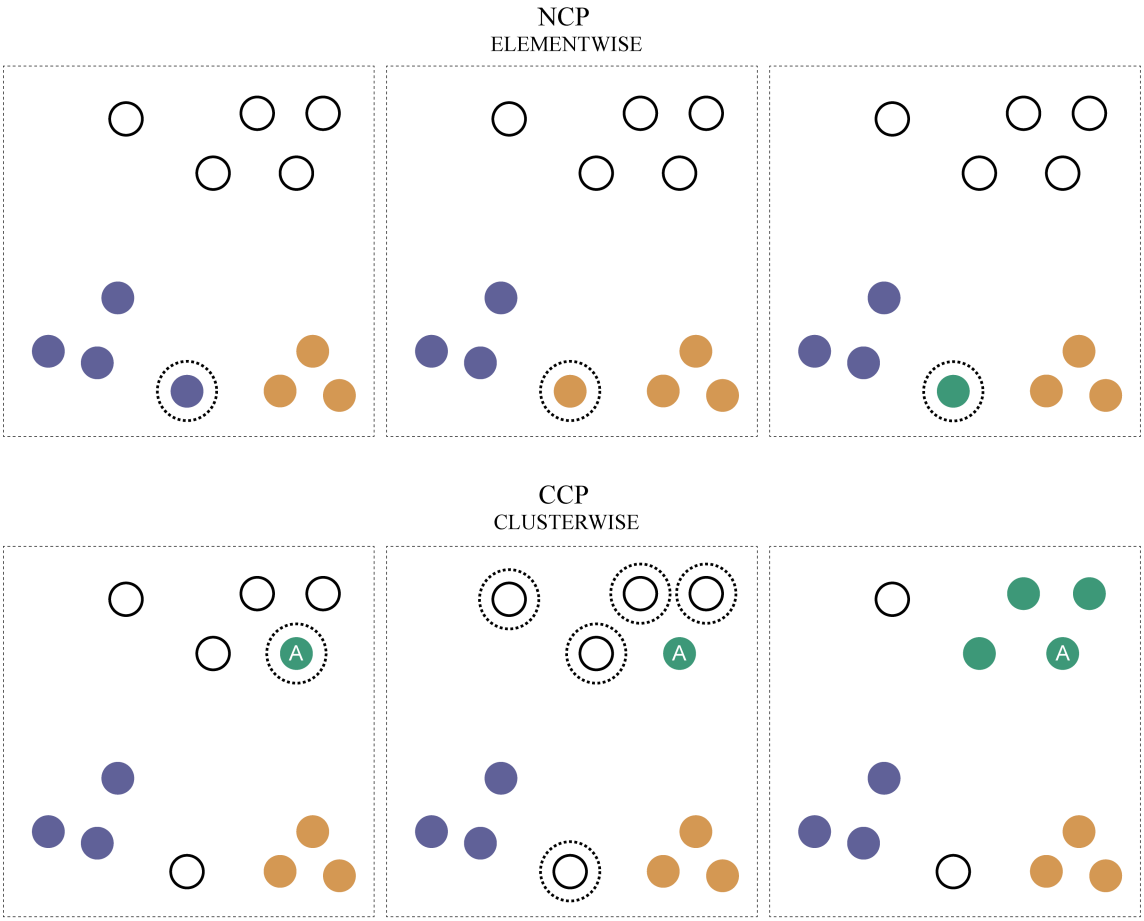


Figure 2.6: **NCP and CCP**. In the top row, an example of how the NCP model makes a single clustering decision during the network's forward pass is shown. NCP makes elementwise predictions. Unassigned elements are denoted as white circles with black borders. The three panels show three possible assignments for the current candidate point (in dotted circle). It can either belong to one of the two already opened clusters (blue and yellow) or form the beginning of a new, green one (rightmost panel). In the bottom row, an example of how the CCP model predicts a single, complete cluster. First, it randomly chooses an anchor element for the current cluster, marked by the dotted circle and letter A (leftmost panel). It then assesses the probability of the remaining unassigned elements (in dotted circles) belonging to the current cluster (middle). These probabilities can then be sampled from or compared to a threshold of 0.5 to complete the final, green cluster (right).

NCP consists of a number of constituent functions that take as input either individual elements of the set $x_i \in \mathbb{X}$ (where $|\mathbb{X}| = n$) or transformations of subsets thereof, and ultimately output n per-element cluster labels $\hat{c}_i \in \{1, \ldots, k\}$ for an a priori unknown number k of eventually predicted clusters. As an elementwise method, n total steps are performed through these functions, at each i^{th} iteration, with j currently opened candidate clusters:

1. $h(x_i)$ - resulting in a learned representation of each element x_i as already assigned to a cluster, together forming their joint representation $\mathbf{H}_\pi$. These per-element representations are summed to obtain a vector representation of each cluster $\hat{\mathbb{C}}_j \approx \mathbf{g}_j$, from the elements that were assigned to it (Equation 2.23).

2. $q(\mathbf{g_j})$ - resulting in a fixed-length vector representation of all j current clusters ($\{\hat{\mathbb{C}}_1, \ldots, \hat{\mathbb{C}}_j\} \approx \{\mathbf{g}_1, \ldots, \mathbf{g}_j\}$) jointly as $\mathbf{q}_i$ at the i^{th} elementwise iteration (Equation 2.24).

3. $u(x_i)$ - resulting in a vector representation of all currently unassigned elements jointly, as $\mathbf{u}_i$ (Equation 2.25).

4. $f(\mathbf{q}_i^p, \mathbf{u}_i)$ - resulting in the final, per-element probability of belonging to each of the j possible candidate clusters, from which cluster assignments $\hat{c}_{1:n}$ are derived. Within the NCP algorithm the currently considered i^{th} element's representation is added to each p^{th} candidate cluster's representation $\mathbf{g}_p$, resulting in j separate joint representations of all candidate clusters $\mathbf{q}_i^p$ (for each $p \in \{1, \ldots, j\}$), and the output of $f()$ is treated as the probability of the currently considered i^{th} element being assigned to each potential p^{th} cluster (Equation 2.26).

Each of these four functions is parameterized by an MLP neural network and designed to adhere to three important permutation symmetries. Firstly, the invariance with regards to the order of elements within each predicted cluster:

$$\mathbf{g}_j = \sum_{i:\hat{c}_i=j} h(x_i) \qquad h \ : \ \mathbb{R}^{d_x} \rightarrow \mathbb{R}^{d_h}. \tag{2.23}$$

Each of the j currently opened clusters' elements are encoded and added in this way, achieving permutation invariance through summation. Secondly, the clustering ought to be invariant to the order of predicted cluster labels, which in NCP is also achieved through adding the per-cluster representations $\mathbf{g}_j$ after they have been transformed by $q()$:

$$\mathbf{q}_i = \sum_{l=1}^{j} q(\mathbf{g}_l) \qquad q \ : \ \mathbb{R}^{d_h} \rightarrow \mathbb{R}^{d_q}. \tag{2.24}$$

Thirdly, the cluster labels should be invariant to the permutations of unassigned data points, which is captured in the following way:

$$\mathbf{u}_i = \sum_{m=i+1}^{n} u(x_m) \qquad u \ : \ \mathbb{R}^{d_x} \rightarrow \mathbb{R}^{d_u}. \tag{2.25}$$

Thus, $\mathbf{q}_i$ and $\mathbf{u}_i$ become fixed-length, permutation invariant vector representations of all the assigned and unassigned set elements for any value of n and i. Finally, the predicted per-element cluster assignments $\hat{c}_i$ are obtained from predicted per-candidate-cluster assignment probabilities $p_\theta(\hat{c}_i = j)$, where the probability of the i^{th} element's cluster assignment being equal to j is equivalent to the probability of the i^{th} element being assigned to the j^{th} predicted cluster: $p_\theta(x_i \in \hat{\mathbb{C}}_j) = p_\theta(\hat{c}_i = j)$. These probabilities are obtained from the output of the f() function:

$$p_\theta(\hat{c}_i = j \mid c_{1:i-1}, \mathbb{X}) = \frac{e^{f(\mathbf{q}_i^j, \mathbf{u}_i)}}{\sum_{p=1}^{j} e^{f(\mathbf{q}_i^p, \mathbf{u}_i)}}, \tag{2.26}$$

where $c_{1:i-1}$ refers to the true cluster assignments of preceding elements (when teacher-forced during training [171]), which are replaced by predicted cluster assignments

$\hat{c}_{1:i-1}$ outside of training. The authors of NCP refer to this as a *variable-input softmax* formulation, the real-valued function $f()$ returning a predicted cluster assignment probability for the current element x_i with regards to any number j of current candidate clusters (including the possibility of x_i becoming the first element of its own, new cluster). The computational cost of NCP as outlined above would be $O(nk)$ for a direct implementation, but through parallelization of the cluster aggregation and label prediction steps it is reduced to n network forward passes. NCP has been successfully applied to clustering mixtures of 2D Gaussians, MNIST classification and Neural Spike Sorting.

As discussed in Section 2.3, due to the size of the input sets in many areas of application, improving the time complexity to reach $O(k)$ is vital. The authors of NCP propose such a variant, called the *Clusterwise Clustering Process* (CCP). In the CCP approach, k iterations are performed over the given set of elements. At each iteration $j \in \{1, \ldots, k\}$, a uniformly sampled anchor element x_{a_j} is first selected from the set of available, unassigned elements. Then, a per-available-element probability of belonging to this anchored cluster is obtained, conditioned on the anchoring element, all unassigned points and previously predicted clusters, as well as an additional vector $\mathbf{z}_j$. This vector represents the features of the j^{th} cluster to which new elements (other than the current anchor x_{a_j}) are to be assigned at the j^{th} step of CCP. It is sampled from a Gaussian latent variable in the following way:

$$p_\theta(\mathbf{z}_j \mid x_{a_j}, \mathbf{u}_j, \mathbf{q}_j) = \mathcal{N}(\mathbf{z}_k \mid \mu(x_{a_j}, \mathbf{u}_j, \mathbf{q}_j), \sigma(x_{a_j}, \mathbf{u}_j, \mathbf{q}_j)), \tag{2.27}$$

where μ and σ are MLPs which take as input the j^{th} anchor element (x_{a_j}), the representations of all currently unassigned points ($\mathbf{u}_j$) and all previously predicted clusters ($\mathbf{q}_j$) and output the means and standard deviations for each dimension of the $\mathbf{z}_j$ vector. This is trained as a conditional variational autoencoder (VAE) as formulated by Sohn *et al.* (2015) [172], parameterized by fully connected neural networks with

sigmoid activation functions as the nonlinearity. The final, per-unassigned-element probability of belonging to the j^{th} cluster in CCP becomes:

$$p_\theta(\hat{c}_i = j \mid \hat{\mathbb{C}}_{1:j-1}, \mathbb{X}) = f(x_i, x_{a_j}, \mathbf{z}_k, \mathbf{u}_j, \mathbf{q}_j), \tag{2.28}$$

where $\hat{\mathbb{C}}_{1:j-1}$ are the clusters predicted during all iterative clustering steps before the current j^{th} step, x_i is each element currently unassigned to a cluster and CCP's $f()$ is again an MLP neural network with sigmoid activation. The output of $f()$ is a vector of assignment predictions for each of the unassigned elements, which are used to determine which of these elements should belong to the current j^{th} cluster, either through a 0.5 threshold value or by treating these predictions as probabilities and sampling from them.

A significant improvement over these two neural clustering methods was achieved through the application of the representationally powerful set encoding methods introduced in Section 2.3, particularly the Set Transformer [21, 22]. Whereas both NCP and CCP use simple summation to aggregate the sets of all unassigned elements, all already clustered elements and finally each predicted cluster individually into fixed-length vector representations, a natural progression would be to apply set encoders capable of retaining the relational information lost in the summation. In the aptly named *Attentive Clustering Process*[8] proposed by Wang *et al.* (2020) stacks of multihead attention blocks (MABs) followed by pooling by multihead attention (PMA) are used to this effect in lieu of the fully-connected NN layers parameterizing the aforementioned four functions in NCP and CCP, reminiscent instead of the Deep Sets method. More specifically, the induced self-attention blocks (ISABs, as outlined in Equation 2.10) were employed to avoid the $O(n^2)$ time complexity of stacked MABs. The inclusion of seed vectors reduces it to $O(nd_s)$, where d_s is the chosen dimensionality of the seed matrix. The constituent functions of ACP thus become:

[8]The *"Amortized Probabilistic Detection of Communities in Graphs"* paper [22] introducing the ACP method has undergone significant changes, some versions referring to ACP as CCP-Attn.

$$\mathbf{H}_\pi = h(\mathbb{X}) = \text{ISAB}_h(\mathbf{X}_\pi) \qquad h \ : \ \mathbb{R}^{n \times d_x} \rightarrow \mathbb{R}^{n \times d_h}, \tag{2.29}$$

$$\mathbf{q}_j = \sum_1^j q(\mathbf{H}_j) = \sum_1^j \text{PMA}_q(\text{ISAB}_q(\mathbf{H}_j)) \quad q \ : \ \mathbb{R}^{n \times d_h} \rightarrow \mathbb{R}^{d_q}, \tag{2.30}$$

$$\mathbf{u}_j = \text{PMA}_u(\text{ISAB}_u(\mathbf{X}_j^u)) \qquad u \ : \ \mathbb{R}^{n \times d_x} \rightarrow \mathbb{R}^{d_u}, \tag{2.31}$$

where $\mathbf{X}_j^u$ refers to the unassigned elements at the j^{th} clustering step and $\mathbf{H}_j$ refers to the representations of elements that belonged to the j^{th} cluster, obtained via the $h()$ function. These attention-based neural clustering methods showed superior performance on graph-structured inputs, such as the SNAP datasets [173], which contain networks of connections between consumer products. Therein, ACP was applied to the task of graph community detection, proving more effective than NCP and CCP at identifying sub-graphs composed of non-overlapping communities. This has been attributed to its ability to model higher-order interactions both within and between clusters [174]. However, there is one key quality relating to the Catalog Problem that the initial experiments suggest neither NCP, CCP nor ACP possesses (as shown in the fourth included article in Chapter 7). Namely, the ability to learn per-cluster cardinality constraints, which proved crucial for predicting varied, target-adherent catalog sections.

While there is a growing body of research investigating NN-based constrained clustering, its main focus until recently appears to be on pairwise, together/apart constraints, where two specific elements must or cannot be part of the same predicted cluster, as seen in the work of Fogel *et al.* (2019) and others [168, 176–179]. The work of Zhang *et al.* (2021) extended the semi-supervised, constrained clustering setting to triplet must-link/cannot-link relations, instance difficulty constraints and global size constraints. That last one is of particular interest to this setting, enabling their proposed *Deep Constrained Clustering* (DCC) model to achieve good performance on the

REUTERS-10K classification dataset [181] and circumvent known negative effects of introducing constraints into clustering methods, which can sometimes result in worse performance than the unconstrained alternatives [182]. Specifically, DCC can learn these and other types of constraints through the introduction of additional, differentiable loss functions that are then combined with the main clustering loss. The authors propose a global cardinality constraint loss, which assumes that each cluster should be approximately the same size. However, the Catalog Problem requires the prediction of clusters of varying cardinalities, dependent on the composition of each section itself as well as on all available elements in the input set. This requirement is further explored in the fourth included article in Chapter 7, where inspiration is taken from this loss-based approach to constrained NN-based clustering and a cluster-level cardinality prediction mechanism is implemented.

This concludes the chapter devoted to the theoretical introduction of relevant topics and notable related methods. It begun with a clear definition of the proposed Catalog Problem through its three core properties - varying-cardinality input, varying number of ordered, partitional clusters as output and the presence of supervision. Additionally, the incomplete and complete approaches to this task were outlined, the former requiring the number of target clusters to be known to the model ahead of time. In Section 2.2 a deeper look into the concept of a catalog was followed by a brief discussion of related approaches and alternative methods, most notably reinforcement learning and graph neural networks. The three subsequent sections were each devoted to one of the three main branches of NN methods that later became or inspired the constituent parts of proposed architectures capable of addressing the Catalog Problem in full. These included set encoding, permutation learning and supervised neural clustering.

In the next chapter, time is devoted to the proposed methods and tools, beginning with a customizable, synthetic catalog generation library for measuring the ability of models to learn compositional and structural rules dependent on n^{th} order interactions

among the elements of the input set. Afterwards, the focus is moved to the proposed Set Interdependence Transformer (SIT) - a set encoding module designed to more efficiently capture interactions within the entirety of the input set and finally to the introduction of Neural Ordering Clusters (NOC), a fully differentiable NN-based model architecture capable of predicting diverse partitional clusters from sets of any cardinality and ordering them according to the target preference in a supervised manner.

References

[1] M. Jurewicz and L. Derczynski, "PROCAT: Product catalogue dataset for implicit clustering, permutation learning and structure prediction," in *Thirty-fifth Conference on Neural Information Processing Systems. Datasets and Benchmarks Track*, 2021.

[20] A. Pakman, Y. Wang, C. Mitelut, J. Lee, and L. Paninski, "Neural clustering processes," in *International Conference on Machine Learning*, PMLR, 2020, pp. 7455–7465.

[21] J. Lee, Y. Lee, and Y. W. Teh, "Deep amortized clustering," *arXiv:1909.13433*, 2019.

[22] Y. Wang *et al.*, "Amortized probabilistic detection of communities in graphs," *arXiv:2010.15727*, 2020.

[23] S. Gershman and N. Goodman, "Amortized inference in probabilistic reasoning," in *Proceedings of the annual meeting of the cognitive science society*, vol. 36, 2014.

[24] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[2] X. Jin and J. Han, "Partitional clustering," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 766–766, ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_631. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_631.

[10] M. Zaheer, S. Kottur, S. Ravanbhakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 3394–3404.

[11] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh, "Set transformer: A framework for attention-based permutation-invariant neural networks," in *International conference on machine learning*, PMLR, 2019, pp. 3744–3753.

[13] B. Yang, S. Wang, A. Markham, and N. Trigoni, "Robust attentional aggregation of deep feature sets for multi-view 3d reconstruction," *International Journal of Computer Vision*, vol. 128, no. 1, pp. 53–73, 2020.

[14] K. Skianis, G. Nikolentzos, S. Limnios, and M. Vazirgiannis, "Rep the set: Neural networks for learning set representations," in *International conference on artificial intelligence and statistics*, PMLR, 2020, pp. 1410–1420.

[15] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *Advances in neural information processing systems*, vol. 28, 2015.

[16] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," *arXiv:1511.06391*, 2016.

[17] Y. Yin *et al.*, "Enhancing pointer network for sentence ordering with pairwise ordering predictions," in *AAAI*, vol. 34, 2020, pp. 9482–9489.

[25] H. Wenzel, D. Smit, and S. Sardesai, "A literature review on machine learning in supply chain management," in *Artificial Intelligence and Digital Transformation in Supply Chain Management: Innovative Approaches for Supply Chains. Proceedings of the Hamburg International Conference of Logistics (HICL), Vol. 27*, Berlin: epubli GmbH, 2019, pp. 413–441.

[26] J. Miles, J. Turner, R. Jacques, J. Williams, and S. Mason, "Using machine-learning risk prediction models to triage the acuity of undifferentiated patients entering the emergency care system: A systematic review," *Diagnostic and prognostic research*, vol. 4, no. 1, pp. 1–12, 2020.

[27] P. Rabiller, P. Boles, and B. Boashash, "Ordered clustering: A way to simplify analysis of multichannel signals," in *10th International Conference on Information Science, Signal Processing and their Applications (ISSPA 2010)*, IEEE, 2010, pp. 237–242.

[28] M. Le, D. Nauck, B. Gabrys, and T. Martin, "Sequential clustering for event sequences and its impact on next process step prediction," in *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Springer, 2014, pp. 168–178.

[29] M. Jurewicz and L. Derczynski, "Set interdependence transformer: Set-to-sequence neural networks for permutation learning and structure prediction," in *31st International Joint Conference on Articial Intelligence, IJCAI-ECAI 2022*, 2022.

[30] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure," in *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, 2007, pp. 410–420.

[31] Q. Huang, H. He, A. Singh, Y. Zhang, S.-N. Lim, and A. Benson, "Better set representations for relational reasoning," *Proc. NeurIPS*, vol. 34, 2020.

[32] R. R. Larson, J. McDonough, P. O'Leary, L. Kuntz, and R. Moon, "Cheshire ii: Designing a next-generation online catalog," *Journal of the American Society for information Science*, vol. 47, no. 7, pp. 555–567, 1996.

[33] S. Q. Yang and K. Wagner, "Evaluating and comparing discovery tools: How close are we towards next generation catalog?" *Library hi tech*, 2010.

[34] D. Allison, "Information portals: The next generation catalog," *Journal of web librarianship*, vol. 4, no. 4, pp. 375–389, 2010.

[35] J. Emanuel, "Usability of the vufind next-generation online catalog," *Information technology and libraries*, vol. 30, no. 1, pp. 44–52, 2011.

[36] A. M. Keller and M. R. Genesereth, "Multi-vendor catalogs: Smart catalogs and virtual catalogs," in *EDI FORUM-OAK PARK-*, THE EDI GROUP, LTD., vol. 9, 1996, pp. 87–93.

[37] J. B. Baty and R. M. Lee, "Intershop: Enhancing the vendor/customer dialectic in electronic shopping," *Journal of Management Information Systems*, vol. 11, no. 4, pp. 9–31, 1995.

[38] J. W. Palmer, "Retailing on the www: The use of electronic product catalogs," *Electronic Markets*, vol. 7, no. 3, pp. 6–9, 1997.

[39] D.-M. Lincke and B. Schmid, "Mediating electronic product catalogs," *Communications of the ACM*, vol. 41, no. 7, pp. 86–88, 1998.

[40] P. Schubert, "The participatory electronic product catalog: Supporting customer collaboration in e-commerce applications," *Electronic Markets*, vol. 10, no. 4, pp. 229–236, 2000.

[41] S. Handschuh, B. F. Schmid, and K. Stanoevska-Slabeva, "The concept of a mediating electronic product catalog," *Electronic Markets*, vol. 7, no. 3, pp. 32–35, 1997.

[42] E. Callahan and J. Koenemann, "A comparative usability evaluation of user interfaces for online product catalog," in *Proceedings of the 2nd ACM Conference on Electronic Commerce*, 2000, pp. 197–206.

[43] A. Kobsa, "An empirical comparison of three commercial information visualization systems," in *IEEE Symposium on Information Visualization, 2001. INFOVIS 2001.*, IEEE, 2001, pp. 123–130.

[44] P. Markellou, M. Rigou, and S. Sirmakessis, "Product catalog shopping cart effective design," in *Web systems design and online consumer behavior*, IGI Global, 2005, pp. 232–251.

[45] M. S. B. Mimoun, M. Garnier, R. Ladwein, and C. Benavent, "Determinants of e-consumer productivity in product retrieval on a commercial website: An experimental approach," *Information & management*, vol. 51, no. 4, pp. 375–390, 2014.

[46] H. K. Farsani and M. Nematbakhsh, "A semantic recommendation procedure for electronic product catalog," *International Journal of Applied Mathematics and Computer Sciences*, vol. 3, no. 2, pp. 86–91, 2006.

[47] D. Fisher, L. Xu, and N. Zard, "Ordering effects in clustering," in *Machine Learning Proceedings 1992*, Elsevier, 1992, pp. 163–168.

[48] J. Roure and L. Talavera, "Robust incremental clustering with bad instance orderings: A new strategy," in *Ibero-American Conference on Artificial Intelligence*, Springer, 1998, pp. 136–147.

[49] L. E. B. da Silva and D. C. Wunsch, "A study on exploiting vat to mitigate ordering effects in fuzzy art," in *2018 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2018, pp. 1–8.

[50] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[51] K. C. Chu, "Applications of artificial intelligence to chemistry. use of pattern recognition and cluster analysis to determine the pharmacological activity of some organic compounds," *Analytical chemistry*, vol. 46, no. 9, pp. 1181–1187, 1974.

[52] M. F. Janowitz, "An order theoretic model for cluster analysis," *SIAM Journal on Applied Mathematics*, vol. 34, no. 1, pp. 55–72, 1978.

[53] D. H. Fisher, "Knowledge acquisition via incremental conceptual clustering," *Machine learning*, vol. 2, no. 2, pp. 139–172, 1987.

[54] S. M. Van Dongen, "Graph clustering by flow simulation," Ph.D. dissertation, Utrecht University, 2000.

[55] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," *ACM Sigmod record*, vol. 28, no. 2, pp. 49–60, 1999.

[56] U. Shaham, K. Stanton, H. Li, B. Nadler, R. Basri, and Y. Kluger, "Spectralnet: Spectral clustering using deep neural networks," *arXiv:1801.01587*, 2018.

[57] K. Turowski, J. K. Sreedharan, and W. Szpankowski, "Temporal ordered clustering in dynamic networks," in *2020 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2020, pp. 1349–1354.

[58] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *science*, vol. 315, no. 5814, pp. 972–976, 2007.

[59] J. Vlasblom and S. J. Wodak, "Markov clustering versus affinity propagation for the partitioning of protein interaction graphs," *BMC bioinformatics*, vol. 10, no. 1, pp. 1–14, 2009.

[60] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[61] J. Fernandes, M. Simsek, B. Kantarci, and S. Khan, "Tabledet: An end-to-end deep learning approach for table detection and table image classification in data sheet images," *Neurocomputing*, vol. 468, pp. 317–334, 2022.

[62] S. Abadal, A. Jain, R. Guirado, J. López-Alonso, and E. Alarcón, "Computing graph neural networks: A survey from algorithms to accelerators," *ACM Computing Surveys (CSUR)*, vol. 54, no. 9, pp. 1–38, 2021.

[63] W. Jiang and J. Luo, "Graph neural network for traffic forecasting: A survey," *Expert Systems with Applications*, p. 117 921, 2022.

[64] R. Winter, F. Noé, and D.-A. Clevert, "Permutation-invariant variational autoencoder for graph-level representation learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 9559–9573, 2021.

[65] K. Schütt, P.-J. Kindermans, H. E. Sauceda Felix, S. Chmiela, A. Tkatchenko, and K.-R. Müller, "Schnet: A continuous-filter convolutional neural network for modeling quantum interactions," *Advances in neural information processing systems*, vol. 30, 2017.

[66] S. Cavallari, V. W. Zheng, H. Cai, K. C.-C. Chang, and E. Cambria, "Learning community embedding with community detection and node embedding on graphs," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 377–386.

[67] A. Tsitsulin, J. Palowitch, B. Perozzi, and E. Müller, "Graph clustering with graph neural networks," *arXiv:2006.16904*, 2020.

[68] D. Jin, B. Li, P. Jiao, D. He, and H. Shan, "Community detection via joint graph convolutional network embedding in attribute network," in *International Conference on Artificial Neural Networks*, Springer, 2019, pp. 594–606.

[69] F.-Y. Sun, M. Qu, J. Hoffmann, C.-W. Huang, and J. Tang, "Vgraph: A generative model for joint community detection and node representation learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[70] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," *arXiv:1803.03324*, 2018.

[71] A. Grover, A. Zweig, and S. Ermon, "Graphite: Iterative generative modeling of graphs," in *International conference on machine learning*, PMLR, 2019, pp. 2434–2444.

[72] R. Liao *et al.*, "Efficient graph generation with graph recurrent attention networks," *Advances in neural information processing systems*, vol. 32, 2019.

[73] S. A. Shah and V. Koltun, "Auto-decoding graphs," *arXiv:2006.02879*, 2020.

[74] W. Jin, R. Barzilay, and T. Jaakkola, "Hierarchical generation of molecular graphs using structural motifs," in *International conference on machine learning*, PMLR, 2020, pp. 4839–4848.

[75] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[76] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[77] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv:1611.09940*, 2016.

[78] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[79] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[80] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1179–1191, 2020.

[81] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[82] A. Karpathy, "Deep reinforcement learning: Pong from pixels," *http://karpathy.github. io/2016/05/31/rl*, 2016.

[83] J. Zhang, J. Kim, B. O'Donoghue, and S. Boyd, "Sample efficient reinforcement learning with reinforce," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 10 887–10 895.

[84] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans, "Bridging the gap between value and policy based reinforcement learning," *Advances in neural information processing systems*, vol. 30, 2017.

[85] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[86] A. X. Lee, A. Nagabandi, P. Abbeel, and S. Levine, "Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model," *Advances in Neural Information Processing Systems*, vol. 33, pp. 741–752, 2020.

[87] L. Zheng, T. Fiez, Z. Alumbaugh, B. Chasnov, and L. J. Ratliff, "Stackelberg actor-critic: Game-theoretic reinforcement learning algorithms," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, 2022, pp. 9217–9224.

[88] O. Vinyals, I. Babuschkin, W. M. Czarnecki, and Mathieu, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019, ISSN: 14764687. DOI: 10.1038/s41586-019-1724-z. [Online]. Available: http://dx.doi.org/10.1038/s41586-019-1724-z.

[89] J. Edmonds, "First mathematical theory of efficient combinatorial algorithms: Jack edmonds share," 1965.

[90] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer Science & Business Media, 2012.

[91] M. M. Flood, "The traveling-salesman problem," *Operations research*, vol. 4, no. 1, pp. 61–75, 1956.

[92] M. R. Garey and D. S. Johnson, "A guide to the theory of np-completeness, a series of books in the mathematical sciences," *Computers and Intractability, WH Freeman and Co, San Francisco, CA*, 1979.

[93] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" In *International Conference on Learning Representations*, 2018.

[94] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[95] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[96] X. Chen and Y. Tian, "Learning to perform local rewriting for combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[97] Q. Ma, S. Ge, D. He, D. Thaker, and I. Drori, "Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning," *arXiv e-prints*, arXiv–1911, 2019.

[98] K. Li, T. Zhang, and R. Wang, "Deep reinforcement learning for multiobjective optimization," *IEEE transactions on cybernetics*, vol. 51, no. 6, pp. 3103–3114, 2020.

[99] Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim, "Learning improvement heuristics for solving routing problems..," *IEEE transactions on neural networks and learning systems*, 2021.

[100] W. Kool, H. van Hoof, J. Gromicho, and M. Welling, "Deep policy dynamic programming for vehicle routing problems," in *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, Springer, 2022, pp. 190–213.

[101] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management science*, vol. 6, no. 1, pp. 80–91, 1959.

[102] M. Veres and M. Moussa, "Deep learning for intelligent transportation systems: A survey of emerging trends," *IEEE Transactions on Intelligent transportation systems*, vol. 21, no. 8, pp. 3152–3168, 2019.

[103] X. Zang, H. Yao, G. Zheng, N. Xu, K. Xu, and Z. Li, "Metalight: Value-based meta-reinforcement learning for traffic signal control," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 1153–1160.

[104] L. Brunke *et al.*, "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, pp. 411–444, 2022.

[105] L.-C. Yang and A. Lerch, "On the evaluation of generative models in music," *Neural Computing and Applications*, vol. 32, no. 9, pp. 4773–4784, 2020.

[106] P. R. Halmos, *Naive set theory*. Courier Dover Publications, 2017.

[107] Z.-Q. Zhao, S.-T. Xu, D. Liu, W.-D. Tian, and Z.-D. Jiang, "A review of image set classification," *Neurocomputing*, vol. 335, pp. 251–260, 2019.

[108] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[109] H. Oliveira, C. Silva, G. L. Machado, K. Nogueira, and J. A. dos Santos, "Fully convolutional open set segmentation," *Machine Learning*, pp. 1–52, 2021.

[110] S. Thrun and L. Pratt, *Learning to learn*. Springer Science & Business Media, 2012.

[111] J. Schmidhuber, "Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-... hook," Ph.D. dissertation, Technische Universität München, 1987.

[112] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*, 2001, pp. 285–295.

[113] J. Wang *et al.*, "Learning fine-grained image similarity with deep ranking," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1386–1393.

[114] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian, "Scalable person re-identification : A benchmark university of texas at san antonio," in *ICCV 2015*, 2015, pp. 1116–1124, ISBN: 978-1-4673-8391-2. DOI: 10.1109/ICCV.2015.133.

[115] G. Bebis and M. Georgiopoulos, "Feed-forward neural networks," *IEEE Potentials*, vol. 13, no. 4, pp. 27–31, 1994.

[116] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

[117] E. Wagstaff, F. Fuchs, M. Engelcke, I. Posner, and M. A. Osborne, "On the limitations of representing functions on sets," in *International Conference on Machine Learning*, PMLR, 2019, pp. 6487–6494.

[118] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[119] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv:1607.06450*, 2016.

[120] R. Girgis *et al.*, "Latent variable sequential set transformers for joint multi-agent motion prediction," in *International Conference on Learning Representations*, 2021.

[121] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, "Point transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 16 259–16 268.

[122] F. Fuchs, D. Worrall, V. Fischer, and M. Welling, "Se (3)-transformers: 3d roto-translation equivariant attention networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1970–1981, 2020.

[123] N. Janakarajan, J. Born, and M. Manica, "A fully differentiable set autoencoder," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 3061–3071.

[124] S. Zare and H. Van Nguyen, "Picaso: Permutation-invariant cascaded attentional set operator," *arXiv:2107.08305*, 2021.

[125] M. J. Hutchinson, C. Le Lan, S. Zaidi, E. Dupont, Y. W. Teh, and H. Kim, "Lietransformer: Equivariant self-attention for lie groups," in *International Conference on Machine Learning*, PMLR, 2021, pp. 4533–4543.

[126] N. Naderializadeh, S. Kolouri, J. F. Comer, R. W. Andrews, and H. Hoffmann, "Set representation learning with generalized sliced-wasserstein embeddings," *arXiv:2103.03892*, 2021.

[127] J. J. Hopfield and D. W. Tank, ""neural" computation of decisions in optimization problems," *Biological cybernetics*, vol. 52, no. 3, pp. 141–152, 1985.

[128] D. Krotov and J. J. Hopfield, "Dense associative memory for pattern recognition," *Advances in neural information processing systems*, vol. 29, 2016.

[129] M. Demircigil, J. Heusel, M. Löwe, S. Upgang, and F. Vermet, "On a model of associative memory with huge storage capacity," *Journal of Statistical Physics*, vol. 168, no. 2, pp. 288–299, 2017.

[130] D. Krotov and J. Hopfield, "Dense associative memory is robust to adversarial inputs," *Neural computation*, vol. 30, no. 12, pp. 3151–3167, 2018.

[131] M. Widrich *et al.*, "Modern hopfield networks and attention for immune repertoire classification," *Advances in Neural Information Processing Systems*, vol. 33, pp. 18 832–18 845, 2020.

[132] V. Srinivasan, "Linear programming computational procedures for ordinal regression," *Journal of the ACM (JACM)*, vol. 23, no. 3, pp. 475–487, 1976.

[133] P. McCullagh, "Regression models for ordinal data," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 42, no. 2, pp. 109–127, 1980.

[134] J. Cheng, Z. Wang, and G. Pollastri, "A neural network approach to ordinal regression," in *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, IEEE, 2008, pp. 1279–1284.

[135] N. Mostafazadeh *et al.*, "A corpus and cloze evaluation for deeper understanding of commonsense stories," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 839–849.

[136] Y. Xiao, B. Liu, and Z. Hao, "Multiple-instance ordinal regression," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 9, pp. 4398–4413, 2018.

[137] H. Zhu *et al.*, "Convolutional ordinal regression forest for image ordinal estimation," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[138] P. A. Gutiérrez, M. Perez-Ortiz, J. Sanchez-Monedero, F. Fernandez-Navarro, and C. Hervas-Martinez, "Ordinal regression methods: Survey and experimental study," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 1, pp. 127–146, 2015.

[139] J. L. Stuart and J. R. Weaver, "Matrices that commute with a permutation matrix," *Linear Algebra and Its Applications*, vol. 150, pp. 255–265, 1991.

[140] A. Grover, E. Wang, A. Zweig, and S. Ermon, "Stochastic optimization of sorting networks via continuous relaxations," *Preprint arXiv:1903.08850*, vol. ArXiv, 2019.

[141] G. Mena, D. Belanger, S. Linderman, and J. Snoek, "Learning latent permutations with gumbel-sinkhorn networks," in *International Conference on Learning Representations 2018*, Feb. 2018.

[142] S. Linderman, G. Mena, H. Cooper, L. Paninski, and J. Cunningham, "Reparameterizing the birkhoff polytope for variational permutation inference," in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2018, pp. 1618–1627.

[143] P. Emami and S. Ranka, "Learning permutations with sinkhorn policy gradient," *ArXiv*, vol. abs/1805.07010, 2018.

[144] N. Nishida and H. Nakayama, "Word ordering as unsupervised learning towards syntactically plausible word representations," in *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2017, pp. 70–79.

[145] Y. Zhang, J. Hare, and A. Prügel-Bennett, "Learning representations of sets through optimized permutations," *arXiv e-prints*, arXiv–1812, 2018.

[146] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: From pairwise approach to listwise approach," in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 129–136.

[147] B. Mitra *et al.*, "An introduction to neural information retrieval," *Foundations and Trends® in Information Retrieval*, vol. 13, no. 1, pp. 1–126, 2018.

[148] K. Pfannschmidt, P. Gupta, and E. Hüllermeier, "Deep architectures for learning context-dependent ranking functions," *ArXiv*, vol. abs/1803.05796, 2018.

[149] Q. Ai, K. Bi, J. Guo, and W. B. Croft, "Learning a deep listwise context model for ranking refinement," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2018, pp. 135–144.

[150] C. Wang, H. Zhu, C. Zhu, C. Qin, and H. Xiong, "Setrank: A setwise bayesian approach for collaborative ranking from implicit feedback," in *Proceedings of the aaai conference on artificial intelligence*, vol. 34, 2020, pp. 6127–6136.

[151] L. Pang, J. Xu, Q. Ai, Y. Lan, X. Cheng, and J. Wen, "Setrank: Learning a permutation-invariant ranking model for information retrieval," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020, pp. 499–508.

[152] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv:1409.0473*, 2014.

[153] O. Vinyals, Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, "Grammar as a foreign language," *Advances in neural information processing systems*, vol. 28, 2015.

[154] J. Donahue *et al.*, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 2625–2634.

[155] W. Zaremba and I. Sutskever, "Learning to execute," *arXiv:1410.4615*, 2014.

[156] D.-T. Lee and B. J. Schachter, "Two algorithms for constructing a delaunay triangulation," *International Journal of Computer & Information Sciences*, vol. 9, no. 3, pp. 219–242, 1980.

[157] B. Cui, Y. Li, M. Chen, and Z. Zhang, "Deep attentive sentence ordering network," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 4340–4349.

[158] L. Logeswaran, H. Lee, and D. Radev, "Sentence ordering and coherence modeling using recurrent neural networks," in *Thirty-second aaai conference on artificial intelligence*, 2018.

[159] Z. Sun, J. Tang, P. Du, Z.-H. Deng, and J.-Y. Nie, "Divgraphpointer: A graph pointer network for extracting diverse keyphrases," in *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2019, pp. 755–764.

[160] Y. Yin *et al.*, "An external knowledge enhanced graph-based neural network for sentence ordering," *Journal of Artificial Intelligence Research*, vol. 70, pp. 545–566, 2021.

[161] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *International conference on machine learning*, PMLR, 2016, pp. 478–487.

[162] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, "Towards k-means-friendly spaces: Simultaneous deep learning and clustering," in *international conference on machine learning*, PMLR, 2017, pp. 3861–3870.

[163] F. Li, H. Qiao, and B. Zhang, "Discriminatively boosted image clustering with fully convolutional auto-encoders," *Pattern Recognition*, vol. 83, pp. 161–173, 2018.

[164] S.-H. Lee and C.-S. Kim, "Deep repulsive clustering of ordered data based on order-identity decomposition," in *International Conference on Learning Representations*, 2020.

[165] Y. Li, P. Hu, Z. Liu, D. Peng, J. T. Zhou, and X. Peng, "Contrastive clustering," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 8547–8555.

[166] E. Aljalbout, V. Golkov, Y. Siddiqui, M. Strobel, and D. Cremers, "Clustering with deep learning: Taxonomy and new methods," *arXiv:1801.07648*, 2018.

[167] Y.-C. Hsu, Z. Lv, J. Schlosser, P. Odom, and Z. Kira, "Multi-class classification without multi-class labels," in *International Conference on Learning Representations*, 2018.

[168] Y.-C. Hsu, Z. Lv, and Z. Kira, "Learning to cluster in order to transfer across domains and tasks," *arXiv:1711.10125*, 2017.

[169] S. Coward, E. Visse-Martindale, and C. Ramesh, "Attention-based clustering: Learning a kernel from context," *arXiv:2010.01040*, 2020.

[170] G. Huang, H. Larochelle, and S. Lacoste-Julien, "Centroid networks for few-shot clustering and unsupervised few-shot classification," *arXiv:1902.08605*, vol. 3, no. 7, 2019.

[171] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[172] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," *Advances in neural information processing systems*, vol. 28, 2015.

[173] J. Leskovec and A. Krevl, *Snap datasets: Stanford large network dataset collection*, 2014.

[174] H. Liu, T. Zhou, J. Wang, L. Jing, and M. K. Ng, "Deep amortized relational model with group-wise hierarchical generative process," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2022.

[175] S. Fogel, H. Averbuch-Elor, D. Cohen-Or, and J. Goldberger, "Clustering-driven deep embedding with pairwise constraints," *IEEE computer graphics and applications*, vol. 39, no. 4, pp. 16–27, 2019.

[176] H. Liu, Y. Jia, J. Hou, and Q. Zhang, "Imbalance-aware pairwise constraint propagation," in *Proceedings of the 27th ACM international conference on multimedia*, 2019, pp. 1605–1613.

[177] B. Boecking and A. Dubrawski, "Pairwise feedback for data programming," *arXiv:1912.07685*, 2019.

[178] A. Shukla, G. S. Cheema, and S. Anand, "Semi-supervised clustering with neural networks," in *2020 IEEE Sixth International Conference on Multimedia Big Data (BigMM)*, IEEE, 2020, pp. 152–161.

[179] B. Boecking, V. Jeanselme, and A. Dubrawski, "Constrained clustering and multiple kernel learning without pairwise constraint relaxation," *arXiv:2203.12546*, 2022.

[180] H. Zhang, T. Zhan, S. Basu, and I. Davidson, "A framework for deep constrained clustering," *Data Mining and Knowledge Discovery*, vol. 35, no. 2, pp. 593–620, 2021.

[181] D. D. Lewis, Y. Yang, T. Russell-Rose, and F. Li, "Rcv1: A new benchmark collection for text categorization research," *Journal of machine learning research*, vol. 5, no. Apr, pp. 361–397, 2004.

[182] I. Davidson, K. L. Wagstaff, and S. Basu, "Measuring constraint-set utility for partitional clustering algorithms," in *European conference on principles of data mining and knowledge discovery*, Springer, 2006, pp. 115–126.

Chapter 3

Proposed Methods

This Chapter outlines a number of select contributions introduced in the included articles. It begins by providing more detail regarding the *synthetic catalogs* concept and corresponding library for procedurally generating[1] simplified, catalog-like structures. These synthetic catalogs follow customizable rules regarding the composition of their sections and the order in which these sections are presented in what SMEs refer to as a narrative structure. These rules depend entirely on the content of the input, which is made up of atomic product offer tokens. The n^{th} order interactions between these atomic tokens define what makes a valid section and a valid order of sections for a given input. The synthetic catalog datasets provide a way to empirically measure the models' ability to learn such interactions, which was suspected, based on discussions with SMEs, to be a vital ability when tackling real catalogs from the PROCAT dataset, as outlined in Chapter 1.3.3. Having established the synthetic catalogs, this chapter describes the proposed set encoder method, the Set Interdependence Transformer (SIT), which uses simple matrix augmentation to enhance the models' ability to condition the predicted catalogs on the properties of the entire input set. SIT is then used as one module of a composite set-to-sequence architecture and tested on the synthetic catalogs and a number of other datasets. However, this method still requires the target number of sections to be known a priori. Therefore, the final

[1]The term "procedural generation" refers here to algorithmic as opposed to manual creation of data [183].

proposed model, the Neural Ordered Clusters (NOC) described in detail in Section 3.3, combines set encoding, neural clustering and pointer attention into a complete approach towards tackling the Catalog Problem.

3.1 Synthetic Catalogs

In the early stages of this project, a number of meetings were held with SMEs responsible for various aspects of the decision-making process behind the actual, human-arranged electronic and paper product catalogs. These conversations revealed that both the composition of individual sections (or pages) and the order in which those sections are placed within the catalog heavily depend on higher-order interactions between the product offers present in the input set. For a practical example, the reader is referred to Chapter 1.3.4.

Knowing that learning compositional and structural rules dependent on n^{th} order interactions among input set elements would be crucial to solving the Catalog Problem on real-world datasets, it was decided to develop a library for generating simplified, synthetic catalogs. These synthetic catalogs were designed to be procedurally generated in adherence with rulesets expressed through an easily customizable configuration file. This library was also required to contain functions capable of taking a set of predicted synthetic catalogs and return detailed metrics regarding model performance, through which one could determine a specific model's demonstrated capability to construct synthetic catalogs that adhere to subsets of the chosen rules. This allows one to separately inspect a model's capacity to learn compositional or structural rules of the chosen n^{th} degree.

The intention was for those synthetically generated catalogs to resemble real catalogs from the PROCAT datasets in key aspects and thus help guide the research towards model architectures that would eventually be able to perform well on PROCAT itself, whilst allowing for quicker training and more fine-grained performance measures. The early, initial version of the synthetic catalog library was therefore in-

troduced in the same paper as the PROCAT dataset [1], included in full in Chapter 5. It was further improved into the format presented here over consecutive publications.

3.1.1 Tokens, Rulesets and Metrics

First, the most atomic constituent part of the proposed synthetic catalog is described. In real-world product catalogs the smallest part is a product offer, consisting of its textual description and corresponding image[2]. In the proposed synthetic simplification, individual product offers are represented by atomic tokens (color-coded for easy visual recognition). For example, these can include red, blue, yellow, green, purple and other tokens, as shown below in Figure 3.1. The input of the model in the synthetic catalog datasets consists of a multiset of n such tokens, allowing for the repetition of elements.

Figure 3.1: **Product Offer Tokens**. Individual product offers from the PROCAT dataset correspond to these colour-coded tokens within the synthetic catalogs. This convention is also upheld in some of the figures referring to actual, real-world catalogs for the sake of simplicity and consistency. The customizable configuration file allows for more token types and is in principle agnostic to the chosen color scheme.

The presence of certain combinations of these color-coded tokens in the input defines which ruleset should be applied to compose valid sections and order these sections into a complete synthetic catalog. Two main types of rules are proposed:

1. *Compositional rules*, which specify what makes a valid section
2. *Structural rules*, which determine what makes a valid order of sections, i.e. structure of the catalog

[2]Rather surprisingly, in practice the actual product image is added after the catalog structure has been decided based on product availability and sales projections.

Compositional rules can be thought of as relating to intra-cluster patterns, whereas structural rules encode inter-cluster interactions. Compositional rules can specify what types of tokens can be combined together to form a valid section, what is the maximum cardinality of a given type of section, what proportion of different types of tokens is allowed and more. Structural rules, in turn, can define what kind of sections should open a catalog, what types of sections must precede or follow one another and what section should be the last in a given catalog. Below, some specific examples of compositional rules are provided:

- "*If the input contains only red, blue and yellow tokens, a section containing red and yellow tokens in 1:1 ratio is a valid section*"
- "*If the input contains only red, blue, yellow and green tokens, a section containing red and green tokens must not exceed a cardinality of 4*"
- "*If the input contains red, blue, yellow and green tokens, a section containing red and yellow tokens is not a valid section*"

For comparison, these are examples of possible structural rules:

- "*If the input contains only red, blue and yellow tokens, a catalog must start with an all-red section*"
- "*If the input contains only red, blue and yellow tokens, a catalog must end on an all-blue section*"
- "*If the input contains only red, blue, yellow, green and purple tokens, a catalog must end on an all-purple section immediately following an all-blue section*"

A set of such rules that are applicable given a specific composition of the provided input tokens is referred to as a *ruleset*. For a visual explanation, see Figure 3.2. Each rule can depend on n^{th} order interactions. A ruleset that encompasses compositional

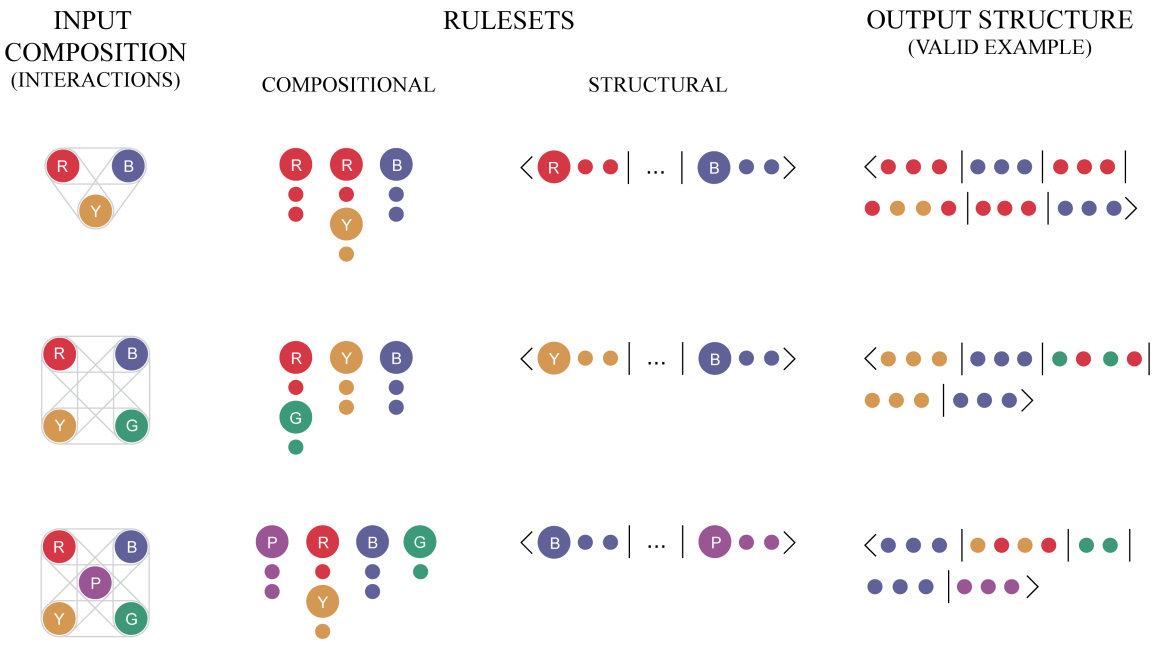


Figure 3.2: **Synthetic Catalog Rulesets**. Interactions between elements of the input (left) define the compositional and structural rules (middle), which inform the generation of these synthetic datasets. The actual input is a multiset of n product offer tokens, the leftmost panel shows only which types of tokens were present in it. Compositional rules define valid sections (shown in vertical rows), structural rules define valid section order, which in the figure is represented only by what section should be first and last (for simplicity). A successful model should learn these rules from supervised exposure to the resulting synthetic datasets, and then be able to order new sets of elements according to the learned rules. One valid example is given for each input composition (right, wrapped over 2 lines).

and structural rules for an input consisting only of any number of red, yellow and blue tokens would thus require the learning of a 3^{rd} order interaction. Another, different 3^{rd} order ruleset would define a valid catalog composed from an input consisting only of red, yellow and green tokens. 4^{th} order interaction rules would have to be learned to construct valid synthetic catalogs from a multiset of only red, yellow, blue and green tokens and so on. The rules that were applicable for a specific subset of tokens may become entirely incorrect with the addition, subtraction or substitution of a single element into the composition of the new input.

The library for procedurally generating synthetic catalogs was originally designed with a set-to-sequence approach in mind, which does not account for a varying number of sections. Thus, the actual input to the model would include a special "*section break*" token, repeated $k-1$ times[3], determining that the target catalog must consist of k sections (with k varying between examples). The set-to-sequence model constructs an ordered list from all input elements, representing the end of a section through the section break tokens. Any tokens placed between a pair of section break tokens are taken to form a single section. As such, set-to-sequence models also predict meaningless in-section order of elements, which would be ignored by the Incito service and is arbitrary within PROCAT, making the loss signal more noisy. Regardless, the library for generating these synthetic catalogs is agnostic to the type of model used to approach the underlying learning task, the section break tokens being added or removed during pre- and post-processing.

Having established an understanding of what elements the synthetic catalogs consist of, what kind of rules their composition and structure must follow and what determines which ruleset is applicable, the next paragraphs describe the kinds of insights this library might allow one to gain. The specific metrics explained in this section are repeatedly used in the experiments reported in the third and fourth included publications in Chapters 6 and 7. Overall, two main metrics are taken advantage of:

1. *Compositional Score* (CoS)

2. *Structural Score* (StS)

The CoS refers to the percentage of predicted sections that would be considered valid under the relevant ruleset. The StS refers to the percentage of predicted catalogs in which the order of sections followed the relevant structural ruleset. Both of these scores can be further broken down into separate metric per order of interaction

[3] $k-1$ section break tokens result in k sections because by adopted convention the last section in a catalog does not need to be followed by a section break token to be considered closed.

required to be learned (by default $n \in \{2, 3, 4, 5\}$), selected subsets thereof or even, on the most fine-grained level, per specific rule.

Generally, performance is also measured and reported through V-measure [30] and Kendall's Rank Correlation Coefficient (τ). V-measure is an external entropy-based cluster evaluation measure that outputs a single score in range over $[0, 1]$, 1 signifying a perfect match with ground truth, lending itself well to this supervised setting. Similarly Kendall's τ is a statistic used to measure ordinal association, returning a single score within $[-1, 1]$. It allows for the measurement of the correspondence between two permutations of the same set of elements, higher value indicating strong agreement. However, the Compositional and Structural Scores enable one to gain a deeper, more fine-grained insight into the actual performance of the tested models.

There are a couple of important properties and limitations that should be mentioned at this point. Good performance with regards to the Structural Score requires the prediction of at least some valid sections. For example, a structural rule requiring that the predicted catalog is opened with an all-red section cannot be adhered to if no all-red section has been composed in the set of predicted clusters. Whilst it is possible to output some invalid sections and still obtain a perfect structural score, these two metrics are not independent. In practice, it has been observed that during training models tend to first obtain a high score on the compositional scale, with the structural performance improving later on. It should also be noted that validating a model through these metrics during the very early stages of training can be noisy. It can take some time before fully valid sections emerge. This is particularly evident in certain set-to-sequence variants that don't preclude the prediction of empty sections (by placing three section break tokens one after another). Nonetheless, the per-rule insights have proven themselves useful when implementing the model architectures described in later sections.

3.1.2 Customizable Configuration

In order to make it easy to adjust the default rulesets from which datasets of synthetic catalogs can be procedurally generated, a simple configuration format is provided. It takes the form of a corresponding json file, snippets of which are shown below to illustrate key points. Firstly, the valid tokens for all included rulesets are specified:

```
"tokens": {
    "T_000": ["gray", 0],
    "T_001": ["red", 1],
    "T_002": ["yellow", 2],
    "T_003": ["blue", 3],
    "T_004": ["green", 4],
    "T_005": ["purple", 5]
  }
```

Each token is given an identifier (e.g. `T_001` for a red token), a short name referring to its colour and an ordered index. These are the basic atomic building blocks of the synthetic catalogs that one wants generated. Tokens can be removed or added. Secondly, the list of all possible valid sections is specified. For a simplified example:

```
"sections": {
    "SEC_001_4": ["T_001", "T_001", "T_001", "T_001"],
    "SEC_001_5": ["T_001", "T_001", "T_001", "T_001", "T_001"],
    "SEC_002_4": ["T_002", "T_002", "T_002", "T_002"],
    "SEC_002_5": ["T_002", "T_002", "T_002", "T_002", "T_002"],
    "SEC_003_4": ["T_003", "T_003", "T_003", "T_003"],
    "SEC_003_5": ["T_003", "T_003", "T_003", "T_003", "T_003"],
    "SEC_004_4": ["T_004", "T_004", "T_004", "T_004"],
    "SEC_004_5": ["T_004", "T_004", "T_004", "T_004", "T_004"],
    "SEC_005_4": ["T_005", "T_005", "T_005", "T_005"],
    "SEC_005_5": ["T_005", "T_005", "T_005", "T_005", "T_005"],
    "SEC_001_002_4": ["T_001", "T_001", "T_002", "T_002"],
    "SEC_001_004_4": ["T_001", "T_001", "T_004", "T_004"],
  }
```

Each section is given an identifier (e.g. `SEC_001_4`, where the last digit refers, by convention, to the cardinality) and a list of token identifiers that defines the section's composition. For example, section `SEC_001_004_4` consists of both red (`T_001`) and green (`T_004`) tokens at a 1:1 ratio, with a total cardinality of 4. The definitions of sections correspond to determining the compositional rules that are combined into larger rulesets connected to specific input compositions (ICs).

Finally, the token and section information is utilized to define both the compositional and structural rulesets per possible input composition. For example, in the simplest case of a 3rd order ruleset which should be applied when the input contains only red, yellow and blue tokens, one could define it as follows:

```
"IC_001": {
      "token_set": ["T_001", "T_002", "T_003"],
      "valid_sections": [
        "SEC_001_4", "SEC_001_5", "SEC_001_6",
        "SEC_002_4", "SEC_002_5", "SEC_002_6",
        "SEC_003_4", "SEC_003_5", "SEC_003_6",
        "SEC_001_002_4", "SEC_001_002_6"
      ],
      "valid_order": {
        "1": [
          "SEC_001_4", "SEC_001_5", "SEC_001_6"],
        "2": [
          "SEC_002_4", "SEC_002_5", "SEC_002_6",
          "SEC_001_002_4", "SEC_001_002_6"],
        "3": [
          "SEC_003_4", "SEC_003_5", "SEC_003_6"]}
```

This input composition is given an identifier (`IC_001`) and three properties:

1. A list of token identifiers that defines the input composition (`token_set`).
2. A `valid_sections` list, which defines which sections can be predicted from the corresponding input.
3. A `valid_order` dictionary, which defines the structure of the catalog.

The `valid_order` dictionary's keys are used to define the target order of sections. In the given example the catalog must start with an all-red section (represented by a list of section identifiers), followed by any of the other available sections, and end on an all-blue section. Other, more complex structural rules can also be defined through this format for higher order interactions in the input compositions made up of more types of atomic elements.

To summarize, a library for procedurally generating datasets of synthetic catalog-like structures from easily customizable configuration has been developed. This configuration consists of higher-order relational rules that define the composition and

structure of these synthetic catalogs. The intention is for this to reflect the kinds of patterns that are present in real-world product catalogs from the PROCAT dataset, based on discussions with SMEs responsible for the design of said catalogs.

The concept and implementation of synthetic catalog generation enables one to gain more fine-grained insights into model performance. It can quickly be seen whether a model performs better with regards to predicting valid sections (or clusters) via the compositional score or with regards to properly ordering sections into a reading narrative via the structural score. Additionally, one can check how a given model performs in relation to a specific order of interaction or even particular rule. The library is also agnostic to the model architecture and approach to the underlying Catalog Problem.

This library also address a limitation of the PROCAT dataset, namely the fact that in PROCAT one only ever has access to one target catalog for a particular input set of product offers. This is a challenge because the same input set could in principle be composed into multiple equally valid catalogs. Synthetic catalogs allow for the generation of multiple valid examples from the exact same underlying input, paving the way for a potential generative approach, as mentioned in the future work ideas in the fourth included article in Chapter 7.

3.2 Set Interdependence Transformer

Having obtained both the curated PROCAT dataset of actual, real-world catalogs designed by human experts and a simpler, synthetic dataset of catalog-like structures, it is time to move on to the particular proposed methods of improving performance and extending the capabilities of neural networks in relation to the Catalog Problem. Bearing in mind how the more and more powerful set encoding methods described in Chapter 2.3 were used to improve the performance of both permutation learning and neural clustering architectures, introduced in Chapters 2.4 and 2.5, a natural next step was to consider further possible enhancements in the area of learning set

representations.

Given the success of Transformer models in general [118, 184, 185], and of the Set Transformer (ST) in particular [11, 186, 187] within set-input ML, focus was initially drawn to the ST architecture. The ST improves over initial neural set encoding method, Deep Sets [10], which loses the information regarding element interactions in the sum-pooling, and over the simple Relational Networks proposed by Santoro *et al.* (2017), which pool over pairwise interactions among set elements. Specifically, ST uses the self-attention transformations of SAB, described in Equation 2.10, to explicitly encode pairwise interactions between set elements in each block (or layer), which can then be stacked to encode higher order interactions [11].

Seeing how human experts appear to consider the interactions among the entirety of the input set of product offers when designing the catalogs from the PROCAT dataset, this would potentially require one to stack n SAB layers to give a model the capacity to learn all relational rules that guide the creation of a catalog consisting of n product offers. This did not appear feasible, guiding the work towards proposing an adjustment to the ST method that would be specifically tailored to these aspects of the Catalog Problem. The core underlying idea was to take advantage of the attention-based set-pooling operations such as PMA, described in Equation 2.9, to obtain a representation of the set in its entirety and then treat it as another element of the initial input set in each layer, through simple matrix augmentation. This set encoder is further referred to as the Set Interdependence Transformer (or SIT for short). An introductory visual explanation[4] can be found in Figure 3.3.

3.2.1 SIT Encoder

The first necessary step is to take a set of elements, embed them and obtain a permutation invariant representation of this set in its entirety. In the included article in Chapter 6 a single layer of MAB is used (as outlined in Equation 2.4), which ob-

[4]The figure takes inspiration from the excellent graphics by Wagstaff *et al.* (2022), which depict various set encoders through the lens of Janossy Pooling [190].

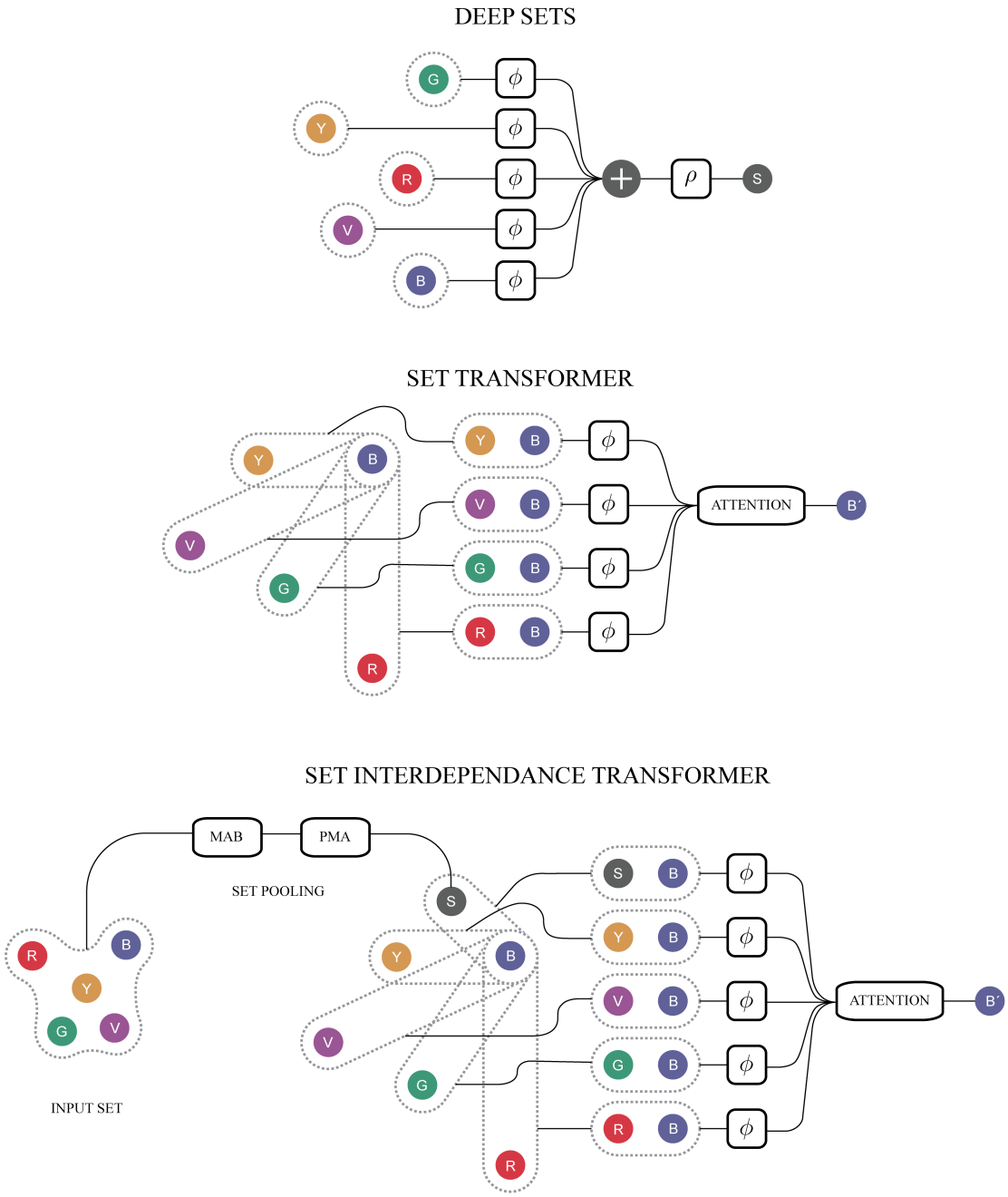


Figure 3.3: **Set Interdependence Transformer**. A comparison between different NN set encoding methods and SIT. At the top-left, Deep Sets encodes each element in an identical and independent way through a fully-connected layer (ϕ), sums these representations and further transforms them through another fully-connected layer ρ. At the top-right, a single Set Transformer layer (MAB) encodes pairwise relations between input set elements (mapping from sets to sets, shown for a single element `B` $\rightarrow$ `B'`). At the bottom, SIT first obtains a permutation-invariant representation of the entire set (gray circle with `S`) through MAB followed by PMA and then performs the attention transformation with this set representation treated as another set element.

tains the first embedded representations of all set elements ($\mathbf{E}_1^\pi$), followed by Pooling by Multihead Attention (or PMA, as described in Equation 2.9) to obtain the initial vector representation of the entire set ($\mathbf{s_1}$). However, in principle many other discussed methods of embedding set elements and transforming them into a permutation invariant representation of the set could be used. In a SIT Encoder, unless specified otherwise, MAB is first used to obtain the matrix of embedded elements ($\text{MAB}(\mathbf{X}_\pi) = \mathbf{E}_1^\pi$) followed by PMA to obtain the permutation invariant representation of the set ($\text{PMA}(\mathbf{E}_1^\pi) = \mathbf{s}_1$).

This initial set encoding step differs from the SIT layers to follow. Specifically, in the subsequent layers of SIT transformations a simple concatenation of the representation of the set in its entirety with the matrix of embedded elements is performed:

$$\mathbf{S}_1^\pi = (\mathbf{E}_1^\pi \mid \mathbf{s}_1), \tag{3.1}$$

where $\mid$ is used to denote matrix augmentation (as per convention), i.e. appending the column vector $\mathbf{s}_1$ to the matrix $\mathbf{E}_1^\pi$. This is followed by familiar transformer-style attention in the form of:

$$\text{SIT}_i(\mathbf{S}_{i-1}^\pi) = \text{softmax}(\frac{(\mathbf{S}_{i-1}^\pi \mathbf{W}_i^Q)\ (\mathbf{S}_{i-1}^\pi \mathbf{W}_i^K)^\top}{\sqrt{d_\mathbf{s}}})\ \mathbf{S}_{i-1}^\pi \mathbf{W}_i^V, \tag{3.2}$$

where $d_\mathbf{s}$ is a scaling factor equal to the length of the permutation invariant vector representing the entire set and the $\mathbf{W}^Q$, $\mathbf{W}^K$, $\mathbf{W}^V$ matrices are learned parameters. This operation can be repeated over any number of attention heads. Its output is a permutation equivariant matrix $\mathbf{S}_i^\pi$:

$$\text{SIT}_i(\mathbf{s_{i-1}}, \mathbf{E}_{i-1}^\pi) = \mathbf{S}_i^\pi = (\mathbf{E}_i^\pi \mid \mathbf{s}_i), \tag{3.3}$$

from which one can obtain the separate set and element representations required by (for example) a Pointer Network by reversing the augmentation and retrieving the

transformed $\mathbf{s}_i$ vector by its index, as has been done in the third included article in Chapter 6, or by adding a final layer of Pooling by Multihead Attention to obtain just the permutation invariant representation of the entire set, as has been done in the fourth included article in Chapter 7. Note that the required permutation invariance of $\mathbf{s}_i$ is maintained during the permutation equivariant SIT_i transformations, as per the formal proof, in Section 3.1 and supplementary material of Lee *et al.* (2019). In principle, these properties allow the Set Interdependence Transformer to relate individual element representations to the input set in its entirety, in each of the stacked SIT layers.

This simple variant of the Set Transformer became the set encoder of choice in later experiments. It was utilized as the set encoder module of a larger set-to-sequence architecture in the third included article in Chapter 6 and for obtaining both the representations of elements belonging to individual clusters and the representations of all clusters jointly within the Neural Ordered Clusters model described in Section 3.3 and introduced in the fourth included article in Chapter 7. As part of a larger architecture, SIT appears to offer better performance on the incomplete, set-to-sequence approach to the Catalog Problem on both PROCAT and synthetic catalogs, as well as a number of other datasets [29]. Although it marked progress within the presented research, the set-to-sequence model of which SIT was the set encoder did not constitute a complete approach to the Catalog Problem. Whilst a small ablation study to investigate SIT's empirical ability to learn higher order interactions in fewer layers than a ST (via the synthetic catalog library) is provided, further experiments reproducing these results on other datasets[5] as well as firm theoretical grounding for this speculated ability are a subject of potential future work.

[5] Especially popular datasets that particularly lend themselves to measuring relational reasoning, such as the humbly named Sort-of-CLEVR proposed by Santoro *et al.* (2017) or the adjusted Kinetics dataset for action recognition by Kay *et al.* (2017) as used in related work [31, 192].

3.3 Neural Ordered Clusters

In order to propose a method constituting a complete approach to the Catalog Problem, in a way that is most aligned with its characteristics, one needs to develop a model capable of predicting an input-dependent number of sections as clusters of product offers without inherent intra-cluster order. The incomplete, set-to-sequence approach, while it does properly predict inter-cluster order, also infers the order of elements within a predefined number of sections[6]. Both of these limitations can be addressed within the supervised setting through a combination of neural clustering, as described in Chapter 2.5, and pointer attention, as elaborated on in Chapter 2.4.

In the proposed Neural Ordered Clusters (NOC) a modified version of the Attentive Clustering Process (ACP, as outlined in Equations 2.29 – 2.31) is employed and combined with an Enhanced Pointer Network (EPN, described in Equations 2.20 – 2.22) and further augmented with a cluster cardinality prediction mechanism. The decision to add this third element stemmed from observed difficulties encountered by NPC, CCP and ACP-based architectures in learning to adhere to maximum cardinality constraints present in the synthetic catalogs datasets described in Section 3.1. Each of the three modular parts of the NOC architecture results in a separate loss factor, the model being trained on the combined, weighted loss with teacher forcing [171].

3.3.1 NOC Architecture

NOC is a *clusterwise* method capable of taking sets of varying cardinality, predicting an input-dependent number of diverse, partitional clusters and ordering these clusters. First, an introductory overview of the flow of information within NOC is given, followed by a closer look at each of its three main modules. For an initial visual ex-

[6]It is possible to modify originally set-to-sequence architectures to enable them to predict an input-dependent number of clusters. Two such variants are proposed in the fourth included article, in Chapter 7, where they are referred to as S2S-B and S2S-C.

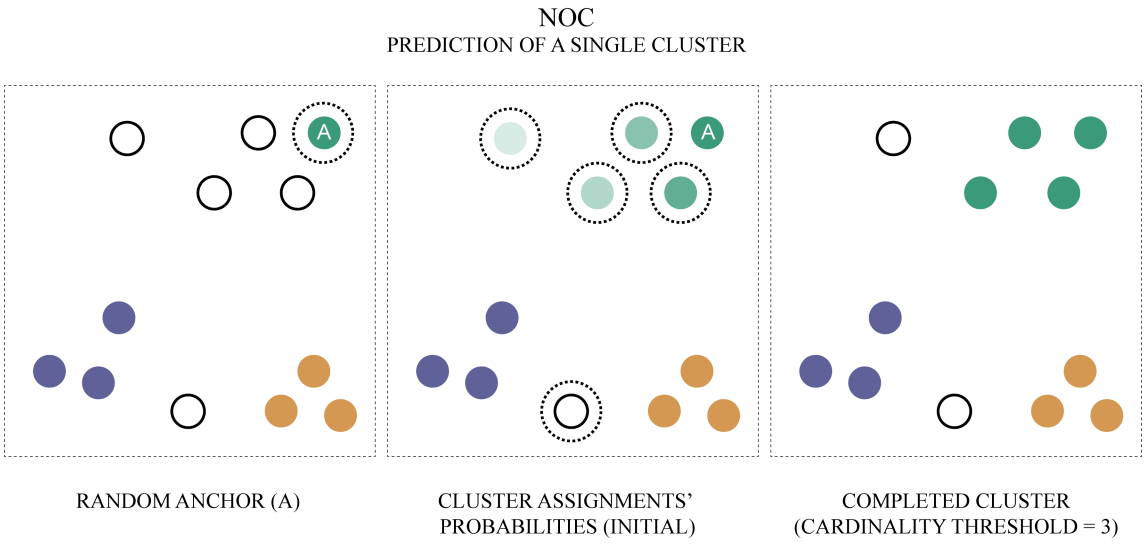


Figure 3.4: **NOC Clustering Step**. A visualization of how NOC predicts a single cluster. In the leftmost panel a random anchor element (circled with dotted line and marked with the letter A) is chosen from the set of unassigned elements, which are marked as white circles. In the middle panel, initial predictions regarding the probability of each remaining unassigned element belonging to the current, green cluster are obtained, higher confidence being marked with higher colour intensity. In the rightmost panel, the remaining candidates are further adjusted based on a predicted cardinality threshold ($t_j = 3$, anchor element not being counted), three elements with the highest predicted probability are assigned to the completed cluster and the other two are returned to the unassigned set. Contrast with Figure 2.6.

planation, the reader is referred to Figure 3.4. Much in line with the process familiar from preceding neural clustering methods, the NOC algorithm begins by selecting a random anchor element from the provided input set of unassigned elements. Based on the representation of this anchor, the remaining available elements and (in following iterations) the already predicted clusters, it obtains a vector $\mathbf{z}_j \in \mathbb{R}^{d_z}$ sampled from a Gaussian latent variable, which encapsulates the properties of the current j^{th} cluster (containing the anchor), which remains to be completed in the course of the j^{th} forward pass, as described in Equation 2.27.

It then calculates the probability of each unassigned element belonging to the j^{th} cluster, as shown in Equation 2.28. The per-element probabilities are used to obtain

initial cluster assignments through a threshold value of 0.5. These assignments are further adjusted through the cardinality prediction module, which outputs a maximum cardinality threshold (t_j). This possibly curtails the initial element selection, again conditioned on the anchor, available elements and previously predicted clusters. Per-element probabilities are sorted and up to t_j elements with the highest corresponding probability become the final members of the j^{th} cluster.

This process is repeated until there are no remaining, unassigned elements, one cluster being formed at each forward pass through these first two stages of NOC. The obtained k clusters ($\hat{\mathbb{C}} = \{\hat{\mathbb{C}}_1, \ldots, \hat{\mathbb{C}}_k\}$) are transformed through stacked SIT layers and PMA into fixed-length, per-cluster vector representations ($\mathbf{G}_k = (\mathbf{g}_1, \ldots, \mathbf{g}_k)$) as well as a representation of the entire set of predicted clusters jointly ($\mathbf{q}_k$, obtained from $\mathbf{G}_k$). The latter forms the first hidden state ($\mathbf{h}_0^d$) of the EPN, which predicts an attention distribution over all predicted clusters iteratively, in each of $k-1$ iterations (last remaining cluster becomes the last one in the sequence). At each iterative step, the cluster with the highest corresponding attention value is selected as next in the output sequence of clusters. In this way the prediction of the order of elements within clusters is avoided and the model is able to obtain an input-dependent number of clusters with varying cardinalities. A more detailed overview of the NOC architecture is provided in Figure 3.5, with every step of the model's forward pass being described in the paragraphs that follow.

NOC's input takes the form of a set of per-element vector representations forming an arbitrarily ordered matrix $\mathbf{E}_\pi \approx \mathbb{X}$. By convention, the subset of these elements that are yet to be assigned to a cluster at the j^{th} clustering iteration of NOC is denoted as the matrix $\mathbf{U}_j$. The **first stage** of the NOC algorithm, further referred to as NOC_1, constitutes the initial selection of elements to be assigned to the current cluster ($\mathbb{C}_j$) at clustering step j. For every cluster to be predicted, a random anchor element ($\mathbf{e}_j^a$) is sampled uniformly, a fixed-length vector representation of all currently unassigned elements ($\mathbf{u}_j$) is obtained from the embedded representation of all unassigned elements

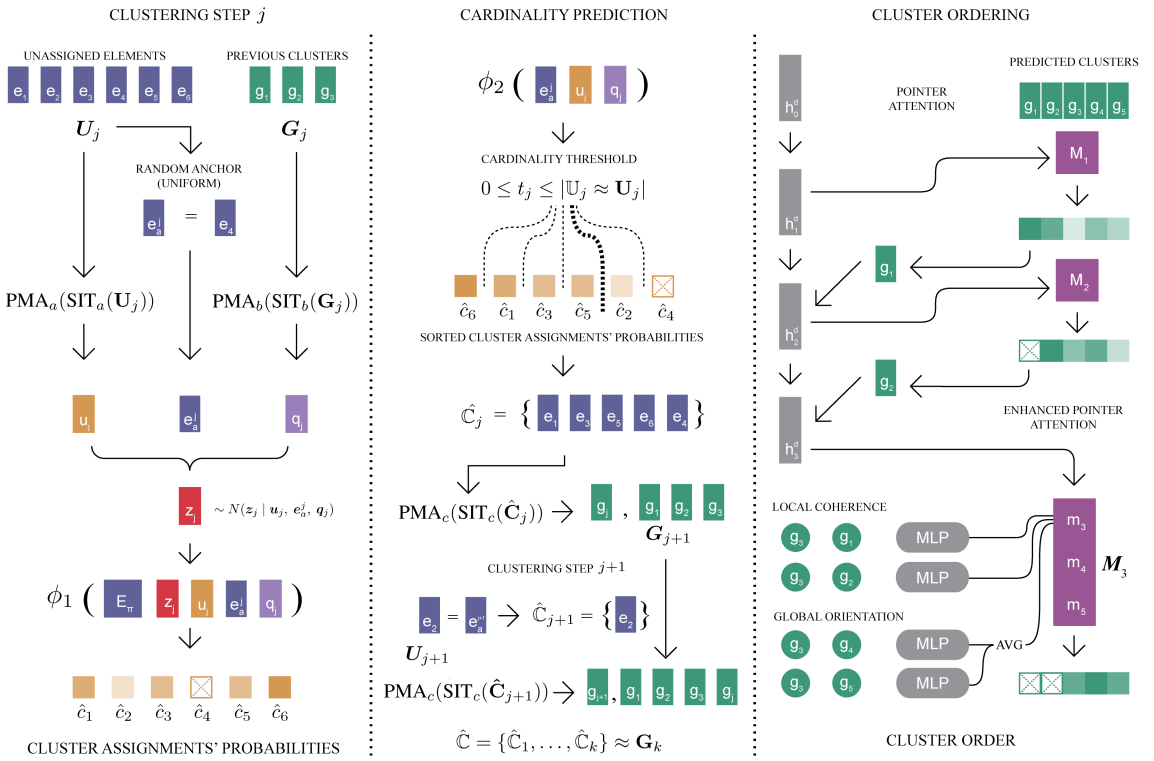


Figure 3.5: **NOC Architecture**. An overview of how NOC completes the clustering of the input set (in two steps, j and $j+1$), followed by ordering of the predicted clusters (rightmost panel). Starting at the top of the leftmost panel and moving to the bottom before switching to the next panel to the right, at clustering step j the representations of unassigned elements ($\mathbf{U}_j = \mathbf{e}_{1:6}$), previously created clusters ($\mathbf{G}_j = \mathbf{g}_{1:3}$) and a randomly selected anchor element ($\mathbf{e}_a^j$) are used to obtain initial cluster assignments' probabilities, which represent how likely each unassigned element is to become part of the current, j^{th} cluster ($\hat{c}_{1:6} \approx p_\theta(\hat{c}_{1:6} = j) = p_\theta(e_{1:6} \in \hat{\mathbb{C}}_j)$), with color opacity indicating higher predicted probability. In the middle panel the current cardinality ($t_j = 4$) is predicted and used to adjust the j^{th} cluster ($\hat{\mathbb{C}}_j$), which is then transformed via PMA$_c$(SIT$_c$($\hat{\mathbb{C}}_j$)) into its embedded representation $\mathbf{g}_j$, which becomes part of the $\mathbf{G}_{j+1}$ matrix and is used during the remaining clustering steps. In the rightmost panel, after k iterations of the NOC$_1$ and NOC$_2$ steps, the predicted clusters ($\hat{\mathbb{C}} \approx \mathbf{G}_{j+1} = \mathbf{G}_k$) are ordered via NOC$_3$'s Enhanced Pointer attention (as described in Equations 2.20 - 2.22).

individually ($\mathbf{U}_j$) through a dedicated stack of SIT layers:

$$\mathbf{u}_j = \mathrm{PMA}_a(\mathrm{SIT}_a(\mathbf{U_j})), \tag{3.4}$$

and then the fixed-length vector representation of all preceding clusters jointly ($\mathbf{q}_j$) is obtained from the representation of previously predicted clusters individually ($\mathbf{G}_j$ referring to $\{\hat{\mathbb{C}}_1, \ldots, \hat{\mathbb{C}}_{j-1}\} \approx (\mathbf{g}_1, \ldots, \mathbf{g}_{j-1})$) in the corresponding way:

$$\mathbf{q}_j = \mathrm{PMA}_b(\mathrm{SIT}_b(\mathbf{G_j})). \tag{3.5}$$

Then, the $\mathbf{z}_j$ vector is sampled from a Gaussian latent variable of predicted means and standard deviations as a representation of the j^{th} cluster, conditioned on the anchor element, the remaining unassigned elements and the previously assigned elements. This function is trained as a conditional variational autoencoder [172]:

$$p_\theta(\mathbf{z_j} \mid \mathbb{X}_j) = \mathcal{N}(\mathbf{z}_j \mid \mu(\mathbf{e}_j^a, \mathbf{u}_j, \mathbf{q}_j), \sigma(\mathbf{e}_j^a, \mathbf{u}_j, \mathbf{q}_j)) \tag{3.6}$$

where $\mathbb{X}_j$ is used as a shorthand for the state of the input set $\mathbb{X}$ at step j, referring to which elements have been assigned to which clusters, μ and σ are MLPs which take as input the j^{th} anchor element ($\mathbf{e}_j^a$), the representations of all currently unassigned points ($\mathbf{u}_j$) and all previously predicted clusters ($\mathbf{q}_j$) and output the means and standard deviations for each dimension of the $\mathbf{z}_j$ vector. Finally the model predicts the per-unassigned-element ($\mathbf{e}_i$) cluster assignment probabilities for the j^{th} cluster ($p_\theta(\hat{c}_i = j)$). This is done through a fully connected network ϕ_1, with sigmoid activation, which takes as input the concatenated vector representations of each unassigned element, the anchor, the sampled vector representing the properties of the j^{th} cluster ($\mathbf{z}_j$), as well as the unassigned elements ($\mathbf{u}_j$) and preceding clusters ($\mathbf{q}_j$) jointly:

$$\mathrm{NOC}_1(\mathbf{e}_i, \mathbb{X}_j) = p_\theta(\hat{c}_i = j \mid \mathbb{X}_j) = \mathrm{sigmoid}(\mathrm{concat}(\phi_1(\mathbf{e}_i, \mathbf{e}_a^j, \mathbf{z}_j, \mathbf{u}_j, \mathbf{q}_j))). \tag{3.7}$$

The **second stage**, further referred to as NOC_2 makes an adjustment to the initial cluster assignments obtained through NOC_1. It is depicted in the upper half of the middle panel in Figure 3.5. Each clustering step j (leftmost and middle panel) consists of a single NOC_1 pass followed by a single NOC_2 pass, at the end of which a complete j^{th} cluster has been predicted and the set of remaining, unassigned elements can be updated. NOC_2 is comprised of a fully-connected neural network ϕ_2, which predicts a max cardinality threshold (t_j) for the j^{th} cluster, based on a single concatenated vector of the representation of the anchor element ($\mathbf{e}_a^j$), the unassigned elements jointly ($\mathbf{u}_j$) and the previously predicted clusters ($\mathbf{q}_j$):

$$t_j = \text{NOC}_2(\hat{\mathbb{C}}_j, \mathbb{X}_j) = \phi_2(\text{concat}(\mathbf{e}_a^j, \mathbf{u}_j, \mathbf{q}_j)), \tag{3.8}$$

$$\hat{\mathbb{C}}_j = \begin{cases} \hat{\mathbf{C}}^j, & \text{if } |\hat{\mathbb{C}}_j| \leq t_j \\ \hat{\mathbf{C}}^j_{1:t_j}, & \text{otherwise} \end{cases}, \tag{3.9}$$

where the model either adjusts the composition of $\hat{\mathbb{C}}_j$ if its cardinality exceeds the predicted threshold, or keeps the cluster as is. The adjustment is performed by maintaining up to t_j elements selected by NOC_1 with the highest cluster assignment probabilities as members of the j^{th} cluster and returning the remainder to the set of unassigned elements. The j^{th} cluster's fixed-length vector representation ($\mathbf{g}_j$) is obtained through a dedicated stack of SIT layers from the representations of the elements assigned to this cluster ($\hat{\mathbf{C}}_j \approx \hat{\mathbb{C}}_j = \{e_m, \ldots\}$ such that $\hat{c}_m = j$):

$$\mathbf{g}_j = \text{PMA}_c(\text{SIT}_c(\hat{\mathbf{C}}_j)). \tag{3.10}$$

Finally, after k iterations of repeated NOC_1 and NOC_2 steps, k partitional clusters ($\hat{\mathbb{C}}_1, \ldots, \hat{\mathbb{C}}_k$) have been obtained from the input set. Each of these clusters has a cardinality between 1 and n. Their fixed-length vector representations ($\mathbf{G}_k = (\mathbf{g}_1, \ldots, \mathbf{g}_k)$)

obtained during NOC_1 and NOC_2 form the input expected by set-to-sequence methods, which can then be used to output these clusters' predicted order.

Thus, in the **third and final stage** of NOC_3 an Enhanced Pointer Network, or EPN [17], is employed to perform $k-1$ iterations over the predicted clusters (third panel in Figure 3.5). The initial hidden state ($\mathbf{h}_0^d$) of the EPN is initialized as the permutation invariant representation of all predicted clusters ($\mathbf{G}_k \approx \hat{\mathbb{C}}$), obtained via SIT and PMA:

$$\mathbf{h}_0^d = \text{PMA}_d(\text{SIT}_d(\mathbf{G}_k)) \tag{3.11}$$

At each iteration step $m \in (1, \ldots, k-1)$ the EPN predicts an attention vector $\mathbf{a}_m$ over all remaining, unordered clusters, the highest attention value pointing to the cluster to be placed at the m^{th} position in the sequence of ordered clusters. Once only one cluster remains to be ordered, it is placed at the end of that sequence (hence $k-1$):

$$\mathbf{a}_m = \text{softmax}(\mathbf{v}^\top \tanh(\mathbf{W}_1\mathbf{M}_m + \mathbf{W}_2\mathbf{h}_m^d)) \tag{3.12}$$

$$\text{where } \mathbf{h}_m^d = \text{LSTM}(\mathbf{h}_{m-1}^d, \mathbf{g}_{m-1}), \tag{3.13}$$

where $\mathbf{g}_{m-1}$ refers to the representation of the cluster placed in the previous, $m-1^{\text{th}}$ position in the iteratively predicted output sequence of ordered clusters. This largely resembles a Pointer Network (see Equations 2.17 - 2.19), with the exception of matrix $\mathbf{M}_m$ specific to the EPN, as outlined in Equations 2.20 - 2.22. The $\mathbf{M}_m$ matrix provides additional context consisting of two kinds of information. The first is global orientation relating all remaining unordered clusters to one another. The second is local coherence between previously selected clusters and remaining candidates. This contextual information is obtained via *HISTORY* and *FUTURE* sub-modules from

the original matrix of all cluster representations ($\mathbf{G}_k \approx \hat{\mathbb{C}}$). These two sub-modules output pairwise ordering predictions in relation to each candidate cluster, which are then concatenated to form $\mathbf{M}_m$. For further implementation details regarding these two sub-modules the reader is referred to Appendix A.2 of the fourth included article or to the original paper by Yin *et al.* (2020). Together, these three elements of NOC allow for the prediction of an input-dependent number k of partitional clusters with varying cardinalities. The full NOC algorithm, additional notes on the objective function and training process are provided in the appendix of the fourth included article, in Chapter 7, for further clarity.

3.3.2 Performance and Limitations

The NOC approach has been tested on a number of datasets that epitomize the Catalog Problem. These include a toy task of ordering 2D Mixtures of Gaussians based on their centroid's distance from the origin point, the procedurally generated synthetic catalogs, introduced in Chapter 3.1, and the main PROCAT dataset, described in Section 1.3.3. The detailed results are presented in the Experiments section of the fourth included article, in Chapter 7. Therein, NOC is compared with two separate groups of baselines:

1. Neural clustering methods with an added set-to-sequence module

2. Proposed variants of the set-to-sequence architecture

The set-to-sequence (S2S) module added to existing neural clustering methods takes the predicted clusters and outputs their order via attention-based pointing (an EPN, for a direct comparison with NOC). These methods include the pointwise Neural Clustering Process (NCP), the Clusterwise Clustering Process (CCP) and the Attentive Clustering Process (ACP) developed by Pakman *et al.* (2020) and Wang *et al.* (2020). The proposed variants include modifications to the pointer mechanism (S2S-C) and a slight reformulation of the task (S2S-B). The former enables the prediction

of ordered *clusters* (as opposed to ordered *elements*), the latter only the prediction of a number of clusters that isn't known *a priori* (whilst still predicting in-cluster order). The specifics of both S2S-B and S2S-C are presented in the Appendix of the included article introducing NOC.

The first group of baselines appears ineffective at learning cluster-level cardinality constraints. NOC offers an improvement in this area through its cardinality prediction module (NOC_2). NOC also outperforms pure S2S methods in terms of both the quality of the clustering and the accuracy of the predicted cluster permutation. S2S methods, in turn, tended to outperform NCP, CCP and ACP at the ordering task, as evidenced by, among others, the structural score (StS) of the predicted synthetic catalogs. The application of NOC to the core PROCAT dataset resulted in best performance to-date, as showcased via Kendall's τ and V-measure. Examples of the rendered Incito catalogs obtained from PROCAT examples clustered and ordered by the NOC model are shown in Figure 3.6.

NOC respects all characteristics of the Catalog Problem, is capable of predicting a partitional clustering of sets of varying cardinalities, obtains diverse clusters that respect cluster-level cardinality constraints, displays a degree of relational reasoning, and orders the resultant clusters (or sections) into complete catalogs, respecting structural rules present in the supervision target. It is a predictive solution to the Catalog Problem, trained to yield a single "ground truth" PROCAT catalog given a set of products. Its main limitation is that it doesn't adhere to a fully generative formulation of the problem, which would respect that there exists a number of valid solutions for both the clustering and the ordering, as discussed in Chapter 1.3.4. This forms one of the potential areas of future work outlined in the conclusion of this thesis.

In summary, in Chapter 3 an overview of a number of the research contributions and proposed methods has been provided. All of these build on the foundation formed by the curated dataset of real-world product catalogs, named PROCAT, introduced as an early deliverable in Chapter 1.3.3. First of the discussed methods was a library

Figure 3.6: **NOC Predictions**. Examples of catalogs predicted by NOC and rendered by the Incito service. Each row consists of three sequential pages (screens), each sequence is from a different predicted catalog to display a representative variety.

for generating simplified, catalog-like structures from colour-coded atomic elements. Initially introduced along with the PROCAT dataset in the second included article in Chapter 5, these synthetic catalogs address some of the limitations inherent to PROCAT. Namely, they facilitate faster model training, contain multiple valid targets per identical input and include customizable, in-depth metrics that can afford the researcher greater insight into model performance. This took the form of a compositional score (CoS), which relates to the tested model's ability to compose valid sections, and a structural score (StS), which measures the model's ability to order sections into a valid catalog. The generation process of these synthetic catalogs can be adjusted through a flexible, easy-to-customize configuration that encodes the underlying compositional and structural rules. This opens the possibility of experimenting with rules not included by default and enables the researchers to separate these rulesets by the order of interaction required to be learned to abide by them.

Additionally, a set encoder method by the name of the Sit Interdependence Transformer was proposed and tested as one module in larger set-to-sequence architectures as an example of an incomplete approach to the Catalog Problem. SIT is a simple modification of the Set Transformer, with the addition of treating the permutation invariant representation of the set, obtained in the first layer through pooling by multihead attention, as another set element in subsequent layers. This was done in an effort to enable the learning of higher order interactions among all n elements of the input set in fewer than n layers and appears to result in improved performance in the experiments outlined in the third included article in Chapter 6.

Finally, the main model architecture designed as an example of a complete approach to the Catalog Problem was presented. This took the form of Neural Ordered Clusters, a method combining elements of set encoding, supervised clustering and pointer-style attention, with an added per-cluster cardinality prediction element. NOC is presented in more depth in the fourth included article in Chapter 7.

References

[1] M. Jurewicz and L. Derczynski, "PROCAT: Product catalogue dataset for implicit clustering, permutation learning and structure prediction," in *Thirty-fifth Conference on Neural Information Processing Systems. Datasets and Benchmarks Track*, 2021.

[20] A. Pakman, Y. Wang, C. Mitelut, J. Lee, and L. Paninski, "Neural clustering processes," in *International Conference on Machine Learning*, PMLR, 2020, pp. 7455–7465.

[22] Y. Wang *et al.*, "Amortized probabilistic detection of communities in graphs," *arXiv:2010.15727*, 2020.

[10] M. Zaheer, S. Kottur, S. Ravanbhakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 3394–3404.

[11] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh, "Set transformer: A framework for attention-based permutation-invariant neural networks," in *International conference on machine learning*, PMLR, 2019, pp. 3744–3753.

[17] Y. Yin *et al.*, "Enhancing pointer network for sentence ordering with pairwise ordering predictions," in *AAAI*, vol. 34, 2020, pp. 9482–9489.

[29] M. Jurewicz and L. Derczynski, "Set interdependence transformer: Set-to-sequence neural networks for permutation learning and structure prediction," in *31st International Joint Conference on Articial Intelligence, IJCAI-ECAI 2022*, 2022.

[30] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure," in *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, 2007, pp. 410–420.

[31] Q. Huang, H. He, A. Singh, Y. Zhang, S.-N. Lim, and A. Benson, "Better set representations for relational reasoning," *Proc. NeurIPS*, vol. 34, 2020.

[118] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[171] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[172] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," *Advances in neural information processing systems*, vol. 28, 2015.

[183] R. Van Der Linden, R. Lopes, and R. Bidarra, "Procedural generation of dungeons," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 1, pp. 78–89, 2013.

[184] N. Parmar *et al.*, "Image transformer," in *International conference on machine learning*, PMLR, 2018, pp. 4055–4064.

[185] K. Han, A. Xiao, E. Wu, J. Guo, C. Xu, and Y. Wang, "Transformer in transformer," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15 908–15 919, 2021.

[186] N. Engel, V. Belagiannis, and K. Dietmayer, "Point transformer," *IEEE Access*, vol. 9, pp. 134 826–134 840, 2021.

[187] C. He, R. Li, S. Li, and L. Zhang, "Voxel set transformer: A set-to-set approach to 3d object detection from point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 8417–8427.

[188] A. Santoro *et al.*, "A simple neural network module for relational reasoning," *Advances in neural information processing systems*, vol. 30, 2017.

[189] E. Wagstaff, F. B. Fuchs, M. Engelcke, M. A. Osborne, and I. Posner, "Universal approximation of functions on sets," *Journal of Machine Learning Research*, vol. 23, no. 151, pp. 1–56, 2022.

[190] R. L. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro, "Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs," in *International Conference on Learning Representations*, 2018.

[191] W. Kay *et al.*, "The kinetics human action video dataset," *arXiv:1705.06950*, 2017.

[192] C.-Y. Ma, A. Kadav, I. Melvin, Z. Kira, G. AlRegib, and H. P. Graf, "Attend and interact: Higher-order object interactions for video understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6790–6800.

Chapter 4

Article 1

Article details:

1. **"Set-to-Sequence Methods in Machine Learning: a Review"**
   Mateusz Jurewicz and Leon Derczynski
   Journal of Artificial Intelligence Research (JAIR), volume 71: 885-924, 2021.

Context. At the time of writing the first published article there appeared to be no single point of entry for researchers interested in machine learning methods that take as input sets of elements and return a sequence, let alone such neural-network-based methods in particular. Although the term *"set-to-sequence"* would appear sporadically in papers that build on earlier sequence-to-sequence models (often referred to as seq2seq), there was a lack of a clear and consistent definition. Within the context of this thesis, the article is a wider and more detailed look at set encoding and permutation learning methods, a more focused and ultimately relevant selection of which was introduced in Chapter 2.

Contributions. The article provides a comprehensive entry point with respect to set-to-sequence methods in machine learning. It defines the field, contrasts it with a number of related areas and presents important foundational concepts. It also firmly grounds the field in multiple examples of practical application and provides a qualitative comparison of the described set encoding methods. Every included model is presented in significant detail, with a discussion of its limitations and notes on

proven areas of use. Best effort is made to compare the models' performance on reported datasets, where applicable.

Recent developments. As the article with the least recent publication date, several set encoding and permutation learning methods have been proposed since. Within learning representations of sets of any cardinality many adaptations of the included Set Transformer have been proposed. These include the Latent Variable Sequential Set Transformer [120] in the area of multi-agent motion prediction, the Point Transformer [121] within semantic scene segmentation on point clouds with positional encoding and the *Perceiver* proposed by Jaegle *et al.* [206], with a proven record of scaling to inputs with cardinality of a hundred thousand. Additionally, optimal transport kernel embedding or OTKE [207] has been proposed to mitigate the computational cost of set transformers, with some approaches deriving it through Expectation-Maximization [208]. Additionally, as referred to in Chapter 2, reinforcement learning methods have since been successfully applied to combinatorial optimization problems such as learning a permutation of a set of elements, as seen in the work of Wu *et al.* (2021) and Kool *et al.* (2022). These could constitute a separate but important update to the article section regarding ordering methods and different formalizations of the set-to-sequence task's target output.

4.1 Set-to-Sequence Methods in Machine Learning: a Review

In this section, the first published article is included in unchanged form, starting on the next page for ease of reading.

Journal of Artificial Intelligence Research 71 (2021) 885 - 924 Submitted 03/2021; published 08/2021

Set-to-Sequence Methods in Machine Learning: a Review

Mateusz Jurewicz MAJU@ITU.DK, MJ@TJEK.COM
Department of Computer Science
IT University of Copenhagen, 2300 Copenhagen, Denmark
Tjek A/S, 1408 Copenhagen, Denmark

Leon Derczynski LEOD@ITU.DK
Department of Computer Science
IT University of Copenhagen, 2300 Copenhagen, Denmark

Abstract

Machine learning on sets towards sequential output is an important and ubiquitous task, with applications ranging from language modelling and meta-learning to multi-agent strategy games and power grid optimization. Combining elements of representation learning and structured prediction, its two primary challenges include obtaining a meaningful, permutation invariant set representation and subsequently utilizing this representation to output a complex target permutation. This paper provides a comprehensive introduction to the field as well as an overview of important machine learning methods tackling both of these key challenges, with a detailed qualitative comparison of selected model architectures.

1. Introduction

We begin by providing a definition of the set-to-sequence field and outline its importance in various areas of application.

1.1 What is Set-to-Sequence?

Set-to-sequence encompasses a group of problems where input takes the form of unordered collections of elements and the output is an ordered sequence. These challenges can be approached as a machine learning problem, where models learn arbitrary functions for performing the set-to-sequence mapping.

Set-to-sequence covers combinatorial optimization and structure prediction problems where exhaustive search is often not tractable. Machine learning (ML) approaches to set-to-sequence combine set-encoding techniques with permutation learning and have found an exceptionally wide range of practical applications.

Many of the successful deep learning approaches take advantage of the structure in their input data. However, sets do not posses the kind of internal structure that images and natural language sentences do. In set-to-sequence our input data does not have an inherent order and therefore our models must take into consideration the *permutation invariance* of sets. Obtaining meaningful permutation invariant representations is an important challenge for machine learning models in order to ensure that the same set will not result in two different outputs, due to the arbitrary initial order in which its elements were presented to the model.

1.2 Why Does Set-to-Sequence Matter?

Machine learning set-to-sequence methods can approximate solutions to computationally intractable problems in many areas. They have been applied to learning competitive solvers for the NP-Hard Travelling Salesman Problem (Vinyals et al., 2015); tackling prominent NLP challenges such as

sentence ordering (Wang & Wan, 2019) and text summarization (Sun et al., 2019); and in multi-agent reinforcement learning (Sunehag et al., 2018). A notable example is the agent employed by the AlphaStar model, which defeated a grandmaster level player in the strategy game of Starcraft II (Vinyals et al., 2019).

Set-to-sequence ML models also play an important role in data-intensive 3D point cloud processing (Qi et al., 2017) and meta-learning (Huang et al., 2018b). Set-input and set-ordering problems themselves are prominent in a wide array of applications ranging from power grid optimization (Cui et al., 2019), where solving them led to power usage savings of up to 30%, through anomaly detection (Jung et al., 2015) to measurements of contaminated galaxy clusters (Ntampaka et al., 2016).

This review contributes to the field in two primary ways:

1. By providing a single point of entry for researchers interested in the set-to-sequence field and for applied practitioners solving set-input ordering challenges.

2. By comparing all the discussed methods via a number of aspects relevant for both academic and applied work and presenting this comparison in the form of easy-to-read tables, which will help guide the reader towards the most applicable method for their specific area of interest.

The remainder of this paper is structured in the following way: firstly, we introduce the reader to the necessary background concepts and related work in section 2, including specific notes on the adopted notation. Secondly, an overview of set encoding methods is given in section 3, with comparison tables and details of the underlying mathematical transformations. Thirdly, section 4 contains a survey of popular ordering methods, which use the encoded set representation to output a complex permutation. The lists provided in sections 3 and 4 are not exhaustive and focus primarily on deep learning approaches. Finally, a discussion of current limitations and directions for further research is given in section 5, followed by a short conclusive summary in section 6.

2. Background

In this section the reader is introduced to the key concepts related to machine learning on sets and permutation learning, with minor notes on notation throughout the rest of the paper. Additionally, a comprehensive overview of related work from other fields of machine learning is given, including natural language processing, information retrieval and set segmentation.

2.1 Important Concepts

For the purposes of this review a *set* can be intuitively defined as a collection of distinct elements, without a canonical order between them (Halmos, 2017). An important property of sets is that they can have a varying number of elements, also referred to as their *cardinality*. Whilst the intuitive definition of a set is susceptible to known paradoxes (Rang & Thomas, 1981), the machine learning methods discussed here do not require an in-depth understanding of the proper definition from axiomatic set theory. Interested readers can find further information pertaining to it in other referred publications (Takeuti & Zaring, 2013).

As per the axiom of extensionality, sets are defined only by their elements (Hayden et al., 1968). In practice this means that given, for example a set $A = \{x, y, z\}$ and set $B = \{z, y, x\}$ we know that $A = B$. The order in which the elements are presented in roster notation does not matter. From now on, when we refer to a set, we specifically limit our considerations to finite sets only.

Set-to-sequence ML methods are distinctly different from earlier, encoder-decoder sequence-to-sequence model architectures (Sutskever et al., 2014). The difference stems from having to handle set-input data. This imposes two requirements on set-to-sequence ML methods that are not met by most neural network models:

1. *Permutation Invariance*

   The output of the model must be the same under every possible permutation of the elements from the input set.

2. *Varying Input Length*

   The same model must be able to process input sets of different lengths.

If these criteria are not met, the ML model by definition treats its input as a sequence, not a set. Fully feed-forward methods fail to meet both criteria and the recurrent neural networks (RNNs), which form the foundation of most sequence-to-sequence autoencoders, are sensitive to alterations of the order of their input (Vinyals et al., 2016). To truly treat input data as an inherently unordered set we must be certain that permuting the input will not result in a different encoded set representation (Zaheer et al., 2017). Additionally, depending on the presence and type of a downstream task that uses this representation, we are interested in whether the final output is also permutation invariant, which is not necessarily the case with all reordering methods.

More formally, a function $f : \mathcal{P}(X) \to Y$ is permutation invariant regarding the order of the elements of its input set if for every permutation π the following property holds: $f(\{x_1, \ldots, x_n\}) = f(\{x_{\pi(1)}, \ldots, x_{\pi(n)}\})$. A related property of functions on sets, which has been formally investigated by Zaheer et al. (2017), is *permutation equivariance*. In tasks where each set element has an associated target label, such that these individual labels depend on the entirety of the set, we would ideally want our predicted labels to remain the same per element, regardless of how the original input set is permuted. That property is permutation equivariance.

At this point it is important to distinguish between two types of set-to-sequence challenges. In the first type the output is a reordering of the input elements, with the possibility of repeating an element multiple times in the output sequence or skipping it entirely. Such permutations with potential repetition and exclusion are further denoted as *complex permutations*. We can refer to the type of problems involving various kinds of permutations of the input elements as *self-referential* set-to-sequence challenges.

The self-referential set-to-sequence domain includes classic combinatorial optimization problems and forms the majority of this review. There are many different ways to frame this task and formalize the resulting output, which are discussed in section 4. They include primarily pointer-based attention (4.2), the generation of permutation matrices (4.3) and ranking scores (4.4).

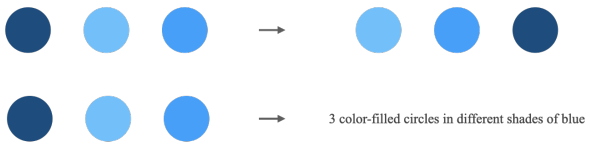


Figure 1: Set-to-Sequence Tasks by Referentiality

In the second type of set-to-sequence challenges, the task is to generate output that is sequential in nature, but is not defined as a permutation of the original input elements. We denote this as *non-referential* set-to-sequence. It encompasses for example summarization of a set of documents. Here, the input is indeed a set, with unique elements that do not have a canonical ordering to them. The output is a sequence of natural language tokens in the form of a human-readable summary, without referring directly to the elements of the input set. This area is only partially covered by this review.

For a simple visual explanation highlighting the difference between self-referential (top) and non-referential (bottom) set-to-sequence tasks, see Figure 1. In both cases the input is a set of 3 disks of varying shades of blue. In a self-referential setting the target might be a permutation from lightest to darkest. In a non-referential setting the target may be a sequence of word tokens describing the input set.

Specifically, all descriptions of set-input encoding methods are shared between the two types of set-to-sequence problems and therefore will be of value to readers interested in either. However, for considerations related to sequence prediction in general, areas of interest include recurrent neural network encoder-decoder models (Sutskever et al., 2014), reinforcement learning actor-critic methods (Bahdanau et al., 2017) and fully-connected transformer architectures as investigated by Vaswani et al. (2017), Devlin et al. (2019) and Brown et al. (2020).

In summary, the former type of set-to-sequence ML architectures, which this review focuses on, tackle two primary challenges:

1. Handling varying-length set-input data in a way that ensures permutation invariance
2. Handling outputs as complex permutations or *reorderings* of the original input

The first challenge, once solved, allows us to use machine learning methods to perform set-input regression, classification, recommendation (Vartak & Thiagarajan, 2017), as well as clustering problems and more (Edwards & Storkey, 2017).

Depending on the specific task at hand, the permutation invariant representation of the input set may also be required to encode higher order interactions between the input set elements, as seen in the work of Lee et al. (2019) and Zhang et al. (2020), which is a separate but important consideration in the area of encoding sets.

The second challenge of *permutation learning* is made simpler by solving the first one, but has also been successfully tackled without addressing it (Vinyals et al., 2015). It focuses on learning the proper order of arbitrary input elements. As a result, the model learns to predict the best structure of the output sequence composition.

2.2 Difficulty of Learning to Reorder

Permutation learning is an inherently difficult challenge. Even a relatively simple application of set-to-sequence methods to the Travelling Salesman Problem (TSP) in a two dimensional Euclidean space involves tackling an NP-Hard problem (Vinyals et al., 2015).

Whilst highly successful, polynomial time algorithms for obtaining good approximate solutions to this task do exist, such as the ones proposed by Arora (1996) and Karlin et al. (2020), it is also valuable to investigate the capacity of current machine learning techniques to learn them iteratively.

In the two dimensional Euclidean version of the TSP our input is a set of point coordinates and our desired output is a permutation of these points in a way that results in the shortest distance travelled between them. Additionally, we must not skip nor revisit any of the points. For a visual example, see Figure 2.

This challenge is difficult because the number of possible permutations increases factorially in the cardinality of the input set. It is a self-referential set-to-sequence problem due to its input having no inherent order and the desired output being a permutation of the input elements. Given the same set of points as input, we expect the output to be the same tour, regardless of the order in which they are originally presented.

Further difficulties arise when we consider how to formalize the resulting reordering. One possible method involves the use of the aforementioned permutation matrices, which are discrete and therefore do not lend themselves to direct use of gradient-based backpropagation without a relaxation of the concept (Emami & Ranka, 2018). This and other formulations of representing a reordering (mentioned below) are discussed in more detail in section 4.

Figure 2: Travelling Salesman Problem

An example of a TSP input is given as a set of points in a 2D Euclidean space (top left) and an arbitrarily ordered array (bottom left). Individual dots represent x, y coordinates. As output, we see the shortest path between points (top right) and an array representing their target order (bottom right).

Such difficulties have motivated researchers to instead investigate differentiable, attention-based methods that involve pointing to the elements of the original set to define their permuted order. This approach often involves potentially computationally expensive beam search during inference.

Finally, if we forego the requirement of handling varying-length inputs, we can look to traditional learn-to-rank approaches for inspiration. In such frameworks the reordering is formulated as the assignment of a relevance score to each element, followed by sorting the elements according to that score, in monotonic order. However, sorting is a piecewise linear function, which therefore may contain many kinks where it is not differentiable. As a result, differentiable proxies to the sorting operator have been developed, but they did not achieve the expected $O(n \log n)$ time complexity until a method consisting of a projection onto a *permutahedron* was proposed by Blondel et al. (2020).

Alternatively, in learn-to-rank, our model may be trained to return *ranks*, i.e. positions of the input elements in the target (properly ordered) sequence. These ranks are piecewise constant functions, with derivatives that are either null or undefined, preventing gradient-based learning. However, significant progress has been made towards directly approximating ranking metrics (Rolínek et al., 2020) and constructing differentiable sorting and ranking operators (Blondel et al., 2020). Additionally, Engilberge et al. (2019) propose a deep neural net which can act as a differentiable proxy for ranking, allowing the use of traditionally non-differentiable metrics such as Spearman's rank-order correlation (Spearman, 1904) as loss functions.

2.3 ML on Sets and Combinatorial Optimization

Set-to-sequence combines techniques from the field of machine learning on sets and combinatorial optimization. The former covers research areas related to both set-input and set-output problems, of which set-to-sequence is only concerned with the first kind. The latter consists of finding an optimal object from a finite set of objects and is strongly related to many forms of ordering tasks. The canonical example is the aforementioned TSP, which in itself has a long history of attempts at solving it through the most popular machine learning methods of the time, for example Smith (1999), Pihera and Musliu (2014), Ishaya et al. (2019) and Bengio et al. (2020).

Combinatorial optimization as such is of vital importance to modern industry applications. Consider the archetypal *Vehicle Routing Problem* (VRP), which poses the task of finding an optimal set of routes for a fleet of vehicles aiming to deliver goods to a given set of locations. The quality of the solution is determined by the *global transportation cost*. In the simplest variant of VRP, this is dependent on the sum of the lengths of tours for all vehicles. This effectively requires an ordering

of the locations into optimal trips, per each vehicle. Given the scale of modern logistical challenges and the environmental impact of freight, it is understandable that there have been many attempts to apply recent machine learning developments to such problems (Ibrahim, R, & Ishaya, 2019).

Current state-of-the-art combinatorial optimization algorithms often rely on handcrafted and hard-to-maintain heuristics for making decisions that are otherwise computationally infeasible or not well defined mathematically, for example Bello et al. (2016). It is a natural area of application for machine learning research and has been approached through the use of graph-based methods (Dai et al., 2017), reinforcement learning (Nazari et al., 2018) and attention mechanisms (Kool et al., 2019). For a comprehensive survey of the wider intersection of combinatorial optimization and machine learning, see Bengio et al. (2020).

2.4 Notation

The paper follows the notational conventions that are most common in literature. Scalar values are marked with lower case italics x_i, vectors with lower case bold typeface $\mathbf{x}$, matrices with capital case italics X. These matrices may be used to represent sets, in which case they are presented through roster notation with curly brackets, for example $X = \{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$.

However, sets may also consist of scalar elements, in which case a capital letter is still used to represent them: $A = \{a_i, \ldots, a_n\}$. Given the importance of differentiating between unordered sets and ordered sequences, the latter are represented through angled brackets $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ for additional clarity. When indicating the index of an element within a vector, whose symbol already contains a subscript (e.g. $\mathbf{v_j}$) the index of the scalar element is given in the superscript (v_j^i).

Individual permutations are marked as π, such that an example $\pi_i = \langle 3, 2, 1 \rangle$, consisting of integer indices referring to the original sequence $\mathbf{x} = \langle x_1, x_2, x_3 \rangle$, would result in the reordered sequence $\mathbf{x}^\pi = \langle x_3, x_2, x_1 \rangle$. In some cases, if the order of elements in the original sequence is nontrivial, a permutation π can also be given in two-line notation making both $\mathbf{x}^\pi$ and the integer indices explicit:

$$\pi = \begin{pmatrix} x_3 & x_2 & x_1 \\ 3 & 2 & 1 \end{pmatrix} \tag{1}$$

2.5 Connections with Other ML Fields

In this section, a brief overview of other related topics from different fields of machine learning research is given. The aim is to point the reader who may only be tangentially concerned with set-to-sequence tasks to the appropriate area within their main field of interest. A reader with a decided focus on set-to-sequence is encouraged to continue reading section 3 directly.

2.5.1 Natural Language Processing (NLP)

There is a number of cases from the field of *Natural Language Processing* (NLP) that require tackling similar challenges to the ones faced by set-to-sequence methods. The popular sequence-to-sequence, encoder-decoder framework proposed by Sutskever et al. (2014) can itself, in principle, be applied to set-to-sequence problems, but does not perform well in practice. An example of such a case is the work on word ordering tasks, also known as *linearizations*, towards syntactically plausible word representations (Nishida & Nakayama, 2017).

The authors use sentences of ordered words to train the network to output a binary permutation matrix. When the original input, in the form of randomly ordered words from the target sentence, gets matrix-multiplied by this permutation matrix, the proper order is recovered. The network, referred to by the authors as the *Word Ordering Network* (WON), is an example of one way to formalize an ordering task. It can be seen as a simplification of the *Pointer Network* encoding method (Vinyals et al., 2015), discussed in more depth in a later section, whilst more closely resembling

the classic sequence-to-sequence models in the decoder (where it sequentially outputs rows of the permutation matrix).

Another example of an NLP task where permuting plays a key role is sentence ordering and order discrimination. The goal is to take a set of sentences and order them back into the original paragraph. Historically, this area of research has been dominated by hierarchical RNN-based approaches, which make use of LSTMs or GRUs in an auto-encoder framework (Logeswaran et al., 2018). First, a recurrent network is used to obtain the embedding of each sentence and then another to obtain the context representation of the entire paragraph.

More recent developments have seen the use of attention mechanisms to make it easier for this embedding to encode vital information regardless of the distance between information-carrying elements, as per the vanishing gradient problem. An example of this can be found in the *ATTOrderNet* architecture (Cui et al., 2018) and more recently in the *Set Transformer* (Lee et al., 2019), discussed in detail in section 3.5.

An example of an NLP set-to-sequence challenge where the output is not a permutation is the task of summarizing multiple documents into a single sequence of text as seen in the works of Ma et al. (2016) and Mani et al. (2018).

2.5.2 Ranking, Information Retrieval and Content Ordering

Another related, but succinctly distinct field comes in the form of ranking problems, information retrieval and content ordering. These encompass a family of challenges where there exists an *optimal* hierarchical order to the input elements, such that given two elements there is always a proper way in which they should be placed in relation to one another, which does not change depending on the other input-set elements.

More specifically, in the context of ranking for search, if we enter the query 'cats', the returned image of a cat should always rank higher than an image of a 'dog', regardless of what the other returned images may contain. This assumption is not always true in complex set-to-sequence problems, where any new element of the input-set can change what the proper relative order or structure of the already available elements' sequence should be.

Traditionally, learn-to-rank problems have been tackled in a pairwise manner (Cohen et al., 1998), later approaches have applied neural methods on a list-based formulation of this problem (Cao et al., 2007). Ranking has also found application in content selection (Puduppully et al., 2019) and been employed as a useful auxiliary objective in a multitask setting for regression problems (Liu et al., 2019). A detailed look at listwise ranking approaches to ordering and structure prediction can be found in section 4.4.

2.5.3 Set Regression, Classification and Segmentation

A more closely related area of work stems from set-input problems that have an output that is not a sequence. These include set regression, classification and segmentation challenges, among others. Effectively, this research field requires solving a near identical challenge to the first of two primary set-to-sequence challenges outlined at the beginning of this section, in that obtaining a proper encoding of the input set is vital.

Examples of such methods include *PointNets* for 3-dimensional point cloud classification and segmentation (Qi et al., 2017), which builds on previous work by Vinyals et al. (2016) in a specific geometric setting requiring both rotation and translation invariance (see section 3.6.3). Another example comes in the form of techniques for labelling objects based on a set of images from multiple viewpoints such as security cameras (Zhao et al., 2019a) and even fully convolutional models for set segmentation (Oliveira et al., 2020).

Methods that obtain the input set representation in a way that is interesting to set-to-sequence problems are included in the main section of this review and given appropriate focus, regardless of whether their original application was in sequence-output challenges.

2.5.4 Set-Output Tasks, Including Set-to-Set

Conversely, a large field of work revolves around methods that learn to generate or predict a set as their output. Sets are the natural representation for many kinds of output data in machine learning tasks. These include a set of objects present in an image in an object detection setting (Zhao et al., 2019b), a group of points in a point cloud (Achlioptas et al., 2018) and a selection of nodes in a molecular graph for the problem of molecule generation (De Cao & Kipf, 2018).

The main challenge of set-output methods mirrors the primary challenge of representing sets in a permutation invariant way in set-input problems. If the order in which the model outputs elements does not matter, there are $n!$ equivalent, correct outputs that the model has to learn to consider equally good.

For example, imagine a simple task where the model must learn to take as input a set of integers and return the set of all primes present in the input. Specifically, given the input $A = \{1, 2, 3, 4, 5\}$ the correct output should take the form of the set $B = \{2, 3, 5\}$. However, given that ML model implementations operate on ordered, multidimensional arrays, the model must learn to treat all of these possible output sequences as equally correct: $\langle 2, 3, 5\rangle, \langle 2, 5, 3\rangle, \langle 3, 2, 5\rangle, \langle 3, 5, 2\rangle, \langle 5, 2, 3\rangle, \langle 5, 3, 2\rangle$.

Failure to properly account for this property of output sets leads to discontinuities that are difficult for most modern neural architectures to learn, even on seemingly trivial synthetic datasets (Zhang et al., 2020). An example that illustrates this comes in the form of an autoencoder trained to embed and then reconstruct input consisting of a set of n 2-dimensional points forming a regular polygon.

Every example in this dataset is a rotation of the same polygon around the origin. The discontinuity arises from this rotation, which forces a switch with regards to which element of the input set the model's output neurons will be responsible for decoding. This is referred to as the *responsibility problem*. Proper handling of the set structure in the output requires the application of permutation invariant and permutation equivariant operations, much like in set-input problems, where the responsibility problem is not present.

Notable recent methods in the field of set prediction include the *Deep Set Prediction Network* (DSPN) by Zhang et al. (2019), which consists of a deep learning vector-to-set model that enables permutation invariance, and the *Transformer Set Prediction Network* (TSPN) by Kosiorek et al. (2020), that additionally takes advantage of the multiheaded self-attention introduced by Vaswani et al. (2017). The TSPN addresses the limitations of the DSPN related to set-cardinality learning. Additionally, an iterative attention mechanism referred to as *Slot Attention* has been proposed by Locatello et al. (2020), which decomposes input features into a set of representations, lending itself to set prediction tasks.

A sub-field of interest within set prediction is referred to as set-to-set, where both the input and output are structured as a set. Such tasks include recommendation (Sarwar et al., 2001), image search (Wang et al., 2014) and person re-identification (Zheng et al., 2015). Set-to-set challenges require both permutation invariance in the parts of the model that encode the input set and a proper cross-similarity function for the output sets (Saito et al., 2019), circumnavigating the responsibility problem.

2.5.5 Ensuring Other Types of Invariance

Finally, it may be of value to mention methods that obtain types of invariance other than under permutation. These methods stem from areas of application where the model's final prediction should not be dependent on such predefined transformations. Examples include translational and rotational (also known as viewpoint) invariance, common in computer vision problems, addressed in the works of Ling et al. (2016) and Marcos (2016). Taking as illustrative case the task of object detection, to ensure the former quality the model needs to recognize the same object regardless of its position within the input image. To ensure the latter, given three dimensional images of a scene, the model must correctly identify an object regardless of the angle from which it is being perceived.

An important traditional approach to learning models that are invariant to certain transformation is data augmentation, as seen in Taylor and Nitschke (2018) and Hernández-García et al. (2019). Here, the underlying idea is that the model will be able to learn to become invariant to the chosen transformations once we augment our data with instructive examples that have been appropriately translated, rotated, partially obscured, blurred, illuminated, resized or had other applicable transformation applied to them, often in combination. More recent approaches that aim to safeguard machine learning models against relying on unintended aspects of data through adversarial strategies have also been proposed (Jaiswal et al., 2018). In order to apply the data augmentation approach to the set-to-sequence domain, we would increase our training set up to $n!$ times, providing the model with every possible permutation of each example set.

To prevent the costs associated with a larger training set, machine learning methods commonly employ various pooling operators after the stacked, equivariant feature extraction layers to obtain the desired invariance. However, in the case of translation invariance through pooled convolutional operations, the assumption that this completely prevents the model from exploiting the absolute location of an object in an image has been challenged (Kayhan & Gemert, 2020).

3. Set Encoding Methods

In this section, a qualitative comparison of different set encoding ML methods is provided, followed by detailed subsections devoted to the individual model architectures.

3.1 Method Comparison

This section introduces the reader to each of the relevant set-encoding methods in turn. These are also sometimes referred to in literature as *set-pooling* methods (Lee et al., 2019). Additionally, a number of comparison tables provides a summary overview: 3.1, 3.1.1.

It is important to note that some of the methods discussed in this section were designed specifically to handle set-to-sequence problems. As such, they contain both a set encoding module and a permutation outputting mechanism. Since it is not always immediately obvious how to combine a method that handles set encoding with a method that is designed to output a reordering, we compare various aspects of these methods in multiple places.

The models are compared with regards to the following aspects:

1. **Permutation Invariance**: whether the model obtains a permutation invariant representation of the input set. The same set must result in the same embedded representation, regardless of how the actual input array was permuted.

This feature does not guarantee that the final output of the model will be the same for differently ordered sequences obtained from a single set, as that may depend on the ordering method applied to the obtained permutation invariant set embedding, in order to output a sequence. This stems from the fact that these methods may require the reintroduction of the information relating to the order of the original input array and refer to it directly when outputting a permutation - particularly pointer-based attention. For more information on this, see section 4.

2. **Multiset Input**: whether the model can distinguish between a given input set and certain examples of its corresponding multisets, with repeated elements.

For example, the average() pooling operator will not be able to distinguish between a set $X = \{1, 2, 3\}$ and a multiset $X' = \{1, 1, 2, 2, 3, 3\}$. The max() operator will similarly fail in the case of $X'' = \{1, 1, 2, 3\}$.

3. **Complexity**: how the model's computational complexity relates to the cardinality n of the input set, and possibly other hyperparameters specific to a given architecture.

4. **Applications**: selected prominent domains which the model has been successfully applied to. Further examples can be found in the later sections devoted to each model.

Additionally, the most prominent architectures can be split into RNN-based methods, namely the Pointer Network and Read-Process-and-Write model, and the more recent fully connected ones, primarily the foundational DeepSets method and the Set Transformer.

Alternatively, an interesting way to distinguish between them would be to consider methods that depend on a variation of the attention mechanism introduced by Bahdanau et al. (2015), such as the Pointer Network and the Set Transformer, and others that do not utilize it.

Model	Perm. Invariance	Multiset	Complexity
Pointer Network (2015)	No	No[1]	$O(n^2)$
Read-Process-and-Write (2016)	Yes	Yes	$O(n^2)$
PointNet (2017)	Yes	No[2]	$O(n)$
DeepSets (2017)	Yes	Yes[3]	$O(n)$
Janossy Pooling (2019)	Yes	Yes	$O(n!)$
Set Transformer (2019)	Yes	Yes	$O(n^2)$
AttSets (2020)	Yes	Yes	$O(n)$
FSPooling (2020)	Yes	Yes	$O(n \log^2 n)$
RepSet (2020)	Yes	Yes	$O(mn + n^2 \log n)$

Table 1: Comparison of set encoding methods, part 1

3.1.1 Notes on Complexity

The complexity of the Read-Process-and-Write method is additionally impacted by the number t of steps in the *Process* block that computes the permutation invariant embedding of the input set. t is constant, but an interesting area of further research would be to learn it adaptively, similar to the method described by Graves (2016). The complexity of PointNet becomes $O(n^2)$ for 2D images and $O(n^3)$ for voxels, due to the convolutional operations. The authors of Janossy Pooling propose 3 methods of balancing tractability and the model's representational power, as outlined in section: 3.6.2

Regarding the Set Transformer, the use of l stacked SAB layers results in quadratic complexity of $O(n^2 l)$, use of a stack of l proposed ISAB layers with m inducing points results in complexity of $O(nlm)$. Similarly to the PointNet architecture, the complexity of AttSets grows depending on the dimensionality of the input, due to the use of convolutional layers in the encoder. However, the authors also provide a novel training paradigm, called *FASet*, and benchmark its mean time consumption for a single object against a selection of simple pooling methods, with favourable performance.

Regarding the RepSet method, m is the chosen number of hidden sets for the bipartite matching algorithm, represented by the columns of a trainable matrix. For more information on this and a proposed, more tractable relaxation, see section 3.6.5.

1. The Pointer Network does not treat its input properly as a set, therefore it cannot be said to properly handle multiset input either, but it will distinguish between input vectors with repeated elements.
2. Due to experimental results on the selected tasks, the authors of PointNet settle on max() as their recommended pooling operator, which does not distinguish between certain multiset variants. However, they report robust measurements of the performance of other pooling methods which can easily be included in the final model architecture and provide comparable results.
3. Depends on the pooling operator used after the stacked fully-connected layers, of which the authors of DeepSets primarily focus on sum(), which does distinguish between sets and multisets. However, max(), which does not

Model	Applications
Pointer Network (2015)	combinatorial, multi-agent
Read-Process-and-Write (2016)	combinatorial, sorting
PointNet (2017)	3D shape classification and segmentation
DeepSets (2017)	set expansion, anomaly detection
Janossy Pooling (2019)	arithmetic, graph classification
Set Transformer (2019)	amortized clustering, anomaly detection
AttSets (2020)	3D shape reconstruction
FSPooling (2020)	set and graph classification
RepSet (2020)	text and graph classification

Table 2: Comparison of set encoding methods, part 2

3.1.2 Notes on Dataset Performance

The listed set encoding methods can be applied to a wide spectrum of tasks. As a result, their performance has been tested on a variety of datasets, often in subtly different settings, which prevents direct comparison. In lieu of a table presenting their performance on a selected subset of such datasets, we provide a short discussion of the experimental results that do lend themselves to being compared. A more comprehensive experimental analysis in this area is a possible direction for future work.

Both Pointer Networks and the Read-Process-and-Write (RPW) method have been tested on the simple task of sorting a set of five floating point numbers between 0 and 1 (Vinyals et al., 2016). The Pointer Network achieved an accuracy of 0.90 compared to 0.94 reached by the RPW. Additionally, the RPW method appeared to be better at handling larger sets of floats. Both DeepSets and Janossy Pooling (Murphy et al., 2019) have been tested on simple arithmetic tasks such as sum-of-digits prediction, with each method reaching an accuracy of 1.0, albeit tested on input sets of different cardinalities. The Set Transformer has instead been tested on maximum value regression.

The Set Transformer, DeepSets and Janossy Pooling have also all been tested in terms of performance on unique count tasks. However, in the case of the Set Transformer experiments were performed on sets of handwritten characters from the Omniglot dataset (Lake et al., 2019), in the case of DeepSets on the MNIST8m hand-written digits (Loosli et al., 2007) and in the case of Janossy Pooling on simple integer sets.

The most commonly shared experimental task in the papers introducing and consequently utilizing the listed methods was point cloud classification. Particularly the ModelNet40 dataset (Wu et al., 2015) has been used to test four of the mentioned models. Whilst AttSets (Yang et al., 2020) employs it to formulate a multi-view reconstruction task, the other three methods are tested on the core classification task with PointNet reaching an accuracy of 0.892 (Qi et al., 2017), DeepSets 0.900 (Zaheer et al., 2017) and the Set Transformer 0.904 (Lee et al., 2019). However, the specific methods used to produce the point clouds from the provided mesh representation of objects showcased certain differences, further highlighting the need for a systematic, uniform comparison.

Both the Set Transformer and DeepSets methods have been tested on the task of set anomaly detection, specifically by way of the CelebA dataset (Liu et al., 2015). However, the DeepSets model was tested in terms of accuracy (0.75) and the Set Transformer in terms of the area under receiver operating characteristic curve and area under precision-recall curve, preventing direct comparison.

is also proposed as a problem-dependent variation. The formal proof extending DeepSets to multiset inputs was given by Xu et al. (2019).

The FSPool technique's performance has been compared to the Janossy Pooling method through a visual question answering task, employing the CLEVR (Johnson et al., 2017) dataset. The accuracy of the latter was reported as 0.97 ± 0.54, and of the former as 0.9927 ± 0.18 (Zhang et al., 2020).

Another useful task for the purposes of performance comparison is document classification, where given a document, the input to the model is the set of embeddings of its terms. DeepSets, Set Transformer and RepSet have been directly compared through their performance in this regard on 8 separate datasets (Skianis et al., 2020), with the Set Transformer consistently outperforming DeepSets, and RepSet outperforming both of the aforementioned methods.

Finally, the performance of DeepSets, Set Transformer and RepSet has been compared on the task of graph classification through the 5 datasets proposed by Kersting et al. (2016). The classification accuracy of DeepSets on the MUTAG dataset was 0.862, Set Transformer's was 0.877 and RepSet's 0.886. However, on the arguably more difficult IMDB MULTI dataset the Set Transformer outperformed RepSet, reaching an accuracy of 0.502, compared to 0.499. For a full overview, see the paper by Skianis et al. (2020).

Further details regarding the performance and limitations of presented methods are available in the sections devoted to them individually (below).

3.2 Pointer Networks

The Pointer Network (Vinyals et al., 2015) is an encoder-decoder neural network architecture including a modified attention mechanism, which allows it to learn a target reordering of input elements. It is the first deep learning method capable of taking sets as input and learning a desired permutation, resulting in complex output sequences.

Pointer Networks were originally designed to tackle combinatorial optimization problems with varying input sizes, which was their main advantage over previous sequence-to-sequence methods. A Pointer Network can be trained on inputs of varying length and has been demonstrated to generalize reasonably well to unseen lengths (Vinyals et al., 2015).

Additionally, Pointer Networks included a modification of the content-based attention mechanism introduced by Bahdanau et al. (2015) which made it possible to treat the output of the model as pointers to elements of the input sequence. This attention-based pointing is one of the most popular methods for giving models the ability to output a permutation of the original input, regardless of the way they encode the original set. Due to its importance as a purely element-ordering technique, it is separately described in further detail in section 4.2.

3.2.1 Pointer Networks Limitations

An important characteristic of Pointer Networks is that they do not strictly treat the input as a set, instead processing it solely through sequential recurrent neural networks. As a partial consequence they do not obtain a permutation invariant representation of the encoded set. This results in a situation where the same set can be represented as two differently ordered input arrays, leading to the model predicting two different outputs for it. Thus returning the optimal order is not guaranteed.

Another important limitation is that nothing is explicitly preventing the model from outputting an invalid reordering of the input set or sequence. This becomes apparent during early training, when the model points to the same elements of the input at various indices of the output sequence. However, this can be mitigated by the addition of a beam search mechanism to the decoder during inference or by progressive masking. In the latter case, the entry in the attention vector referring to an element that had already been pointed to is preset to an infinitely negative value at each successive iteration, preventing it from being pointed to again, at the cost of certain inductive bias being introduced into the model.

3.2.2 Pointer Networks Details

The Pointer Network consists of a recurrent neural network (RNN) encoder and an RNN decoder with a modified attention mechanism. The model obtains a content-based attention vector $\mathbf{a_j} \in \mathbb{R}^n$ at each decoder step j. This vector represents the conditional probability of each input element x_i being the correct one to be pointed to at this step, conditioned on all previous steps as well as the entire input sequence $\mathbf{x} = \langle x_i, \ldots, x_n \rangle$, in the form of all encoder hidden states $E = \langle \mathbf{e_1}, \ldots, \mathbf{e_n} \rangle$ obtained when the encoder block iterates over the input array.

For simplicity, we will assume that each element x_i must be pointed to exactly once, meaning that the output sequence of nonnegative integer pointers $\mathbf{y} = \langle y_i, \ldots, y_n \rangle \in \mathbb{Z}^n$ represents a valid permutation π, such that a sample target output $\mathbf{y}^\pi = \langle 0, 2, 1 \rangle$ would represent the reordered sequence $\mathbf{x}^\pi = \langle x_1, x_2, x_3 \rangle$ for the sample input $\mathbf{x} = \langle x_1, x_3, x_2 \rangle$. This will mean that when iterating over both encoder states $\mathbf{e_i}$ and decoder states $\mathbf{d_j}$, we will always be in range 1 to n. The input sequence $\mathbf{x}$ can itself consist of multidimensional elements, or such embeddings of each x_i can be obtained prior to the pointer network module through a chosen embedding layer.

The attention mechanism in the decoder block is as follows:

$$z_j^i = \mathbf{v^T} \tanh(W_1 \mathbf{e_i} + W_2 \mathbf{d_j}) \qquad \text{for } i \in (1, \ldots, n) \tag{2}$$

$$\mathbf{a_j} = \text{softmax}(\mathbf{z_j}) \qquad \text{for } j \in (1, \ldots, n) \tag{3}$$

$$P(y_i | y_1, \ldots, y_{i-1}, \mathbf{x}) = \mathbf{a_j} \qquad \text{for } j \in (1, \ldots, n) \tag{4}$$

Where $\mathbf{d_j}$ is the decoder's hidden state at the j-th output element, $\mathbf{e_i}$ is the encoder hidden state at the i-th input element, W_1, W_2 and $\mathbf{v}$ are trainable tensors. The $\mathbf{z_j}$ vector is of the same length as the input $\mathbf{x}$ and represents an output distribution over the dictionary of input elements. After the application of the softmax nonlinear activation function, turning it into $\mathbf{a_j}$, it becomes an attention vector. For a visual explanation, see Figure 3.

3.2.3 Pointer Networks Applications

The primary application of Pointer Networks are tasks where the target output is a reordering of the elements of the initial input. This reordering is based on pointers to indices of the original input sequence. Examples of such problems in currently active research areas include element sorting, coherence modeling (Logeswaran et al., 2018), word ordering (Cui et al., 2018) and sentence ordering (Wang & Wan, 2019), as well as summarization (Sun et al., 2019) and ranking in information extraction (Bello et al., 2018).

In the original paper, the Pointer Network models have been tested on challenging combinatorial optimization problems such as finding planar convex hulls, computing Delaunay triangulations and the Travelling Salesman Problem. Experiments have shown that even with computationally intractable, NP-Hard problems such as TSP, this model architecture was able to learn competitive approximate solutions, limited by the scale of the problem, with $n <= 50$ for the TSP.

Pointer Networks also found usage within the AlphaStar reinforcement learning model which defeated a grandmaster level player in the competitive real-time strategy game of Starcraft II (Vinyals et al., 2019). They were employed to help the agent manage the structured, combinatorial action space in conjunction with an auto-regressive policy.

3.3 Read-Process-and-Write Model

The Read-Process-and-Write (RPW) model is a neural network architecture consisting of three distinct blocks and aiming to obtain a permutation invariant representation of the input set, whilst learning a function mapping it to arbitrary target outputs (Vinyals et al., 2016).

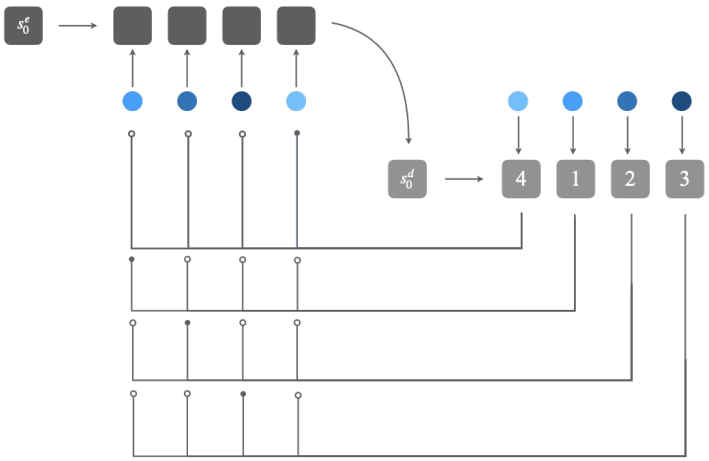


Figure 3: Pointer Network

In Figure 3, an encoding RNN sequentially processes each element of the input array (blue dots), encoding it into a hidden state s_n^e (dark gray), which is fed to the pointing decoder RNN (light gray). At every step, the second network produces a vector that represents a content-based pointer attention over the encoded inputs.

RPW satisfies the key property of obtaining a permutation invariant representation of its input through a variation of the attention mechanism. It can be seen as a special case of a *Neural Turing Machine* (Graves et al., 2014) or a *Memory Network* (Weston et al., 2015) in that it is a recurrent neural network model that creates a memory representation of each element in the input sequence and accesses it via an attention mechanism.

In the original RPW paper its authors also demonstrated that the order in which elements are organized as input has a significant impact on the learning performance of earlier sequence-to-sequence architectures. This is an important observation given the fact that the recurrent neural networks employed in them are, in theory, universal approximators (Schafer, 2007).

3.3.1 RPW Limitations

Whilst the RPW model constitutes a significant improvement in the way machine learning methods handle input sets, it suffers from the same limitation as Pointer Networks in terms of ordering their elements into the output sequence. Namely, it is not strictly prevented from pointing to the same element of the input set multiple times in the output, effectively returning either an invalid sequence or an incomplete reordering. This is a particularly important limitation in relation to handling *multisets* (also known as *msets* or *bags*), where the same element can occur multiple times in the input. However, it can be mitigated through beam search or progressive masking as described in the Pointer Network section. It also suffers from a significant decrease in performance as the size of the input set increases.

3.3.2 RPW Details

The RPW architecture consists of three distinct blocks:

1. *Read Block* - which embeds every element of the input set using the same neural network for each $x_i \in X$.
2. *Process Block* - which consists of a recurrent neural network that evolves its hidden state using a modified content-based attention mechanism to obtain a permutation invariant representation of the input over a separately predefined number of steps t.
3. *Write Block* - which takes the form of a Pointer Network in the set-to-sequence tasks but can also be another recurrent neural network decoder for tasks where the output elements come from a fixed dictionary.

The Process Block evolves the permutation invariant representation of the input set by repeating the following steps t times:

$$\mathbf{q_t} = \text{LSTM}(\mathbf{q^*_{t-1}}) \tag{5}$$

$$z^i_t = f(\mathbf{m_i}, \mathbf{q_t}) \tag{6}$$

$$a^i_t = \frac{\exp(z^i_t)}{\sum_j \exp(z^j_t)} \tag{7}$$

$$\mathbf{r_t} = \sum_i a^i_t \ \mathbf{m_i} \tag{8}$$

$$\mathbf{q^*_t} = \langle \mathbf{q_t}, \mathbf{r_t} \rangle \tag{9}$$

where i is the index over all embedded elements of the memory vector $\mathbf{m_i}$ obtained by the Read Block, $\mathbf{q_t}$ is effectively a query vector allowing us to read the permutation invariant representation $\mathbf{r_t}$ from the memories using an attention mechanism and $f()$ is any differentiable operation that takes two vectors and returns a scalar, most commonly the dot product $f(\mathbf{a}, \mathbf{b}) = \mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$.

An important implementation nuance is related to the third step, where the attention vector $\mathbf{a_t}$ is obtained via a softmax operation. Depending on weights initialization, that step can result in the undefined operation of dividing infinity by infinity. This can be prevented by bounding the value range of the $\mathbf{z_t}$ vector by the use of the tanh function (Logeswaran et al., 2018).

The LSTM() is a recurrent neural network that takes no inputs, only evolving the hidden state $\mathbf{q_t}$. The final set encoding $\mathbf{q^*_t}$, is obtained by concatenating the previous hidden state $\mathbf{q_t}$ and the permutation invariant representation $\mathbf{r_t}$. $\mathbf{q^*_t}$ becomes the hidden state input during the next iteration t of the Process Block.

3.3.3 RPW Applications

The RPW architecture has been applied to both continuous and discrete inputs. In the former case, the input can be a set of floating point numbers or a high-dimensional embedding of the entities of interest. In the latter, it can be dictionary entries. Considerations related to the specific structure of those input elements are out of scope for this paper as they pertain to the wider topic of representation learning.

This model architecture has also been used in few-shot object recognition (Xu et al., 2017), graph classification (Ying et al., 2018) and one-shot learning in the context of drug discovery (Altae-Tran et al., 2017). The original paper tests it on the problem of sorting a varying-size set of floating point numbers between 0 and 1. It achieves accuracy of 94% on sets of 5 elements, performance dropping significantly for larger ones (50% for 10 elements, 10% for 15).

This key property of obtaining a permutation invariant representation was further formalized in the DeepSets paper, presented in the following section.

3.4 DeepSets

The DeepSets framework (Zaheer et al., 2017) provides a robust mathematical analysis for designing *permutation invariant* and *permutation equivariant* deep learning models. Both of these concepts are explained in section 2.1. Where the Vinyals et al. (2016) paper focused on the former in the setting of a specific practical application, the authors of DeepSets provide a generic framework for the proper handling of set-inputs for both supervised and unsupervised learning.

The primary contribution of the DeepSets method lies in tackling the first part of the more general set-to-sequence challenge, which is to implement arbitrary set functions that result in permutation invariant representations. We have already described one example of an operation that ensured this in the RPW model's Process Block, namely a variation of the content-based attention mechanism consisting of a modified recurrent neural network. DeepSets propose a simpler, sum-based method to achieve this.

The DeepSets framework offers a simplified procedure by relying on summation of all element representations prior to further nonlinear transformations, which then transform these summed representations into the desired output (e.g. a class probability distribution or a single number for set regression). Later reimplementations of the DeepSets architecture experiment with replacing the sum operation with other permutation invariant alternatives such as taking the mean or maximum, with comparable results (Lee et al., 2019).

Additionally, the DeepSets analysis expands upon the set-input challenge by allowing for permutation equivariance, where the order of the output elements mirrors the order of the input sequence. This can be achieved, in one case, by adding a diagonal symmetry and diagonal identity constraint to the weights matrix of a fully-connected neural network layer, prior to the nonlinearity. However, set-to-sequence methods do not make extensive use of permutation equivariance therefore these are not detailed here. For more information, see the original paper.

3.4.1 Deep Sets Details

The proposed permutation invariant function for inference over sets takes the following general form:

1. Each element x_i in the input set is transformed *independently* into an embedded representation $\phi(x_i)$, possibly through multiple layers of a feed-forward neural network.

2. The representations $\phi(x_i)$ are summed together and the result is further processed using another network ρ consisting of any number of fully-connected layers with nonlinearities.

Both ϕ and ρ can be replaced by universal approximators, which can be learned to approximate arbitrary polynomials. In cases where additional information q is available, it can be used to obtain the conditional mapping $\phi(x_i|q)$. The key to permutation invariance in this framework is simply summation of the obtained per-element representations.

$$\text{DeepSets}(\{x_1, \ldots, x_n\}) = \rho(\text{sum}(\{\phi(x_1), \ldots, \phi(x_n)\})) \tag{10}$$

3.4.2 Deep Sets Limitations

This approach is much simpler to implement, compared to the RNN-based Pointer Networks and the RPW model. However, it generally prevents the model from learning pair-wise and higher order interactions between the elements of the set, which are lost during the summation. Additionally, significant doubts have been raised by Wagstaff et al. (2019) relating to the limits of the representational power of the DeepSets method. More precisely, the $O(n)$ computational complexity comes at the cost of the dimensionality of the latent space having to be at least equal to the cardinality of the input set n to ensure universal function approximation.

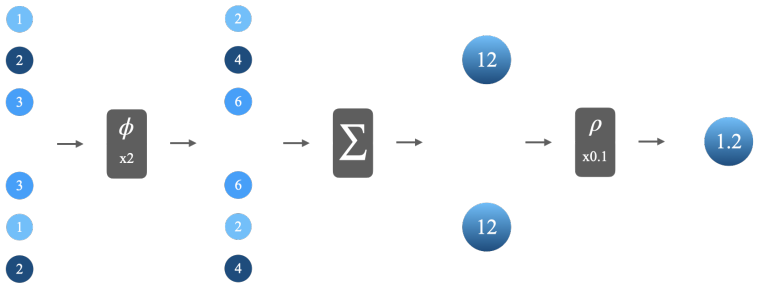


Figure 4: DeepSets

Every element of the two identical, shuffled sets of blue dots (leftmost) is embedded in an independent and identical way by the ϕ layer, resulting in a permutation equivariant transformation. These are then summed (Σ) into a permutation invariant representation and further transformed by the ρ layer.

3.4.3 Deep Sets Applications

The DeepSets framework has been applied to point cloud classification (Qi et al., 2017), generalization tasks in reinforcement learning (Karch et al., 2020), outlier detection and anomaly classification (Oladosu et al., 2020) among others.

3.5 Set Transformer

One of the most elaborate methods designed for set-input problems is the *Set Transformer* (Lee et al., 2019). This method can be seen as an extension of the popular feed-forward, attention-based Transformer (Vaswani et al., 2017) to the domain of machine learning on sets.

The Set Transformer consists of the expected stacked multi-head self-attention layers for both the internal encoder and decoder as seen in the classic Transformer. One aspect that separates it from the previously described set-to-sequence methods is that instead of using a fixed pooling operation such as summing or taking the average to ensure permutation invariance, it employs a parameterized pooling function that is learned and therefore much more adaptive to the particular task at hand. This is further referred to as *Pooling by Multihead Attention* (PMA) and explained in more detail later in this section.

The Set Transformer is specifically designed to model higher-order interactions among elements and their subsets within the input set, whilst satisfying the permutation invariance and variable input size requirements common to set-to-sequence problems. Its key novel contribution is that it concurrently encodes the entire input set through a sequence of permutation equivariant *Set Attention Blocks* (SABs). By comparison, the previously discussed DeepSets method obtained element features independently of other input set elements. This modification allows the Set Transformer to explicitly learn pairwise and even more complex interactions between set elements during the encoding step, dependent on the number of stacked SAB layers.

3.5.1 Set Transformer Limitations

However, the Set Transformer also introduces certain costs. The SAB, a proposed variation on the multihead attention block employed in the classic Transformer, which enables the Set Transformer to encode higher-order interactions between set elements, has the limiting quality of requiring quadratic time complexity $O(n^2)$ relative to the cardinality of the input set n.

The authors of the method address this limitation by proposing an *Induced Set Attention Block* (ISAB). It takes advantage of a vector of *inducing points*, which is of predetermined size and is used to obtain a hidden representation of the input set by attending to it. This is in effect a low-rank projection that might be familiar to readers who have experience with autoencoder models. The technique reduces the required computation time to a linear $O(mn)$, where m is the chosen number of inducing points and n is the set cardinality, at the cost of reduced performance and an additional hyperparameter to be tuned.

3.5.2 Set Transformer Details

What follows is a more detailed inspection of the SAB, ISAB and PMA methods, which assumes some familiarity with the multihead attention mechanism proposed by (Vaswani et al., 2017) as per the Transformer architecture. However, to establish full clarity, we will formally define both the basic Transformer attention and its multihead variant first.

Assume we have a set of n elements as our input, each with dimensions d_e. First, we obtain three new vectors per each element by multiplying it with three matrices, whose weights are learned during the training step. These transformed input representations are further referred to as n query vectors $Q \in \mathbb{R}^{n \times d_q}$, key vectors ($K \in \mathbb{R}^{n \times d_k}$) and value vectors ($V \in \mathbb{R}^{n \times d_v}$). These are then mapped to the desired attention outputs, applying an activation function such as softmax in the following way:

$$\text{TransformerAttention}(Q, K, V) = \text{softmax}(QK^\top)V \tag{11}$$

The original implementation includes a scaling factor, which has been omitted from the above equation for the sake of simplicity. The pairwise dot product of the query Q and key K vectors measures how related each pair is. The final output is a weighted sum of V.

The multihead attention mechanism extends this further by projecting each query, key and value onto h separate vectors via sets of the three parameter matrices W_i^Q, W_i^K, W_i^V, one per each of the h heads. Then, the TransformerAttention() function is applied to each of these h vectors to obtain each head's preliminary output O_i:

$$O_i = \text{TransformerAttention}(QW_i^Q, KW_i^K, VW_i^V) \tag{12}$$

Finally, these outputs $\{O_i\}_{i=1}^h$ are concatenated and then linearly transformed:

$$\text{MultiheadAttention}(Q, K, V) = \text{concatenate}(O_1, \ldots, O_h)W^0 \tag{13}$$

Now we can begin to move on to the Set Attention Block (SAB). It is designed to take a set and perform a slightly modified self-attention operation between its individual elements, which results in an output set of the same size. It will be useful to first define a *Multihead Attention Block* (MAB), which is an intermediate building block in both SAB and its less computationally intensive alternative - ISAB. MAB takes as input two matrices of the same dimensions: $A, B \in \mathbb{R}^{n \times d}$ and first obtains a hidden representation Z using a layer normalization operation as defined by Ba et al. (2016):

$$Z = \text{LayerNormalization}(A + \text{MultiheadAttention}(A, B, B)) \tag{14}$$

Then, it processes each element in each row of Z in an independent, identical way, in the same manner we have seen as part of the DeepSets method, prior to the summation. This can be done via a row-wise feed forward layer $\phi()$:

$$\text{MAB}(A, B) = \text{LayerNormalization}(Z + \phi(Z)) \tag{15}$$

However, in our set-to-sequence setting we do not have two separate matrices of sets A and B that we wish to encode into some joint representation. Therefore the actual SAB() is defined on a single matrix of an input set $X \in \mathbb{R}^{n \times d}$:

$$\text{SAB}(X) = \text{MAB}(X, X) \tag{16}$$

If we stack l SAB layers in our set encoder, which we are able to do since both the input and output of the Set Attention Block is a set of the same size, the model's computational complexity in the cardinality of this set is $O(n^2 l) \approx O(n^2)$. At the cost of quadratic computation time, stacking 2 SAB layers enables the model to encode pairwise interactions between elements. Stacking more such layers makes explicitly encoding higher-order interactions possible, which is a crucial novel contribution for tasks where such interactions define the target output.

The authors of the Set Transformer method address the computational cost of SAB by proposing a less expensive variation of it, called the Induced Set Attention Block. In ISAB, an additional array of *inducing points* $I \in \mathbb{R}^{m \times d}$ is included. This vector is of predefined dimension m, resulting in a computational complexity $O(mn)$ or $O(lmn)$, if stacked ISAB layers are applied. The calculations within ISAB are defined as follows:

$$\text{ISAB}_m(X) = \text{MAB}(X, \text{MAB}(I, X)) \tag{17}$$

The learned values of the inducing points I are expected to encode large-scale aspects of the input set as meaningful features for the ultimate task. Similar to SAB, the output of ISAB is a set of the same size as the input set, which is why we still need a pooling operation to be applied to it at this point.

The final aspect of the Set Transformer that distinguishes it from the earlier set-encoding methods is the Pooling by Multihead Attention (PMA) stage. Unlike a simple sum, mean or max, the PMA pooling function has learnable parameters, which allows it to increase or decrease the relative importance given to the encoding of individual encoded elements of the output of the SAB and ISAB blocks. Usage of PMA requires specifying the number k of seed vectors $S \in \mathbb{R}^{k \times d}$. Assuming we have already obtained the encoded set features $E \in \mathbb{R}^{n \times d}$ via stacked SAB or ISAB layers:

$$\text{PMA}_k(E) = \text{MAB}(S, \phi(E)) \tag{18}$$

In most cases a single ($k = 1$) seed vector is used, resulting in a single pooled set encoding, but certain clustering tasks may require multiple related outputs, justifying the use of a larger k.

3.5.3 Set Transformer Applications

The overall Set Transformer architecture can, in principle, be applied to any set-input problem (and therefore any set-to-sequence task). It will perform particularly well in problems where pairwise and higher-order interactions between the input set's elements are important to the task at hand. The authors of the original paper demonstrate the model's usefulness within such areas on the challenge of counting unique characters in a set of input images, amortized clustering with mixture of Gaussians, point cloud classification and set anomaly detection.

It has since been applied to set-of-sets embedding problems (Meng et al., 2019) and transfer learning in dialogue systems (Wolf et al., 2019).

3.6 Other Set-Input Methods

Below is a list of other set encoding methods that do not necessarily lend themselves directly to set-to-sequence problems, but may be of interest to the reader. The specifics of using the learned, permutation invariant set representation to produce a sequence of set elements are discussed at the beginning of the next chapter, specifically in subsection 4.1.

3.6.1 Featurewise Sort Pooling

This method, also known by its abbreviation as FSPool (Zhang et al., 2020), came from the field of set prediction, in relation to a problem where both the input and output can be conceived of as sets. The authors expand upon one of the naive approaches to encoding sets in a permutation invariant way. Namely, the technique of simply sorting all the elements of the input set by the values of their single chosen feature, as seen in previous work by Zhang et al. (2018).

In set-output problems this approach results in discontinuities in the optimisation whenever two elements swap positions after the sort. This is referred to in set prediction challenges as the *responsibility problem* (Zhang et al., 2020). To avoid this difficulty, the authors have developed a novel pooling method which sorts each feature across the elements of the input set and then performs a weighted sum.

This allows the model to remember the permutation applied through the featurewise sorting and apply its inverse in the decoder. This process restores the original, arbitrary order of the input elements making the encoding a permutation equivariant operation, preventing the discontinuity in the outputs of the model.

3.6.2 Janossy Pooling

Another interesting approach to set-encoding through the use of simpler pooling operators was proposed by Murphy et al. (2019). In the titular *Janossy Pooling*, the symmetric (permutation invariant) encoding function is expressed as the average of a mixture of permutation sensitive functions applied to all reorderings of the original input.

This approach immediately raises the question of tractability. Generating all permutations of a set results in $n!$ intermediate inputs, all of which would then require the application of the chosen permutation sensitive function. To mitigate this, the authors propose a number of strategies, among them the use of a smaller number of selected canonical orderings that are presumed to carry relevant information for the specific task at hand, such as simple sorting, *betweenness centrality* and others (Niepert et al., 2016).

As an alternative to canonical orderings, the authors also propose a method related to a model's ability to explicitly learn pairwise and higher-order interactions between the elements of the input set. This method is referred to as k-ary dependencies. It consists of projecting the input to a length k sequence, for example by only keeping the first k elements, limiting the number of permutations that need to be averaged to $k!$, which can be tractable for a small enough k. The number k becomes a hyperparameter capable of balancing tractability with the model's ability to learn k-ary interactions in the input. Finally, the authors also experiment with permutation sampling as the third method of reducing the computational complexity of Janossy Pooling, as proposed by Moore and Neville (2017) and Hamilton et al. (2017) in relation to machine learning on graphs.

3.6.3 PointNet

Not to be confused with the Pointer Network described in section 3.2, *PointNet* is a set-encoding method designed to handle 3D point clouds proposed by Qi et al. (2017). The geometric setting of the problem tackled by the authors of this model shares many similarities with the set-to-sequence domain. These include the lack of order in the input, requiring the use of symmetric, permutation invariant encoding functions, and the importance of interactions between individual input elements or, specifically in this case, points. An additional requirement for this setting is invariance under certain geometric transformations of the entire point cloud. For example, if we are tasked with classifying 3D objects represented by a point cloud, we want to correctly classify a chair regardless of its rotation and translation.

In practice, PointNet first obtains an embedding of each of the input points through stacked, fully-connected layers in the form of a multilayer perceptron, such that each element is identically

and independently transformed. This permutation equivariant representation is then pooled via the max() operator (per dimension) and further transformed through an additional fully-connected layer. This is in effect a slight variation of the procedure proposed by Zaheer et al. (2017) and discussed in section 3.4. Other pooling operations, including an attention-based sum inspired by the RPW model (Vinyals et al., 2016), are also experimented with in the original paper..

Finally, the obtained point cloud encoding is concatenated with the embedding of each point, reminiscent of the approach seen in listwise ranking, described in section 4.4. This combination of local and global features is shown to be crucial for point segmentation tasks. The authors also provide proof that their network is a universal approximator for continuous set functions and demonstrate its robustness to small perturbations of the input set.

3.6.4 AttSets

The *AttSets* model, proposed by Yang et al. (2020), uses weighted attention to obtain a permutation invariant representation of the input set. It was originally applied to a multi-view 3D reconstruction task, where a set of images of the same object from different angles is used to estimate its true three dimensional shape.

AttSets improves the performance of previous, simpler pooling functions used for 3D object recognition. These include both first-order operators such as max(), average() and sum(), which do not have any trainable parameters, as well as higher-order statistical functions such as bilinear pooling (Lin et al., 2018), log-covariance pooling (Ionescu et al., 2015) and harmonized bilinear pooling (Yu et al., 2018), which have only few.

In order to achieve this, each element of the set is individually and independently transformed via a learned attention function, which can take the form of a fully connected layer or a multidimensional convolutional layer, depending on the form of the input. The output of this function is normalized via softmax() and then used as an attention mask over the original input elements. This allows the model to learn to pay a varying degree of attention to individual dimensions of the input elements' representations. To obtain the final, fixed-length set encoding, the original input elements are multiplied by the attention mask and summed together.

3.6.5 RepSet

An interesting set-encoding method, referred to as *RepSet*, has been proposed by Skianis et al. (2020). The RepSet model consists of stacked feed-forward, fully connected layers, reminiscent of the DeepSets method (Zaheer et al., 2017), followed by a custom permutation invariant layer replacing the sum() operator. This layer is inspired by concepts from the field of bipartite graph matching and has allowed the model to show promising performance on text and graph classification tasks.

The permutation invariance is achieved through a configurable number of hidden sets (potentially of different cardinalities), whose elements correspond to columns of trainable weight matrices. These are then compared with the elements of the actual input set to create matrices that are fed to a bipartite matching algorithm, specifically the Hungarian Algorithm (Grinman, 2015). The resulting values can be further transformed through standard neural network layers for set classification and regression purposes.

A significant issue with this approach is the computational complexity $O(mn + n^2 \log n)$, where n is the cardinality of the input set and m is the chosen number of hidden sets. This characteristic, limiting the usefulness of the method regarding larger sets, stems from the bipartite matching algorithm needed to obtain the final set encoding. The authors of the method address this by proposing a relaxation of RepSet, referred to as *ApproxRepSet* (Skianis et al., 2020), which removes one of the constraints on the range of values taken by the elements of the hidden sets.

4. Ordering Methods

This section focuses on the second of the two primary challenges inherent to set-to-sequence tasks, which is outputting a permutation. Assuming we are able to obtain a meaningful representation of an input set of any length, how do we use that representation to produce a reordering of the input's original elements? This is a constraint that is easy to satisfy when designing traditional combinatorial optimization algorithms, yet in deep learning it requires relatively complex model architectures (Bengio et al., 2020).

Three particular branches of ordering methods have emerged as most prominent in deep learning:

1. *Attention-based Pointing*

   In which a vector of attention weights over all elements of the input set is generated iteratively at every index of the output sequence. The highest attention value points to the element that should be placed in the current position within the sequence.

2. *Permutation Matrices*

   Where a square, binary, doubly-stochastic matrix or a relaxation thereof is generated for each input set. The index of the highest value in each row identifies the element that should take the position at the same index as the number of the row. The input can be left-multiplied by this matrix to obtain the final reordering.

3. *Listwise Ranking*

   In ranking methods the target order is represented through the assignment of a score to each element of the input set, which enables the final permutation to be obtained through sorting. Listwise ranking takes into consideration the relative scores of all other set elements when computing the score for a particular one.

Each of these three basic frameworks is described in the following sections, with references to specific methods that expand upon them, where relevant. First, however, we must discuss how best to utilize the permutation invariant set representation, obtained via the set encoding methods discussed in section 3, to output the target sequence.

4.1 From Set Representation to a Sequence

As stated previously, deep learning models do not directly take unordered sets as input. Instead they transform ordered arrays representing one (usually arbitrary) of $n!$ permutations of a given set's elements. Therefore, an inductive bias is introduced into the model's internal architecture to first obtain a permutation invariant representation of the underlying input set, which by definition will be the same regardless of which arbitrary permutation the model happened to receive.

However, when our prediction target is the optimal order of a given set's elements and we are directly feeding our model an array in some arbitrary order, then our target output sequence must refer to this initial, random order when predicting the preferred one. In effect, in the case of simple permuted sequences without repetition and exclusion, we are seeking a permutation equivariant function that outputs ranks, conditioned on a permutation invariant representation of the entire set. In the case of complex permutations, we also expect the same sequence to be predicted based on the same input set.

Regardless, the output sequence has to refer to the arbitrary order in which the input set elements are presented to it. In essence, whilst we assume there is one optimal order of a given set's elements, there are $n!$ target permutations relative to the $n!$ arbitrary ones that form the actual input of the model, per input set. This is because each target permutation reorders an arbitrary one that the model receives as actual input, each representing the same, singular optimal order of the underlying set's elements.

The remaining question is how to utilize the fixed-length, permutation invariant set representation when outputting the sequence representing a permutation of the arbitrarily ordered input's elements. There are two dominant approaches, namely summation and concatenation of the representation of the entire set with the learned representations of each element. In this way, every transformation of individual set elements has the chance to take into consideration the entire available set, as previously seen in the PointNet model by Qi et al. (2017).

Considerations regarding which of those two methods is preferable relate to the larger field of representation learning, with ample examples both on the side of summation (Szegedy et al., 2016) and concatenation (Noreen et al., 2020). These include the *ResNet* architecture by He et al. (2016), with addition used in the eponymous residual connections, and the *DenseNet* model and its descendants, employing concatenation (Huang et al., 2018a), within the field of computer vision.

An alternative approach, applicable to RNN-based ordering methods, would be to utilize the set representation in the hidden state of the recurrent network. Similarly, attention-based methods could use the set representation to influence which elements the model focuses on. Finally, in ranking approaches the set representation can rather elegantly be used in place of the *query* (see section 4.4 for more details). Investigation of the efficacy of these methods is a possible direction for future research, as outlined in section 5.6.

4.2 Attention-Based Pointing

The term *attention* covers a wide spectrum of methods within the field of machine learning. First introduced as part of an encoder-decoder model applied to a sequence-to-sequence translation task by Bahdanau et al. (2015), it has since been utilized in a number of other domains, including computer vision (Xu et al., 2015), graph-based problems (Veličković et al., 2018), reinforcement learning (Iqbal & Sha, 2019) and many more. For a comprehensive overview, see Chaudhari et al. (2019).

In structured output tasks that are inherent to the set-to-sequence domain, the key idea is to induce a vector of attention weights over the entire input, allowing for the selection of elements by their position, conditioned on the learned representation of the entire input and the sequentially predicted outputs. An example of such a setting from the field of natural language processing (NLP) is *linearization*, which involves solving a problem that will be familiar to most readers who have tried to learn a new language. Namely, given a randomly shuffled set of words, we are tasked with ordering them into a properly structured sentence. A survey of these and other attention-based methods in NLP can be found in the work of Hu (2019).

In effect, the model can learn to identify the current most pertinent position in the input through the attention weights, which we then interpret as pointers to the next preferred element for the output sequence. It may be worth mentioning that valid concerns have been raised as to the overall interpretability of attention weights in text classification and question answering tasks by Jain and Wallace (2019). However, in attention-based pointing these weights have a direct impact on the predicted permutation, leaving little room for interpretational ambiguity.

4.2.1 Details of Attention-Based Pointing

Attention-based pointing is an adaptation of content-based attention (Graves et al., 2014) to an ordering challenge. The model learns a distribution over all input elements at each position of the output sequence. This should not be confused with self-attention, also known as intra-attention (Cheng et al., 2016), where the target sequence is always the same as the input. In self-attention, the parameterized attention function learns to relate different positions of the input sequence to each other. In attention-based pointing the attention function learns to relate the elements of the input with the current positions of the output.

Assume we have a randomly ordered input sequence of varying length n, consisting of d-dimensional elements: $X = \langle \mathbf{x_1}, \ldots, \mathbf{x_n} \rangle$ such that $X \in \mathbb{R}^{n \times d}$, which we want to order according to some pref-

erence. This sequence is an n-tuple, representing a single permutation of the corresponding set $X' = \{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$, which we cannot represent directly as a single vector, due to the fact that vectors posses inherent order whereas sets do not.

Our target output is a sequence of integer pointers $Y = \langle y_1, \ldots, y_n : y_i \in \mathbb{Z}^+ \rangle$,, satisfying the conditions: $\forall i, j : y_i \neq y_j$ and $\forall i, 1 \leq y_i \leq n$, signifying that no element from the input can take multiple positions in the output sequence and every element must be assigned a position in the final reordering. Our objective is to define a differentiable function $f : X \to Y$ that meets these criteria, with some parameters θ, lending itself to gradient-based training.

The simplest effective formulation of such an attention-based ordering function requires the following learnable parameters: two square weights matrices $W_1, W_2 \in \mathbb{R}^{d \times d}$ as well as a single vector $\mathbf{v} \in \mathbb{R}^{d \times 1}$. Additionally, given the sequential nature in which the ordering is generated, we also have access to a decoder state $\mathbf{s_t}$ at each step t, for which we calculate the attention-based probability distribution over input elements. The array of encoded set elements X is processed in the following way:

$$z_t^i = \mathbf{x_i} W_1 + \mathbf{s_t} W_2 \qquad \text{for } i \in (1, \ldots, n) \tag{19}$$

$$a_t^i = \frac{\exp\left(\mathbf{v}^T \sigma\left(z_t^i\right)\right)}{\sum_j^n \exp\left(\mathbf{v}^T \sigma\left(z_t^j\right)\right)} \qquad \text{for } i \in (1, \ldots, n) \tag{20}$$

$$y_t = \text{argmax}(\mathbf{a_t}) \qquad \text{for } t \in (1, \ldots, n) \tag{21}$$

where σ is a nonlinear activation function, commonly the hyperbolic tangent (Logeswaran et al., 2018). The vector $\mathbf{z_t} = \langle z_t^i, \ldots, z_t^n \rangle$ is a representation of all input elements, adjusted by the representation of the generated output sequence at step t, through additive attention (Bahdanau et al., 2015). Finally, y_t is an integer pointer to the element of X which received the highest attention value within the attention vector $\mathbf{a_t} = \langle a_t^i, \ldots, a_n^i \rangle$ at the current step t of the sequentially predicted output sequence.

Most methods obtain the embedding of the entire input X through more complex encoding methods, such as stacked bidirectional LSTMs, as seen in the work of Vinyals et al. (2015) and Vinyals et al. (2016). This representation retains the information about the original arbitrary order of the elements and should be dependent on the entire set. In the above example, the length t of the output sequence of pointers Y is the same as the cardinality of X', which is the case in many combinatorial optimization problems. However, in some structured prediction tasks the optimal length of the output is also subject to the learning process, allowing for pointing to the same element $\mathbf{x_i}$ multiple times or not at all, as is the case in catalog design (Carlson-Skalak et al., 1998).

It should be noted that in the above formulation nothing is explicitly preventing the model from pointing to the same element $\mathbf{x_i}$ at multiple positions of the output sequence. In practice the iterative learning process can largely prevent this from occurring, given appropriate training data. One method to explicitly prohibit repetition is progressive masking, as described in section 3.2.1.

In practice, beam search is commonly used during inference to increase the probability that the most optimal sequence is predicted, as evidenced in the work of Bahdanau et al. (2017) and Kool et al. (2019). Beam search employs a heuristic search algorithm to expand a limited number of most promising vertices of the graph defined by the attention vectors over a predefined number of output steps. It tracks a small number of potential partial solutions, at the controlled cost of memory and computation. However, the exact reason why a small beam number results in qualitatively better predictions has recently been called into question (Meister et al., 2020). For a more detailed overview in the context of sequence prediction, see Wiseman and Rush (2016).

4.2.2 Methods Using Attention-Based Pointing

Many neural network model architectures across distant fields of application include variations on the basic attention-based pointing mechanism. These include specifically set-to-sequence methods, such as the Pointer Networks discussed in section 3.2.2 and the RPW model detailed in section 3.3.2, as well as elements of complex models in reinforcement learning applied to competitive real-time strategy challenges (Vinyals et al., 2019).

Additionally, attention-based pointing found usage in identifying entailment between documents (Rocktäschel et al., 2016), abstractive text summarization (See et al., 2017), rare and out-of-vocabulary word prediction (Merity et al., 2016) as well as describing multimedia content (Cho et al., 2015). An interesting augmentation of attention-based pointing has been experimented with in the context of generating structured queries (in SQL) from natural language sequences (Zhong et al., 2018). However, the improvement stems from augmenting the input with SQL keywords to limit the output space, not from an essential adjustment to the underlying attention-based pointing mechanism.

An application of attention-based pointing to generate solutions to another classic combinatorial optimization challenge, the Vehicle Routing Problem, was proposed by Kool et al. (2019). The resulting architecture is an encoder-decoder *Graph Attention Network* (Veličković et al., 2018), employing multiheaded attention in the encoder and node masking in the decoder, which uses the embeddings of all the nodes and the entire graph at each step t to point to the next node to be visited. Unlike the previously discussed methods, this model is trained using a gradient estimator from the field of reinforcement learning, first proposed by Williams (1992).

4.3 Permutation Matrices

A permutation matrix is a square, binary matrix P having exactly one entry $p^i_j = 1$ in each row i and each column j (Stuart & Weaver, 1991). All other entries are equal to 0. P's dimensions are defined by the length of the desired output sequence, most commonly equal to the length of the input. A permutation matrix is unimodal in that each of its rows has a unique, highest value. It is also doubly-stochastic, since it consists entirely of nonnegative numbers, with each row and column summing to 1.

The key property of a permutation matrix is that given an arbitrarily ordered sequence, we can left-multiply it by a permutation matrix to obtain a reordered sequence. For example, given an arbitrarily shuffled input sequence $\mathbf{x} = \langle x_1, x_5, x_3, x_4, x_2 \rangle$, which we want to permute in such a way as to restore the $i-$indexed ascending order to obtain $\mathbf{x}^\pi = \langle x_1, x_2, x_3, x_4, x_5 \rangle$, we can predict the permutation matrix visualized below. Our target is the permutation π, given in two-line notation:

$$\pi = \begin{pmatrix} x_1 & x_5 & x_3 & x_4 & x_2 \\ 1 & 5 & 3 & 4 & 2 \end{pmatrix} \tag{22}$$

We can apply this permutation to the transposed input $\mathbf{x}^\top$ via left-multiplication by the target permutation matrix P:

$$\mathbf{x}^\pi = P\mathbf{x}^\top = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_5 \\ x_3 \\ x_4 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \tag{23}$$

This gives us a way to represent the target permutation in the form of a matrix of numbers between 0 and 1, marking an important step towards enabling the application of machine learning methods.

4.3.1 Making Permutation Matrices Learnable

In order to use a permutation matrix as the target output of a machine learning model that can be trained through some form of gradient-based iterative optimization, we must introduce certain relaxations of the concept. Otherwise, we are left with the equivalent of a sorting operator which is non-differentiable (Grover et al., 2019).

Traditionally, sorting operations result in either a permutation $\pi = \langle 1, 5, 3, 4, 2 \rangle$ (from the previous example) or a vector of reordered elements $\mathbf{x}^\pi = \langle x_1, x_2, x_3, x_4, x_5 \rangle$. Much like the permutation matrix, the former is non-differentiable with respect to the input due to being integer-valued and the latter due to being piecewise linear (Cuturi et al., 2019).

The most direct way to obtain a differentiable relaxation P' of the permutation matrix P is to map the input $\mathbf{x}$ to a continuous codomain, as opposed to the original discrete one. Thus, we need a relaxation such that $P' \in \mathbb{R}^{n \times n}$, lending itself to gradient-based optimization. Additionally, an efficient projection from the continuous codomain back to the discrete one must exist to allow for the use of loss functions and evaluation metrics. This can be achieved by applying the argmax() function per row of P' to find the position in which the unique 1 would have been located in the actual permutation matrix P.

The resulting relaxation must retain the property of row-stochasticity, such that:

$$\forall i, j \in \{1, \dots, n\} : p'_{ij} \geq 0 \tag{24}$$

$$\forall i \in \{1, \dots, n\} : \sum_{j=1}^{n} p'_{ij} = 1 \tag{25}$$

and of unimodality, such that a vector $\mathbf{y}$ is both obtainable from the relaxation matrix P' through row-wise argmax() and a valid permutation of the input:

$$\forall i \in \{1, \dots, n\} : y_i = \text{argmax}(\mathbf{p}'_i) \tag{26}$$

There are many possible methods of obtaining this relaxation from the input after it has been transformed by a chosen neural network architecture, such as adding elementwise Gumbel perturbations (Mena et al., 2018), applying the Sinkhorn operator to directly sample matrices near the Birkhoff polytope (Linderman et al., 2018), which is the convex hull whose points are doubly-stochastic matrices (Emami & Ranka, 2018), or through the application of a softmax() operator on a derived matrix of absolute pairwise distances between the individual input elements (Grover et al., 2019).

Depending on the specifics of the task at hand, the target matrix can be predicted in a single pass (if the length of the input and output is a constant, known prior to inference) or sequentially, row by row (Nishida & Nakayama, 2017). An interesting method employing permutation matrices has been proposed by Zhang et al. (2019), in which a trainable, pairwise *ordering cost* function is used to produce an anti-symmetric matrix C, whose entry c_{ij} represents the cost of placing the i-th element before the j-th. This function is represented as a neural network, which is then used to continuously adjust the learned permutation matrix. This is referred to as a *Permutation-Optimisation* module, and has been demonstrated to perform well on number sorting, re-assembling image mosaics and visual question answering, with one limiting feature of entailing cubic time complexity.

4.4 Listwise Ranking

Ranking methods determine the predicted order of elements by assigning a score to each element and then sorting them according to these scores. In listwise ranking, a score is calculated based on a list of available elements and a *query*, for which a specific order is to be predicted. The terminology stems from applications in information retrieval, where the task is to rank the available *documents*

(such as web pages) in order of relevance for a given query (e.g. a search term). For an overview of neural ranking methods, see Mitra et al. (2018).

Listwise ranking is distinguished from point- and pair-wise ranking. Pointwise methods reduce the ranking problem to regression, in that the relevance score for an element is obtained only from its own representation and the query. Pairwise approaches reduce it to binary classification (Lei et al., 2017). Given a query and a pair of elements, they predict which element is the more relevant of the two. In listwise ranking, the prediction is performed on a list of objects.

Application of listwise ranking methods to set-to-sequence problems requires certain adjustments. Most importantly, in such ordering and structure prediction challenges we are not given a specific query, for which an ordering is to be generated. However, we are able to obtain a permutation invariant set representation through the set encoding methods detailed in section 3. We can use this learned embedding to fill the role that a query performs in traditional learn-to-rank methods. Intuitively, the relative rank of each element in the output sequence should depend on the entire available set.

This marks an important departure from the assumption that two elements have a canonical relative order, which should remain unchanged regardless of what other elements are present in the input set. In practice, higher-order interactions between available elements can entirely change the target sequence. In order to learn such properties, a ranking method can utilize the permutation invariant set representation as the query and predict the relevance scores of all available elements in one go, in the listwise manner described in the following subsection.

4.4.1 Details of Listwise Ranking

In order to illustrate the basic underlying mechanisms in listwise ranking methods, this subsection investigates the first method that formulated order prediction on a list of objects, namely the *ListNet* model, proposed by (Cao et al., 2007). From this point on, whenever we refer to a query, we are referring to the permutation invariant representation of the entire set obtained via the previously outlined set encoding methods. This representation can be learned in parallel with the weights of the ranking the model.

Once more, assume we are given an arbitrarily ordered input sequence $X = \langle \mathbf{x_1}, \ldots, \mathbf{x_n} \rangle$ such that $X \in \mathbb{R}^{n \times d}$. This sequence is one of $n!$ possible permutations of the corresponding set $X' = \{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$. The objective is to transform the given X vector into the best possible permutation of X', according to some preference. This target permutation is $\pi = \langle \pi(1), \ldots, \pi(n) \rangle$, such that $\pi(i)$ is the object at position i in the permutation. The set of all possible permutations of length n is denoted as Ω_n. The target permutation π is represented by a vector of scores $\mathbf{y}^\pi = \langle y_{\pi(i)}, \ldots, y_{\pi(n)} \rangle$, where $y_{\pi(i)}$ is the predicted relevance score for the element $\mathbf{x_i}$. All elements of the original input X get assigned a score by the learned neural network, in a listwise manner.

A naive approach would be to try to obtain the probability $P_y(\pi)$ of each of the $n!$ possible permutations from Ω_n, given a set of scores $\mathbf{y}$:

$$P_y(\pi) = \prod_{i=1}^{n} \frac{\phi(y_{\pi(i)})}{\sum_{k=i}^{n} \phi(y_{\pi(k)})} \tag{27}$$

where $\phi()$ is any increasing and strictly positive function applied to the scores and $y_{\pi(i)}$ is the relevance score of element i in the vector $\mathbf{y}_\pi$ corresponding to some permutation π (Cao et al., 2007).

However, this is a computationally inefficient approach, therefore we instead calculate the *top one probability* for each element $\mathbf{x_i}$. This probability is equal to the sum of the permutation probabilities of every permutation where the i-th element was ranked first. For the exact proof of this equivalence, see the appendix in Cao et al. (2007). The crucial observation is that we do not need to calculate all the permutation probabilities to obtain the correct top one probability $P_{top}(\mathbf{x_i})$ of each element, given a list of scores $\mathbf{y}_\pi$:

$$P_{top}\left(\mathbf{x_i}\right) = \frac{\exp\left(y_{\pi(i)}\right)}{\sum_{k=1}^{n}\exp\left(y_{\pi(k)}\right)} \tag{28}$$

In effect, the probability distribution over permutations is obtained by applying the softmax() function to the predicted relevance scores. Since this is a supervised learning framework, our model has access to the ground truth distribution for each example list. This allows for the use of loss functions that compare two distributions, such as the Kullback-Leibler (KL) divergence, also referred to as relative entropy (Liu & Shum, 2003).

In order to obtain the vector of relevance scores $\mathbf{y}_\pi$, the query vector $\mathbf{q} \in \mathbb{R}^m$ is concatenated with a fixed-length, learned embedding of each element of the input set $\mathbf{x_i}$, resulting in the final input $I \in \mathbb{R}^{n\times(m+d)}$, per single example list. In the original ListNet model, this input is transformed through stacked, fully-connected layers, of which the last one has a single unit outputting the relevance score per element.

As such, this transforms each concatenation of the query and the embedded element in an independent and identical manner. This is practical for ranking tasks, where two items always have the same relative target order, given a query. However, in more complex structure prediction tasks, common in set-to-sequence challenges, relying entirely on the set encoding $\mathbf{q}$ and the listwise loss to identify interactions between set elements limits the ability to approximate complex functions, in which higher-order interactions between elements have an impact on the final order (Lan et al., 2009). This limitations of ListNet is partially addressed by later methods, discussed in section 4.4.2.

Additionally, a limitation of the ranking approach is that by representing the target output sequence through relevance scores per input element, we cannot learn to output sequences of any length other than n. This also precludes the use of listwise ranking in challenges where the output sequence is of the same length as the number of input elements n, but repetition and exclusion of elements is required. For a more detailed look into the underlying theoretical aspects of listwise ranking, see Xia et al. (2008).

4.4.2 Methods Using Listwise Ranking

Multiple listwise ranking methods in machine learning have been developed since the original ListNet model. A notable framework that addresses the challenge of learning pairwise interactions between elements came in the form of *BoltzRank*, in which the rank probabilities are sampled from a Boltzmann distribution, employing an energy function that depends on a score influenced by both individual and pairwise potentials (Volkovs & Zemel, 2009).

Another approach of interest is the *FATE* framework proposed by Pfannschmidt et al. (2018). The authors identify the problem of predicting a relevance score for an element and the query with only the loss function carrying the signal regarding the context of other available elements. In order to address this, they effectively employ a permutation invariant set-encoder (Zaheer et al., 2017), whose output is concatenated to the learned representation of each element, in a variation of the basic method described in section 4.4.1.

Ai et al. (2018) propose a complex architecture combining two-stage ranking, sequential recurrent neural networks and an attention-based loss function. The proposed *Deep Listwise Context Model* (DLCM) sequentially encodes the most relevant results using the corresponding feature vectors, trains an additional, local context model and employs it to re-rank the best k results.

Finally, Pang et al. (2020) employ the Set Transformer to obtain element representations that encode cross-document interactions and return a permutation invariant ranking by sorting the permutation equivariant relevance scores per each document.

5. Discussion

In this section, the key challenges and contexts of application for set-to-sequence deep learning methods are discussed. Additionally, separate sections are devoted to the progress in set encoding and permutation learning, along with current limitations and proposed directions for future research.

5.1 Key Challenges

Core challenges of set-to-sequence include representing sets of varying sizes in a permutation invariant way, encoding pairwise and higher-order interactions between set elements and keeping the computational complexity moderate in the cardinality of the input set. This last requirement stems from the fact that in many important set-to-sequence problems the size of the input set can be quite large, particularly in computer vision point-cloud based tasks (Ge et al., 2018). This has been addressed by various proposed set-encoding or set-pooling methods discussed in section 3, such as RNN-based Pointer Networks (Vinyals et al., 2015) and Read-Process-and-Write models (Vinyals et al., 2016) as well as through fully connected methods such as DeepSets (Zaheer et al., 2017) and the Set Transformer (Lee et al., 2019), among others.

Another set of key challenges in set-to-sequence stems from the many possible ways of formalizing the process of outputting a permutation sequence. Potential methods include learn-to-rank approaches, permutation matrices and attention-based methods discussed in section 4. Additionally, it is not always trivial to combine the output of the specific set-encoding technique with the expected input of these permutation methods in a way that results in good model performance. Finally, we expect not just the set-encoding module but the entire set-to-sequence model to be permutation invariant and always give the optimal output regardless of how the array representing the set is reordered at input.

5.2 Contexts of Application

There are many contexts where the input data does not have an inherent ordering and the number of input elements varies (i.e. *set-input* problems) and possibly even more where the input elements are not unique, instead repeating a meaningful number of times, in which case the task presents an *mset-* or *multiset-input* problem. Additionally, the set elements may be allowed to reoccur multiple times in the predicted sequence or be excluded from it entirely, forming a *complex permutation*. These elements can be both continuous (e.g. word embeddings) or discrete (coming from an index-based dictionary). An example of such a problem with a strong industrial application is the question of how to order available product offers into a displayable catalogue that will keep the reader engaged and eventually inspire them to make a relevant purchase (Liao & Chen, 2004).

Similar challenges are faced by the experts who order news articles into a coherent publication, book authors composing chapters into a novel and by engineers tackling the challenge of *catalog design*, in which a configuration is created by assembling off-the-shelf components into a functional system (Carlson-Skalak et al., 1998). Set-to-sequence methods can also benefit architects in their search for the best configurations of the design space, taking into consideration structural efficiency, daylight availability and other aspects of building performance (Brown & Mueller, 2017). Whenever the elements come from a pre-existing set and the output is structured as a complex permutation, we are facing a set-to-sequence challenge.

5.3 Progress in Set Encoding

As machine learning methods become more and more widely used, the range of input and output data structures that these methods are applied to becomes larger. Mirroring the development and consecutive growth in popularity of methods created specifically to handle certain types of input data, such as convolutional neural networks for images and recurrent models for sequences, one of the

most significant developments in set-to-sequence has been the work towards obtaining permutation invariant representations of sets of varying lengths.

Particularly, this representation can now be obtained in a way that is capable of explicitly encoding not just pairwise but also higher-order interactions between input elements. Intuitively these improvements mirror the way we as humans process input sets, in that we group the elements into meaningful collections, allowing for enough flexibility that this grouping may change entirely upon the introduction of a new element.

5.4 Progress in Permutation Learning

Comparative progress has been made regarding the ways in which we formalize the permutation tasks at hand. Given that the simplest form of permutation (reordering) is sorting, the early methods formalized this challenge as a ranking problem, with pointwise, pairwise and eventually listwise loss being used to train the model. These methods suffer from the assumption that any two elements have a canonical pairwise ranking, regardless of the features of the other elements in the input set.

Another alternative emerged in the form of self-attention applied in such a way as to output softmaxed pointers to the elements of the input set at each step in the output sequence, with the additional use of beam search during inference (Vinyals et al., 2015). The third dominant formalization comes in the form of learning to output a permutation matrix. This allows for the original input to then be matrix-multiplied by the row-stochastic permutation matrix, resulting in the desired reordering (Nishida & Nakayama, 2017), (Emami & Ranka, 2018). It is a sign of the complexity of the field that no clear preferred formalization of its core challenge has emerged, with ranking (for example) still finding useful application in active research (Kumar et al., 2020).

5.5 Limitations

A crucial limitation of many of the cutting edge set-to-sequence models, such as the Set Transformer (Lee et al., 2019) and the permutation mechanism in Pointer Networks, is their reliance on self-attention. Whilst Transformer-based methods that rely solely on multiheaded self-attention have seen remarkable success, even in applications beyond a fixed-length context (Dai et al., 2019), their ability to process hierarchical structure has hard limits. Specifically, purely self-attention architectures are entirely dependent on the number of multi-attention heads and layers growing alongside the size of the input to retain the ability to model recursion, finite-state languages and other hierarchical aspects of our data (Hahn, 2020).

In order to overcome this limitation, efforts have been made to combine the benefits of sequential computation, inherent to recurrent neural networks, with the advantages of parallel computation and the global receptive field of the Transformer. This method is referred to as the *Universal Transformer* (Dehghani et al., 2019). In addition, it also includes the *Adaptive Computation Time* mechanism proposed by Graves (2016), which enables the model to dynamically learn how many computational steps to perform per the features of the input sequence. However, these advances have not yet been translated to the domain of set-input challenges.

5.6 Directions for Further Research

There are many possible directions for further investigations, pertaining to both the area of encoding sets and proposing novel permutation learning methods. As hinted in section 4.1, the question of how best to utilize the fixed length set representation within the internals of the ordering module remains open. However, we believe the areas of learning complex permutations (5.6.1) and differentiable loss functions (5.6.2) deserve separate attention, given in the following subsections.

5.6.1 Complex Permutations

An important and natural extension of the research within set-to-sequence is to apply it to challenges where the output sequence allows for repetition and exclusion of input set elements, thus going beyond traditional permutation learning (Diallo et al., 2020). As such, these *complex permutations* present an additional challenge of dynamically predicting the optimal sequence length without sacrificing the second of the two aforementioned prerequisites for ML methods on sets, namely the requirement that the same model must be able to process finite input sets of any cardinality. It is not immediately clear how to achieve both properties, with multiple promising approaches gaining prominence, ranging from the addition of a confidence loss for long time-series prediction (Harmon & Klabjan, 2019), to the aforementioned adaptive computation time (Wu et al., 2020). For an overview of dynamic neural network methods in general, see Han et al. (2021).

Important tasks involving a complex target permutation include predicting a configuration representing the assembly of off-the-shelf components into a functional system (Carlson-Skalak et al., 1998) and the composition of product offers into rendered catalogues (Liao & Chen, 2004). Additionally, an interesting area for further research would be to extend these complex permutation sequences to grids and lattices, as suggested by Zhang et al. (2019), or even to graphs, expanding on the work of Serviansky et al. (2020).

5.6.2 Differentiable Loss Functions

Another important area for further work is centered around the problem of framing the set-to-sequence challenges in such a way as to enable the use of differentiable loss functions. A naive but often practically effective approach is to frame the problem as categorization and use a cross-entropy loss, as discussed in Engilberge et al. (2019). However, it precludes meaningful distinction between pointing to an incorrect element that is very similar to the correct one and pointing to an entirely different incorrect element.

Alternatively, if a ranking framework is applied, where a score is generated for each element and subsequently used to sort all elements into a new permutation, we gain access to well documented listwise losses, such as the ones successfully employed in the ListNet or *ListMLE* (Kumar et al., 2020) frameworks. Many metrics that would lend themselves to ordering challenges do not have defined derivatives for their entire domain. The development and testing of their smooth approximations is of great potential value, as seen in the works of Rolínek et al. (2020) and Blondel et al. (2020).

6. Conclusion

Set-to-sequence is currently established as a family of methods with an exceptionally wide range of applications. At its essence, it is a combination of three areas seeing a lot of attention within the machine learning and deep learning research communities, namely set-input problems, combinatorial optimization and structured prediction. As such, a number of methods that can further our understanding of this field originates from other areas of interest, sometimes without seeing immediate application to set-to-sequence challenges. Progress owes to advances in model architectures, parallel computing, hyperparameter optimization as well as the overall growing interest in applying machine learning solutions to a wider and wider gamut of industrial challenges.

We have presented an overview of set-to-sequence methods from the fields of machine learning and deep learning. The initial sections of this paper introduced the reader to the key concepts in relation to machine learning on sets, most notably the properties of permutation invariance, permutation equivariance and the requirement of handling inputs of varying lengths. In relation to this, in section 3, the reader is introduced to a number of selected set-encoding model architectures, including a significantly detailed look at their underlying mathematical transformations. To facilitate comprehensive understanding, we compared these and other related methods through several summary tables presented in section 3.1.

Additionally, a survey of potential ordering methods has been provided in section 4. This included the three primary ways of formalizing the output permutation in the set-to-sequence setting, namely listwise ranking, relaxations of permutation matrices and attention-based pointing. Once a permutation invariant set representation is obtained through one of the aforementioned set-encoding models, the ordering methods' calculations can be conditioned on this embedding to output a sequence of elements from the original input set, in one fully trainable deep learning framework.

Acknowledgements

This work was partly supported by an Innovation Fund Denmark research grant (number 9065-00017B) and by Tjek A/S. The authors would like to acknowledge Rasmus Pagh's assistance in the conceptualization of different set-to-sequence settings and general feedback on the early drafts of this paper.

References

Achlioptas, P., Diamanti, O., Mitliagkas, I., & Guibas, L. (2018). Learning representations and generative models for 3D point clouds. In *6th International Conference on Learning Representations, ICLR 2018 - Workshop Track Proceedings*.

Ai, Q., Bi, K., Guo, J., & Croft, W. B. (2018). Learning a deep listwise context model for ranking refinement. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pp. 135–144.

Altae-Tran, H., Ramsundar, B., Pappu, A. S., & Pande, V. (2017). Low data drug discovery with one-shot learning. *ACS central science*, *3*(4), 283–293.

Arora, S. (1996). Polynomial time approximation schemes for euclidean tsp and other geometric problems. In *Proceedings of 37th Conference on Foundations of Computer Science*, pp. 2–11. IEEE.

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*, *ArXiv*.

Bahdanau, D., Brakel, P., Xu, K., Goyal, A., Lowe, R., Pineau, J., Courville, A., & Bengio, Y. (2017). An Actor-Critic Algorithm for Sequence Prediction. In *ICLR*, pp. 1–17.

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. *CoRR*, *abs/1409.0473*.

Bello, I., Kulkarni, S., Jain, S., Boutilier, C., Chi, E., Eban, E., Luo, X., Mackey, A., & Meshi, O. (2018). Seq2slate: Re-ranking and slate optimization with rnns. In *Proceedings of the Workshop on Negative Dependence in Machine Learning at the 36th International Conference on Machine Learning,*.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, *ArXiv*.

Bengio, Y., Lodi, A., & Prouvost, A. (2020). Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, *1*.

Blondel, M., Teboul, O., Berthet, Q., & Djolonga, J. (2020). Fast differentiable sorting and ranking. In *Proceedings of the 37th International Conference on Machine Learning, PMLR 119, 2020*.

Brown, N., & Mueller, C. (2017). Designing with data: Moving beyond the design space catalog. *Disciplines and Disruption - Proceedings Catalog of the 37th Annual Conference of the Association for Computer Aided Design in Architecture, ACADIA 2017*, *1*, 154–163.

Brown, T., Mann, B., Ryder, N., & Subbiah, M. (2020). Language models are few-shot learners. *ArXiv*, *abs/2005.14165*.

Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., & Li, H. (2007). Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pp. 129–136.

Carlson-Skalak, S., White, M. D., & Teng, Y. (1998). Using an evolutionary algorithm for catalog design. *Research in Engineering Design*, *10*, 63–83.

Chaudhari, S., Polatkan, G., Ramanath, R., & Mithal, V. (2019). An attentive survey of attention models. *arXiv preprint arXiv:1904.02874*, *ArXiv*.

Cheng, J., Dong, L., & Lapata, M. (2016). Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 551–561, Austin, Texas. Association for Computational Linguistics.

Cho, K., Courville, A., & Bengio, Y. (2015). Describing multimedia content using attention-based encoder-decoder networks. *IEEE Transactions on Multimedia*, *17*(11), 1875–1886.

Cohen, W. W., Schapire, R. E., & Singer, Y. (1998). Learning to order things. In *Advances in Neural Information Processing Systems*, pp. 451–457.

Cui, B., Li, Y., Chen, M., & Zhang, Z. (2018). Deep attentive sentence ordering network. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018*, pp. 4340–4349.

Cui, G., Yu, W., Li, X., Zeng, Z., & Gu, B. (2019). Machine-Learning-Driven Matrix Ordering for Power Grid Analysis. In *Proceedings of the 2019 Design, Automation and Test in Europe Conference and Exhibition, DATE 2019*, pp. 984–987. EDAA.

Cuturi, M., Teboul, O., & Vert, J.-P. (2019). Differentiable ranking and sorting using optimal transport. In *Advances in Neural Information Processing Systems*, pp. 6861–6871.

Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., & Song, L. (2017). Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems*, *2017-December*(Nips), 6349–6359.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q., & Salakhutdinov, R. (2019). Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2978–2988, Florence, Italy. Association for Computational Linguistics.

De Cao, N., & Kipf, T. (2018). MolGAN: An implicit generative model for small molecular graphs. *ICML Deep Generative Models Workshop*, *1*.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., & Kaiser, Ł. (2019). Universal transformers. In *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–23.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Diallo, A., Zopf, M., & Fürnkranz, J. (2020). Permutation learning via lehmer codes. *Frontiers in Artificial Intelligence and Applications*, *325*, 1095–1102.

Edwards, H., & Storkey, A. (2017). Towards a Neural Statistician. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pp. 1–13.

Emami, P., & Ranka, S. (2018). Learning permutations with sinkhorn policy gradient. *ArXiv*, *abs/1805.07010*.

Engilberge, M., Chevallier, L., Pérez, P., & Cord, M. (2019). Sodeep: A sorting deep net to learn ranking loss surrogates. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Ge, L., Cai, Y., Weng, J., & Yuan, J. (2018). Hand pointnet: 3d hand pose estimation using point sets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8417–8426.

Graves, A. (2016). Adaptive Computation Time for Recurrent Neural Networks. *Preprint arXiv:1603.08983*, *ArXiv*.

Graves, A., Wayne, G., & Danihelka, I. (2014). Neural turing machines. *Preprint arXiv:1410.5401*, *ArXiv*.

Grinman, A. (2015). The hungarian algorithm for weighted bipartite graphs. *Massachusetts Institute of Technology*, *MIT*.

Grover, A., Wang, E., Zweig, A., & Ermon, S. (2019). Stochastic optimization of sorting networks via continuous relaxations. *Preprint arXiv:1903.08850*, *ArXiv*.

Hahn, M. (2020). Theoretical Limitations of Self-Attention in Neural Sequence Models. *Transactions of the Association for Computational Linguistics*, *8*.

Halmos, P. R. (2017). *Naive set theory*. Courier Dover Publications.

Hamilton, W. L., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. In *NIPS*.

Han, Y., Huang, G., Song, S., Yang, L., Wang, H., & Wang, Y. (2021). Dynamic neural networks: A survey. *Preprint arXiv:2102.04906*, *ArXiv*.

Harmon, M., & Klabjan, D. (2019). Dynamic prediction length for time series with sequence to sequence networks. *arXiv*, *1*(2014), 1–23.

Hayden, S., Zermelo, E., Fraenkel, A. A., & Kennison, J. F. (1968). *Zermelo-Fraenkel set theory*. CE Merrill.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hernández-García, A., König, P., & Kietzmann, T. (2019). Learning robust visual representations using data augmentation invariance. *ArXiv*, *abs/1906.04547*.

Hu, D. (2019). An introductory survey on attention mechanisms in nlp problems. In *Proceedings of SAI Intelligent Systems Conference*, pp. 432–448. Springer.

Huang, G., Liu, S., Van der Maaten, L., & Weinberger, K. Q. (2018a). Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2752–2761.

Huang, P. S., Wang, C., Singh, R., Yih, W. T., & He, X. (2018b). Natural language to structured query generation via meta-learning. *NAACL HLT 2018 - 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, *2*, 732–738.

Ibrahim, A., R, A., & Ishaya, J. (2019). The Capacitated Vehicle Routing Problem. *International Journal of Research - Granthaalayah,*, *3*(2), 30–33.

Ionescu, C., Vantzos, O., & Sminchisescu, C. (2015). Matrix backpropagation for deep networks with structured layers. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2965–2973.

Iqbal, S., & Sha, F. (2019). Actor-attention-critic for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 2961–2970. PMLR.

Ishaya, J., Ibrahim, A., & Lo, N. (2019). A comparative analysis of the travelling salesman problem: Exact and machine learning techniques. *Open Journal of Discrete Applied mathematics*, *2*(3), 23–37.

Jain, S., & Wallace, B. C. (2019). Attention is not explanation. *Preprint arXiv:1902.10186*, *ArXiv*.

Jaiswal, A., Wu, R. Y., Abd-Almageed, W., & Natarajan, P. (2018). Unsupervised adversarial invariance. In *Advances in Neural Information Processing Systems*, pp. 5092–5102.

Johnson, J., Hariharan, B., Van Der Maaten, L., Fei-Fei, L., Lawrence Zitnick, C., & Girshick, R. (2017). Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2901–2910.

Jung, I.-S., Berges, M., Garrett, J. H., & Poczos, B. (2015). Exploration and evaluation of ar, mpca and kl anomaly detection techniques to embankment dam piezometer data. *Adv. Eng. Inform.*, *29*(4), 902–917.

Karch, T., Colas, C., Teodorescu, L., Moulin-Frier, C., & Oudeyer, P.-Y. (2020). Deep sets for generalization in rl. *Preprint arXiv:2003.09443*, *ArXiv*.

Karlin, A. R., Klein, N., & Gharan, S. O. (2020). A (slightly) improved approximation algorithm for metric tsp. *ArXiv*, *abs/2007.01409*.

Kayhan, O. S., & Gemert, J. C. v. (2020). On translation invariance in cnns: Convolutional layers can exploit absolute spatial location. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14274–14285.

Kersting, K., Kriege, N. M., Morris, C., Mutzel, P., & Neumann, M. (2016). Benchmark data sets for graph kernels. *URL http://graphkernels. cs. tu-dortmund. de*, *29*.

Kool, W., Van Hoof, H., & Welling, M. (2019). Attention, learn to solve routing problems!. In *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–25.

Kosiorek, A. R., Kim, H., & Rezende, D. J. (2020). Conditional Set Generation with Transformers. In *Workshop on Object-Oriented Learning at ICML 2020*.

Kumar, P., Brahma, D., Karnick, H., & Rai, P. (2020). Deep Attentive Ranking Networks for Learning to Order Sentences. In *AAAI 2020*.

Lake, B. M., Salakhutdinov, R., & Tenenbaum, J. B. (2019). The omniglot challenge: a 3-year progress report. *Current Opinion in Behavioral Sciences*, *29*, 97–104.

Lan, Y., Liu, T.-Y., Ma, Z., & Li, H. (2009). Generalization analysis of listwise learning-to-rank algorithms. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 577–584.

Lee, J., Lee, Y., Kim, J., Kosiorek, A. R., Choi, S., & Teh, Y. W. (2019). Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. *Proceedings of the 36th International Conference on Machine Learning*, *36*.

Lei, Y., Li, W., Lu, Z., & Zhao, M. (2017). Alternating pointwise-pairwise learning for personalized item ranking. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 2155–2158.

Liao, S. H., & Chen, Y. J. (2004). Mining customer knowledge for electronic catalog marketing. *Expert Systems with Applications*, *27*(4), 521–532.

Lin, T.-Y., Maji, S., & Koniusz, P. (2018). Second-order democratic aggregation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 620–636.

Linderman, S., Mena, G., Cooper, H., Paninski, L., & Cunningham, J. (2018). Reparameterizing the birkhoff polytope for variational permutation inference. In *International Conference on Artificial Intelligence and Statistics*, pp. 1618–1627. PMLR.

Ling, J., Jones, R., & Templeton, J. (2016). Machine learning strategies for systems with invariance properties. *Journal of Computational Physics*, *318*, 22–35.

Liu, C., & Shum, H.-Y. (2003). Kullback-leibler boosting. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, Vol. 1, pp. I–I. IEEE.

Liu, X., Weijer, J. V. D., & Bagdanov, A. D. (2019). Exploiting Unlabeled Data in CNNs by Self-Supervised Learning to Rank. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *41*(8), 1862–1878.

Liu, Z., Luo, P., Wang, X., & Tang, X. (2015). Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.

Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., & Kipf, T. (2020). Object-Centric Learning with Slot Attention. *arXiv*, *1*(NeurIPS), 1–27.

Logeswaran, L., Lee, H., & Radev, D. (2018). Sentence ordering and coherence modeling using recurrent neural networks. In *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pp. 5285–5292.

Loosli, G., Canu, S., & Bottou, L. (2007). Training invariant support vector machines using selective sampling. *Large scale kernel machines*, *2*.

Ma, S., Deng, Z.-H., & Yang, Y. (2016). An unsupervised multi-document summarization framework based on neural document model. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pp. 1514–1523.

Mani, K., Verma, I., Meisheri, H., & Dey, L. (2018). Multi-document summarization using distributed bag-of-words model. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pp. 672–675. IEEE.

Marcos, Diego, V. (2016). Learning rotation invariant convolutional filters for texture classification. In *2016 23rd International Conference on Pattern Recognition*, pp. 2012–2017. IEEE.

Meister, C., Vieira, T., & Cotterell, R. (2020). If beam search is the answer, what was the question?. *Preprint arXiv:2010.02650*, *ArXiv*.

Mena, G., Belanger, D., Linderman, S., & Snoek, J. (2018). Learning latent permutations with gumbel-sinkhorn networks. In *International Conference on Learning Representations 2018*.

Meng, C., Yang, J., Ribeiro, B., & Neville, J. (2019). HATS: A hierarchical sequence-attention framework for inductive set-of-sets embeddings. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019*, pp. 783–792.

Merity, S., Xiong, C., Bradbury, J., & Socher, R. (2016). Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*.

Mitra, B., Craswell, N., et al. (2018). An introduction to neural information retrieval. *Foundations and Trends® in Information Retrieval*, *13*(1), 1–126.

Moore, J., & Neville, J. (2017). Deep collective inference. In *AAAI*.

Murphy, R. L., Srinivasan, B., Ribeiro, B., & Rao, V. (2019). Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–21.

Nazari, M., Oroojlooy, A., Takáč, M., & Snyder, L. V. (2018). Reinforcement learning for solving the vehicle routing problem. *Advances in Neural Information Processing Systems*, *2018-December*, 9839–9849.

Niepert, M., Ahmed, M., & Kutzkov, K. (2016). Learning convolutional neural networks for graphs. In *International conference on machine learning*, pp. 2014–2023.

Nishida, N., & Nakayama, H. (2017). Word ordering as unsupervised learning towards syntactically plausible word representations. In *Proceedings of the 8th International Joint Conference on Natural Language Processing (IJCNLP 2017)*, pp. 70–79.

Noreen, N., Palaniappan, S., Qayyum, A., Ahmad, I., Imran, M., & Shoaib, M. (2020). A deep learning model based on concatenation approach for the diagnosis of brain tumor. *IEEE Access*, *8*, 55135–55144.

Ntampaka, M., Trac, H., Sutherland, D., Fromenteau, S., Poczos, B., & Schneider, J. (2016). Dynamical mass measurements of contaminated galaxy clusters using machine learning. *The Astrophysical Journal*, *831*.

Oladosu, A., Xu, T., Ekfeldt, P., Kelly, B. A., Cranmer, M., Ho, S., Price-Whelan, A. M., & Contardo, G. (2020). Meta-learning one-class classification with deepsets: Application in the milky way. *Preprint arXiv:2007.04459*, *ArXiv*.

Oliveira, H., Silva, C., Machado, G. L., Nogueira, K., & Santos, J. A. d. (2020). Fully convolutional open set segmentation. *Preprint arXiv:2006.14673*, *ArXiv*.

Pang, L., Xu, J., Ai, Q., Lan, Y., Cheng, X., & Wen, J. (2020). Setrank: Learning a permutation-invariant ranking model for information retrieval. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 499–508.

Pfannschmidt, K., Gupta, P., & Hüllermeier, E. (2018). Deep architectures for learning context-dependent ranking functions. *ArXiv*, *abs/1803.05796*.

Pihera, J., & Musliu, N. (2014). Application of machine learning to algorithm selection for tsp. In *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, pp. 47–54. IEEE.

Puduppully, R., Dong, L., & Lapata, M. (2019). Data-to-Text Generation with Content Selection and Planning. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*, 6908–6915.

Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2017). PointNet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, pp. 77–85.

Rang, B., & Thomas, W. (1981). Zermelo's discovery of the "russell paradox". *Historia Mathematica*, *8*(1), 15–22.

Rocktäschel, T., Grefenstette, E., Hermann, K., Kociský, T., & Blunsom, P. (2016). Reasoning about entailment with neural attention. *CoRR*, *abs/1509.06664*.

Rolínek, M., Musil, V., Paulus, A., Vlastelica, M., Michaelis, C., & Martius, G. (2020). Optimizing rank-based metrics with blackbox differentiation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7620–7630.

Saito, Y., Nakamura, T., Hachiya, H., & Fukumizu, K. (2019). Deep set-to-set matching and learning. *ArXiv*, *abs/1910.09972*.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, p. 285–295, New York, NY, USA. Association for Computing Machinery.

Schafer (2007). Recurrent neural networks are universal approximators. *International Journal of Neural Systems, Vol. 17, No. 4 (2007) 253–263, World Scientific Publishing Company Int.*, *17*(4), 253–263.

See, A., Liu, P., & Manning, C. (2017). Get to the point: Summarization with pointer-generator networks. In *Association for Computational Linguistics*.

Serviansky, H., Segol, N., Shlomi, J., Cranmer, K., Gross, E., Maron, H., & Lipman, Y. (2020). Set2graph: Learning graphs from sets. *Preprint arXiv:2002.08772*, *ArXiv*.

Skianis, K., Nikolentzos, G., Limnios, S., & Vazirgiannis, M. (2020). Rep the Set: Neural Networks for Learning Set Representations. *AISTATS*, *1*(23rd).

Smith, K. A. (1999). Neural networks for combinatorial optimization: A review of more than a decade of research. *INFORMS Journal on Computing*, *11*(1), 15–34.

Spearman, C. (1904). The proof and measurement of association between two things.. In *American Journal of Psychology*.

Stuart, J. L., & Weaver, J. R. (1991). Matrices that commute with a permutation matrix. *Linear Algebra and its Applications*, *150*, 255 – 265.

Sun, Z., Tang, J., Du, P., Deng, Z. H., & Nie, J. Y. (2019). DivGraphPointer: A graph pointer network for extracting diverse keyphrases. In *SIGIR 2019 - Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 755–764.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., & Graepel, T. (2018). Value-decomposition networks for cooperative multi-agent learning based on team reward. *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, *3*, 2085–2087.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, *4*(January), 3104–3112.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826.

Takeuti, G., & Zaring, W. M. (2013). *Axiomatic set theory*, Vol. 8. Springer Science & Business Media.

Taylor, L., & Nitschke, G. (2018). Improving deep learning with generic data augmentation. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1542–1547.

Vartak, M., & Thiagarajan, A. (2017). A Meta-Learning Perspective on Cold-Start Recommendations for Items Manasi. *31st Conference on Neural Information Processing Systems (NIPS 2017)*, *29*(3), 171–180.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *2017-Decem*(Nips), 5999–6009.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph Attention Networks. In *International Conference on Learning Representations (2018)*. Accepted as poster.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., & Mathieu (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, *575*(7782), 350–354.

Vinyals, O., Bengio, S., & Kudlur, M. (2016). Order matters: Sequence to sequence for sets. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pp. 1–11.

Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer networks. In *Advances in Neural Information Processing Systems*, Vol. 2015-January, pp. 2692–2700.

Volkovs, M. N., & Zemel, R. S. (2009). Boltzrank: learning to maximize expected ranking gain. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1089–1096.

Wagstaff, E., Fuchs, F. B., Engelcke, M., Posner, I., & Osborne, M. (2019). On the limitations of representing functions on sets. *36th International Conference on Machine Learning, ICML 2019*, *2019-June*, 11285–11298.

Wang, J., Song, Y., Leung, T., Rosenberg, C., Wang, J., Philbin, J., Chen, B., & Wu, Y. (2014). Learning fine-grained image similarity with deep ranking. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1386–1393.

Wang, T., & Wan, X. (2019). Hierarchical Attention Networks for Sentence Ordering. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*, 7184–7191.

Weston, J., Chopra, S., & Bordes, A. (2015). Memory Networks. In *International Conference on Learning Represen- tations, ICLR, 2015*, pp. 1–15.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, *8*(3-4), 229–256.

Wiseman, S., & Rush, A. M. (2016). Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1296–1306, Austin, Texas. Association for Computational Linguistics.

Wolf, T., Sanh, V., Chaumond, J., & Delangue, C. (2019). TransferTransfo: A Transfer Learning Approach for Neural Network Based Conversational Agents. *Association for the Advancement of Artificial Intelligence*, *1*(ii).

Wu, Y., Riedel, S., Minervini, P., & Stenetorp, P. (2020). Don't read too much into it: Adaptive computation for open-domain question answering. *Preprint arXiv:2011.05435*, *ArXiv*.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., & Xiao, J. (2015). 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1912–1920.

Xia, F., Liu, T.-Y., Wang, J., Zhang, W., & Li, H. (2008). Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*, pp. 1192–1199.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., & Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pp. 2048–2057.

Xu, K., Jegelka, S., Hu, W., & Leskovec, J. (2019). How powerful are graph neural networks?. In *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–17.

Xu, Z., Zhu, L., & Yang, Y. (2017). Few-shot object recognition from machine-labeled web images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1164–1172.

Yang, B., Wang, S., Markham, A., & Trigoni, N. (2020). Robust attentional aggregation of deep feature sets for multi-view 3d reconstruction. *International Journal of Computer Vision*, *128*(1), 53–73.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., & Leskovec, J. (2018). Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*, pp. 4800–4810.

Yu, T., Meng, J., & Yuan, J. (2018). Multi-view harmonized bilinear network for 3d object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 186–194.

Zaheer, M., Kottur, S., Ravanbhakhsh, S., Póczos, B., Salakhutdinov, R., & Smola, A. J. (2017). Deep sets. *Advances in Neural Information Processing Systems*, *2017-December*(ii), 3392–3402.

Zhang, M., Cui, Z., Neumann, M., & Chen, Y. (2018). An end-to-end deep learning architecture for graph classification. In *AAAI*.

Zhang, Y., Hare, J., & Prügel-Bennett, A. (2019). Deep Set Prediction Networks. In *NeurIPS 2019*.

Zhang, Y., Hare, J., & Prügel-Bennett, A. (2020). FSPool: Learning Set Representations with Featurewise Sort Pooling. In *ICLR 2020*.

Zhang, Y., Prügel-Bennett, A., & Hare, J. (2019). Learning representations of sets through optimized permutations. In *7th International Conference on Learning Representations, ICLR 2019*.

Zhao, Z.-Q., tao Xu, S., Liu, D., Tian, W., & Jiang, Z.-D. (2019a). A review of image set classification. *Neurocomputing*, *335*, 251–260.

Zhao, Z. Q., Zheng, P., Xu, S. T., & Wu, X. (2019b). Object Detection with Deep Learning: A Review..

Zheng, L., Shen, L., Tian, L., Wang, S., Wang, J., & Tian, Q. (2015). Scalable person re-identification : A benchmark university of texas at san antonio. In *ICCV 2015*, pp. 1116–1124.

Zhong, V., Xiong, C., & Socher, R. (2018). Seq2sql: Generating structured queries from natural language using reinforcement learning. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*.

Chapter 5

Article 2

Article details:

2. **"PROCAT: Product Catalogue Dataset for Implicit Clustering, Permutation Learning and Structure Prediction"**
   Mateusz Jurewicz and Leon Derczynski
   Proceedings of the Thirty-fifth Conference on Neural Information Processing Systems, Datasets and Benchmarks Track, NeurIPS 2021.

Context. At the time of writing the second published article there appeared to be no product catalog dataset that would lend itself to the kind of structure prediction tasks inherent to the Catalog Problem. As such, a natural step was to curate such a dataset, further enriched with an early version of the synthetic catalog library outlined in Chapter 3.1. Therefore, the intention was twofold. The hope was to both popularize the specific, e-commerce application and provide early benchmarks for future models capable of addressing the underlying Catalog Problem.

Contributions. Within the second included article, the PROCAT dataset of over 10,000 human-designed product catalogs is introduced. Comprised of more than 1.5 million individual product offers, which are grouped into almost a quarter of a million complementary sections, PROCAT provides the supervision data needed for training subsequent neural models. Additionally, it forms an NLP corpus for an underrepresented language (Danish). The provided catalog examples are given in both raw

and highly preprocessed form, allowing the researchers to quickly begin testing their own methods of approaching the Catalog Problem. Furthermore, the article reports early benchmarks of performance on the provided datasets, quantitatively comparing the results of earlier set encoding methods combined with permutation learning modules into complete set-to-sequence architectures via two rank correlation coefficients, Kendall's τ and Spearman's ρ, as well as an early version of the compositional and structural metrics outlined in Chapter 3.1.1.

Recent developments. To my knowledge, no new dataset lending itself to a supervised approach to the Catalog Problem has been made publicly available since the publication of the included article. Whilst there exist machine learning datasets with the field of e-commerce that contain some information pertaining to product catalogs, such as Fashion-mnist introduced by Xiao *et al.* (2017) and the review-centric dataset used by Haque *et al.* (2018), PROCAT remains the only dataset containing product catalog information needed for the prediction of a catalog's composition and structure.

5.1 PROCAT: Product Catalogue Dataset for Implicit Clustering, Permutation Learning and Structure Prediction

In this section, the second published article is included in unchanged form, starting on the next page for ease of reading.

PROCAT: Product Catalogue Dataset for Implicit Clustering, Permutation Learning and Structure Prediction

Mateusz Jurewicz*
Department of Computer Science
IT University of Copenhagen
København, 2300
maju@itu.dk

Leon Derczynski
Department of Computer Science
IT University of Copenhagen
København, 2300
leod@itu.dk

Abstract

In this dataset paper we introduce PROCAT, a novel e-commerce dataset containing expertly designed product catalogues consisting of individual product offers grouped into complementary sections. We aim to address the scarcity of existing datasets in the area of set-to-sequence machine learning tasks, which involve complex structure prediction. The task's difficulty is further compounded by the need to place into sequences rare and previously-unseen instances, as well as by variable sequence lengths and substructures, in the form of diversely composed catalogues. PROCAT provides catalogue data consisting of over 1.5 million set items across a 4-year period, in both raw text form and with pre-processed features containing information about relative visual placement. In addition to this ready-to-use dataset, we include baseline experimental results on a proposed benchmark task from a number of joint set encoding and permutation learning model architectures.

1 Introduction

Intelligent product presentation systems and catalogue structure prediction are important areas of research, with clear practical applications [de Melo et al., 2019] and a substantial impact on the environment [Liu et al., 2020]. With the ultimate goal being the reduction of paper waste stemming from print catalogues, in this paper we present a dataset of over 10,000 catalogues consisting of more than 1.5 million individual product offers. This dataset lends itself to machine learning research in the area of set-to-sequence structure prediction, clustering and permutation learning.

Whilst there are many e-commerce product datasets containing information about individual product offers for the purposes of recommendation [Fu et al., 2020] and categorization [Lin et al., 2019], there is a scarcity of publicly-available, easily accessible and reliably maintained product datasets for catalogue structure prediction and permutation learning. Providing such a dataset can help foster the transition from print to digital catalogues [Wirtz-Brückner and Jakobs, 2018].

This task is challenging for machine learning methods due to the necessity of learning to obtain useful representations of **rare and unseen instances** of product offers, the **variable offer and catalogue lengths**, as well as the **implicit clustering task** necessary for predicting the split of offers into a **varying number of clusters** (sections) to output the final catalogue structure.

With this work, we aim to address this domain lacuna in three ways. First, we provide a large dataset of product catalogues designed by marketing experts. These are structured, and the task over them is to predict a catalogue structure given a set of product offers (the set items). This structure takes the form of grouping product offers into complementary sections and ordering or *permuting* the

*Affiliated with the Tjek A/S Machine Learning Department (København, 1408), contact via mj@tjek.com.

35th Conference on Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks.

sections into a compelling catalogue narrative [Szilas et al., 2020], a currently qualitative aspect of human-performed task.

Second, we perform a series of experiments on this dataset, obtain initial benchmarks of performance and propose a number of combined set-to-sequence model architectures. These architectures, along with all model parameters, are also made publicly available, along with a repository containing all code necessary for repeated experiments.

Third, we supplement the real-world catalogue data with a code library for generating simplified, automatically-synthesized product catalogues that adhere to flexible, adjustable structural and distributional rules. These synthetic catalogues can then be used to train set-to-sequence structure prediction models analogous to the ones we benchmark on the main dataset. Additionally, the library allows for detailed functional metrics on the performance of these models, grouped into specific aspects of the chosen structural rules. This allows for greater insight into what kinds of structures different types of models are effective at learning and full control over the task's difficulty.

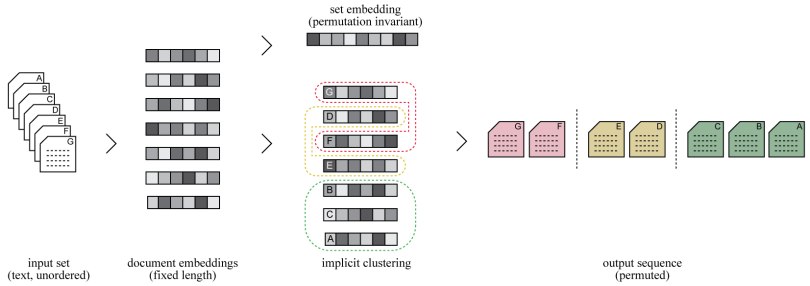


Figure 1: Diagram visualizing the core set-to-sequence structure prediction task through permutation learning with implicit clustering and set representation learning.

The remainder of this paper is structured in the following way: in section 2 we elaborate on prior work, existing datasets and relevant structure prediction methods in more detail. In section 3 we introduce the specifics of the main dataset contribution, including data collection, composition, pre-processing, distribution and ethical considerations. For further details regarding the dataset see the datasheets for datasets checklist in the supplementary materials. In subsection 3.4, we outline the synthetic dataset generation library and its related functional testing capacities. We then move on to section 4, where the experimental setup and initial benchmark results are presented. Finally, sections 5 and 6 discuss the limitations of our work and conclusions respectively, with minor notes on the potential for future work.

1.1 Our contributions

- PROCAT dataset of over 10,000 human-designed product catalogues consisting of more than 1.5 million individual product offers, across 15 GPC commercial product categories.
- Library for generating simplified, synthetic catalogues according to chosen structural rules and measuring related model performance through functional tests, with full control over the task's difficulty.
- Benchmark evaluation tasks and baseline results for 4 proposed deep learning models utilizing both datasets.

The links to all mentioned resources including the PROCAT dataset, the code repository for repeated experiments and the best performing model weights are provided in the supplementary materials. A direct link to the dataset is also provided here for convenience: `https://doi.org/10.6084/m9.figshare.14709507`.

2 Prior work

Research interest into the process of digitizing paper product catalogues into internet-based electronic product catalogues (IEPCs / EPCs) has a long history [Palmer, 1997, Stanoevska-Slabeva and Schmid,

2000, Guo, 2009, de Melo et al., 2019]. There are ample machine learning datasets consisting of individual products [Xiao et al., 2017] or product reviews [Haque et al., 2018], but excluding information about the structure of a readable catalogue composed from such offers. To the authors' knowledge, no publicly available dataset containing both the features of individual product offers and the order and grouping in which they were presented as a product catalogue exists.

In order to empower more businesses to present their available products in a visually pleasing digital form and move away from wasteful paper-based solutions, an automatic method for turning a set of offers into a structured presentation needs to be obtained [Guo, 2009]. We propose a set-to-sequence formulation of this task, enabling machine learning models to learn the optimal structure of a viewable product catalogue from historic examples.

With that framing of the task in mind, a very brief overview of existing set-to-sequence, permutation learning model architectures and datasets is given below.

2.1 Set-to-sequence methods

Machine learning set-to-sequence methods can approximate solutions to computationally expensive combinatorial problems in many areas. They have been applied to learning competitive solvers for the NP-Hard Travelling Salesman Problem [Vinyals et al., 2015]; tackling prominent NLP challenges such as sentence ordering [Wang and Wan, 2019] and text summarization [Sun et al., 2019]; and in multi-agent reinforcement learning [Sunehag et al., 2018]. A notable example is the agent employed by the AlphaStar model, which defeated a grandmaster level player in the strategy game of Starcraft II, where set-to-sequence methods were used to manage the structured, combinatorial action space [Vinyals et al., 2019]. For a survey of set-to-sequence in machine learning, see Jurewicz and Derczynski [2021].

These model architectures often obtain a meaningful, permutation-invariant representation of the entire available set of entities [Zaheer et al., 2017], either through adjusted recurrent neural networks [Vinyals et al., 2016] or transformer-based methods [Lee et al., 2019]. This is then followed by a permutation learning module whose output is conditioned on the above-mentioned representation. Such modules can take many forms, ranging from listwise ranking [Ai et al., 2018], through permutation matrix prediction [Zhang et al., 2019] to attention-based pointing [Yin et al., 2020].

2.2 Set-to-sequence datasets

In lieu of domain-specific datasets for product catalogue structure prediction through set-to-sequence permutation learning, we can look to other areas of machine learning research where predicting a permutation is the goal. These include sentence ordering [Cui et al., 2018], where any source of consecutive natural language sentences can be used, such as the NIPS abstract, AAN abstract, NSF abstract datasets [Logeswaran et al., 2018]. However, this formulation precludes the model from learning an implicit clustering.

Furthermore, sequential natural language tasks such as sentence continuation are fundamentally different from catalogue structure prediction because word tokens come from a predefined vocabulary, whereas new offers may have never been seen before by our models, presenting a further challenge.

Alternatively, one can look to learn-to-rank datasets from the domain of information retrieval, such as Istella LETOR[1] or MSLR30K[2], as used for permutation learning by Pang et al. [2020]. However, learn-to-rank frameworks presuppose an existence of a *query* for which a relevance rating is assigned to each document, which are then sorted according to this rating. It is unclear what could constitute the query in the context of product catalogue structure prediction. The permutation invariant representation of the entire set of available offers is a possible candidate, requiring further research, as mentioned in the conclusion section (6).

Finally, there exist ways to obtain visual permutation datasets consisting of image mosaics, where the task is to reorder the puzzle pieces back into the original image. Santa Cruz et al. [2018] obtain these mosaics from the Public Figures and OSR scene datasets [Parikh et al., 2012]. This resembles the product catalogue prediction task in terms of permuting previously unseen atomic instances (image fragments), but lacks the element of implicit clustering into meaningful, complementary sections.

[1] http://blog.istella.it/istella-learning-to-rank-dataset/

[2] http://research.microsoft.com/en-us/projects/mslr/

Table 1: Sample PROCAT offers with raw text features

section	header	description	priority
1	Lamb chops	Approx. 400 grams. Marinated chops with mushrooms, bacon. Best served with cream.	A
1	Ham roast	700-800 grams. Oriental. Mexico.	B
1	Melon	Organic piel de sapo or cantaloupe melon. Unit price 20.00. Spain, 1st class.	C
2	Hair spray	ELNETT. Extra strong. Strong hold. 400 ml.	A
2	Deodorants	Spray. Roll-on. 50-150 ml. REXONA	B

3 PROCAT

In order to mitigate the lack of product catalogue datasets, with the prediction target being a complex permutation requiring implicit clustering, we propose a new dataset further referred to as PROCAT.

This dataset consists of 11,063 human-designed catalogue structures, made up of 1,613,686 product offers with their text features, grouped into a total of 238,256 sections. The dataset's diversity stems from the catalogues covering 15 different GPC-GS1 commercial categories and from their original composition being created by 2398 different retailers, including cross-border shops that have a significant following in Denmark and neighboring Scandinavian countries, particularly Sweden and Norway, as well as Germany. For more details, see the supplementary materials.

What follows is a more in-depth look into the collection and content of this data. For an introductory excerpt demonstrating sample offers from the same catalogue through raw text features, section assignment and priority class, see table 1.

Additionally, we briefly introduce a supplementary library for generating simpler, synthetic structures meant to resemble product catalogues in section 3.4.

3.1 Data collection

The data was acquired through a combination of feed readers and custom scraping scripts developed by Tjek A/S, a Danish e-commerce company. The scripts read the feeds and scrape a list of stores and PDF catalogs associated with said stores. Afterwards, a human curation step is performed by the operations department to make sure the obtained data is correct.

The data was collected within the full 4 year period between 2015 and 2019. The original structure of each catalogue is preserved through retaining information about which offers were presented together on which section (page), what the order of sections was and through a separate feature referred to as *priority class*, which represents the relative size of the corresponding offer's image on the page in the original catalogue. A visual representation is given in figure 2.

Figure 2: Product offers grouped into 3 consecutive sections extracted from a single catalogue.

3.2 Catalogue data

The dataset consists of instances representing 3 types of entities. The most atomic entity is an *offer*, which represents a specific product with a text heading and description, which often includes its on-offer price. Individual product offers are then grouped into *sections*, which represent pages in

a physical catalogue brochure. Finally, an ordered list of sections comprise a single *catalogue*, for which a prediction about its optimal structure is made. This takes the form of permuting the input set of offers into an ordered list, with section breaks marking the start and end of a section.

Each offer instance consists of its unique id, its related section and catalogue ids, a text heading and description in both raw form and as lowercase word tokens obtained via the nltk tokenizer [Bird, 2006], the total token count, and finally the full offer text as a vector referencing a vocabulary of the most common 300 thousand word tokens. Additionally, each offer is categorized into a priority class, representing how visually prominent it was in the original catalogue in terms of relative image size (on a 1-3 integer scale).

Each catalogue instance consists of its unique id, an ordered list of associated section ids, and an ordered list of offer ids that comprise the catalogue in question, including section break markers. Additionally, each catalogue instance also includes information in the form of ordered lists of sections, each containing a list of offers as vectors, with their corresponding priority class and the catalogue's length as the total number of offers within it. Finally, a randomly shuffled x of offer vectors (with section breaks) is provided for each catalogue, along with the target y representing the permutation required to restore the original order.

Every catalogue instance consists of both raw data and pre-processed features. The dataset is not a sample, it contains all catalogue instances from the years 2015 - 2019 available for viewing in the Tjek A/S app. No other selection filter was used. For a more detailed look at the structure and format of the files comprising the dataset, please see the code repository linked in the supplementary materials.

3.3 Sustainability

The dataset is made publicly available under the CC BY-NC-SA license. It is hosted by *figshare*, an open access repository where researchers can preserve and share their research outputs, supported by Digital Science & Research Solutions Ltd. The platform was chosen due its prominence, provision of a persistent identifier and rich metadata for discoverability. The dataset will be continuously maintained by the authors of this paper, who can be contacted via the emails provided in the contact information above the abstract.

If labeling errors are found, they will be corrected. The dataset may be expanded with further instances, depending on the academic interest. All previous versions of the dataset will continue to be available. Others are encouraged to extend the dataset and can choose to do so either in cooperation with the authors or individually, in accordance with the chosen license.

3.4 Synthetic data and functional testing

In order to experimentally demonstrate the initial viability of model architectures on the type of structure prediction task presented by the product catalogues, we also propose a library for generating simpler, synthetic catalogue datasets. Additionally, we enable researchers to use this library to easily specify hand-picked distributional, structural and clustering rules that determine what kinds of synthetic catalogues are generated. Finally, we provide tooling for obtaining detailed metrics regarding the models' performance per specified rule.

The synthetic datasets also allow for predicting multiple valid catalogue structures from the same underlying input set, which addresses an important limitation of the main dataset, where only one target permutation is available.

The main difference between the real and synthetic datasets is that the basic building block of a catalogue in the latter case takes the form of a vocabulary-based token representing a single product offer. This circumvents some of the difficulty related to representation learning in a few and zero shot setting inherent to the main PROCAT dataset. It becomes natural to think of each offer as representing a member of a wider, colour-coded class, such as green for vegetables, red for meats and so forth. For a visual example see figure 3.

The chosen clustering and structural rules can include pairwise and higher-order interactions between offer types. For example, the presence of both a green and purple offer type in the initial available set can result in a rule which forces the catalogue to be opened with an all-purple section and closed with

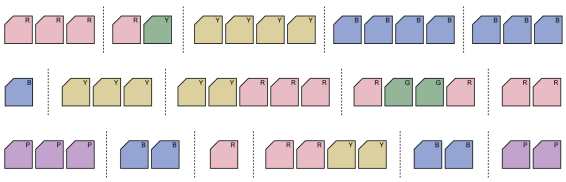


Figure 3: Three synthetic catalogue sequences, consisting of instances of 5 colour-coded offer types, separated into sections and ordered according to chosen distributional, clustering and structural rules. a mixed red and yellow section. The presence of all three primary colours can make a mixed purple and blue section invalid, forcing these offers to be split between two separate sections and so forth.

The ability to obtain structure prediction accuracy metrics per rule enables us to, for example, experimentally test the ability of models such as the Set Transformer [Lee et al., 2019] to encode such higher order interactions in various controlled settings.

4 Benchmark task and results

The data provided in PROCAT can motivate a number of benchmarking tasks related to representation learning, clustering, catalogue completion and structure prediction. We focus on a permutation learning approach to predicting the proper structure of a product catalogue, with implicit clustering of the provided set of offers into varying-length sections.

4.1 Baseline methods

Three baseline model architectures are tested, both on a set of synthetically generated catalogue structures and on the main PROCAT dataset.

Each method consists of a set encoding module and an attention-based pointing mechanism [Vinyals et al., 2015, Yin et al., 2020] for outputting the predicted permutation. The encoding module first obtains an embedding of individual offers through a recurrent neural network consisting of gated recurrent units [Chung et al., 2014] and then uses one of the three included methods of deriving the embedded representation of the entire set, which is permutation-invariant in 3 of the 4 cases.

The single exception to permutation invariance is a pure Pointer Network (1), which encodes the set sequentially through a stack of bidirectional LSTMs [Hochreiter and Schmidhuber, 1997, Schuster and Paliwal, 1997]. The remaining 3 methods are the Read-Process-Write model (2) [Vinyals et al., 2016], the Deep Sets encoder (3) [Zaheer et al., 2017] and the Set Transformer (4) [Lee et al., 2019].

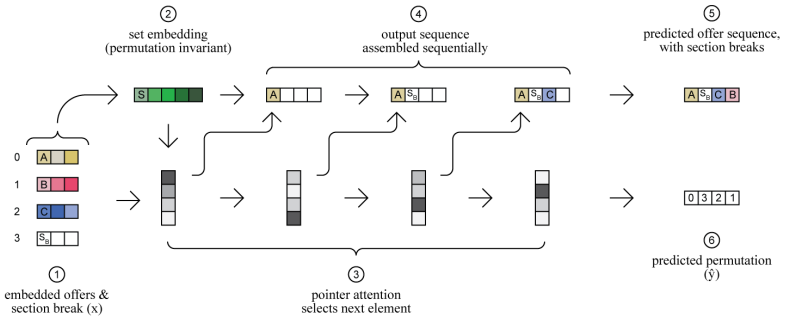


Figure 4: The input and output of the tested models, after the offer text embedding step.

In effect the random, shuffled order in which the available set of offers is originally presented to the model does not influence the representation of the set in methods 2, 3 and 4. The output of the attention-based pointing module is conditioned on this set representation through concatenating it with the embedding of each individual offer constituting the set. All models are implemented in

Table 2: Rank correlation coefficients for PROCAT

	PROCAT		Synthetic ($n = 20$)	
Model	Spearman ρ	Kendall τ	Spearman ρ	Kendall τ
Random Baseline	0.004	-0.01	0.09	-0.07
Pointer Network (2015)	0.26	0.13	0.49	0.37
Read-Process-Write (2016)	0.30	0.18	0.52	0.41
DeepSets (2017)	0.35	0.22	0.55	0.44
Set Transformer (2019)	**0.44**	**0.30**	**0.61**	**0.49**

PyTorch following code written by their respective authors (where provided), and made publicly available on GitHub.

For a visual explanation of the input and output of the permutation-learning modules of the neural networks, see figure 4. The input to the compared models is always a list of raw-text documents representing offer instances, in a randomly permuted order that needs to be reverted to the target one.

4.2 Experimental setup and results

We perform experiments on an 80-20 training-validation split of the PROCAT dataset. Every model's weights are adjusted based on a cross entropy loss applied to the pointer attention vector over all set input elements at each step of the output sequence [Yin et al., 2020]. We use two rank correlation coefficients as our metrics, namely Spearman's rho (s_ρ):

$$s_\rho(y, \hat{y}) = 1 - \frac{6 \sum_{i=1}^{n} y_i - \hat{y}_i}{n(n^2 - 1)} \tag{1}$$

where y is the target permutation in the form of integer ranks per element and $\hat{y}$ is the prediction; and Kendall's tau (k_τ), which is calculated based on the number of concordant pairs between the target and predicted rank assignments [Shieh, 1998]. Additionally, we provide an aggregated percentage based correctness metric tracking how many elements per example input set were placed correctly.

Training on PROCAT is performed for 300 epochs with batch size of 64 using the Adam stochastic optimizer [Kingma and Ba, 2015] with a learning rate 10^{-4} and momentum 0.9. Each catalogue consists of $n = 200$ offers. Training on the synthetic dataset of 50,000 catalogue sequences of $n = 20$ elements is performed for 400 epochs with the same batch size and optimization hyperparameters, training on the synthetic dataset with sequences of $n \in \{15, 10\}$ is performed for 600 epochs, in an effort to show the feasibility of achieving better performance through the proposed, scaled-up set-to-sequence model architectures.

Every PROCAT model had a total of approximately 1 million trainable parameters, every model tested on the synthetic dataset had approximately 900 thousand. For details on the dimensions of layers, see the provided repository with code for repeated experiments.

An important implementation nuance comes in the form of progressive masking preventing the models from repeatedly pointing to the same element, which forces the output to be a valid permutation. It is also important to note that we do not currently directly measure the quality of clusters (sections) in PROCAT, and that whilst the target number of clusters varies per catalogue instance, that number is known to the model through the total count of section break tokens in the input set.

4.2.1 PROCAT results

Tables 2 and 3 present results for each of the 4 tested models and a baseline which always outputs valid but random permutations of the original input set. The final values of the Spearman's ρ and Kendall's τ rank correlation coefficients are given for both the PROCAT dataset, with average cardinality of the input set (and therefore the length of the predicted permutation sequence) $n = 200$, and a sample of synthetic catalogue structures with $n \in \{20, 15, 10\}$. Metrics are averaged over 5 full training runs.

Overall, the models that obtain a permutation invariant representation of the set consistently perform better on the PROCAT dataset than a pure Pointer Network, which encodes the set sequentially through stacked RNNs. Furthermore, the top performing method has a built in mechanism for

Table 3: Rank correlation coefficients for synthetic datasets

	Synthetic ($n = 15$)		Synthetic ($n = 10$)	
Model	Spearman ρ	Kendall τ	Spearman ρ	Kendall τ
Random Baseline	-0.026	-0.019	0.051	0.023
Pointer Network (2015)	0.67	0.54	0.73	0.61
Read-Process-Write (2016)	0.77	0.60	0.83	0.71
DeepSets (2017)	0.84	0.72	0.92	0.80
Set Transformer (2019)	**0.96**	**0.85**	**0.98**	**0.93**

encoding pairwise and higher-order interactions between set elements through transformer-style attention. Domain expertise suggests that interplay between individual product offers is indeed crucial when designing a product catalogue [Xu et al., 2013].

In figure 5 an analogous comparison of the average percentage of correctly predicted ranks per input set is given. Overall, the initial results are relatively low (under 7% for the Set Transformer), which illustrates the difficulty of the underlying task. Specifically, being able to predict a good section consisting of complementary offers but placing this section later in the output catalogue than in the original one would here be reflected with a 0% score regarding those elements. However, performance of the attention-based set encoder is more consistent, as indicated by narrower error bars.

Development of a more sensitive evaluation metric is both a direction for future work and the motivation behind the creation of the synthetic datasets, allowing for full control of the task's difficulty and more detailed insights into model performance.

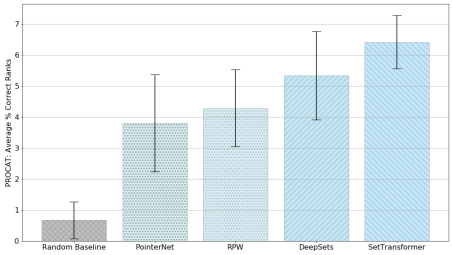


Figure 5: Comparison of the average percentage of correctly predicted ranks per input set element in the PROCAT dataset for the 4 main models and a random baseline, with error bars over 5 runs.

The fact that models which can explicitly encode higher-order interactions perform better suggests a range of future approaches. These could include: using the provided priority class information that encodes visual offer placement information; applying learn-to-rank frameworks with the set representation as the query for which offer relevance is determined; and exploring the possibility of predicting catalogues as directed graphs, particularly ones consisting of disjoint cliques guaranteeing a valid clustering [Serviansky et al., 2020].

4.2.2 Functional results on synthetic data

The results for synthetic datasets consisting of 50,000 simplified catalogue structures of lengths $n_i \in \{20, 15, 10\}$, generated following the challenging default set of clustering and structural rules, are given in the right half of table 2 as well as in tables 3 and 4. All results are averaged over 5 full training and testing runs.

The results for functional tests for reporting model performance per rule and type of rule in table 4 are of particular interest. These have been aggregated into the *clustering* score, which is the average percentage of valid sections per catalogue (based on default section rules), the *structural* score, which is the average percentage of predicted catalogues following the structural (section order) rules that do not depend on pairwise or higher order interactions between input set elements, and finally *structural*$^{2+}$, which relates to structural rules that do.

Table 4: Functional tests

	Synthetic ($n = 20$)			Synthetic ($n = 15$)		
Model	Clustering	Structural	Structural $^{2+}$	Clustering	Structural	Structural $^{2+}$
Random Baseline	0.08	0.03	0.01	0.09	0.03	0.02
Pointer Network (2015)	0.39	0.21	0.13	0.61	0.53	0.29
Read-Process-Write (2016)	0.40	0.25	0.13	0.64	0.45	0.34
DeepSets (2017)	0.43	0.35	0.16	0.75	0.61	0.37
Set Transformer (2019)	**0.63**	**0.57**	**0.32**	**0.89**	**0.88**	**0.75**

Overall, in terms of the clustering score, i.e. whether the section composition in predicted catalogues followed the rules from the synthetically generated ones, the difference in performance between methods that obtain a permutation invariant representation of the input set and those that do not was less pronounced than in terms of the two structural scores. It is unclear as to why this occurs, as both section composition and section order are defined by the composition of the input set.

Nonetheless, the model capable of explicitly encoding pairwise and higher order interactions between input set elements (4) outperforms the rest in terms of the *structural*$^{2+}$ score, predicting catalogues abiding by such structural rules in 32% of cases for $n = 20$ and 75% of cases for $n = 15$, showcasing a significant impact of set cardinality and sequence length on model performance.

4.3 Computational resources

The experiments were performed on cloud-based GPU instances provisioned from the Paperspace computing platform, with NVIDIA Quadro P6000 graphics cards (24 GB) and 8 CPU cores. Following the carbon emission calculator developed by Lacoste et al. [2019], we estimate the total CO^2 emissions for all performed experiments at 32.4 kg, and the cost of training the best performing model at 1.08 kg (over 10 hours).

Whilst the Paperspace cloud platform does not provide specific information about how much of its infrastructure's energy consumption it offsets, it is worth noting that one of the goals of solving the set-to-sequence catalog prediction task is to reduce paper waste by making physical catalogues obsolete. Thus it is hard to calculate the final impact on CO^2 emissions [Pivnenko et al., 2015].

4.4 Ethical considerations and societal impact

Given the e-commerce context of the main presented dataset, we must highlight the wider problem of *endless scroll* user interfaces in product presentation apps and social media [Lupinacci Amaral, 2020].

Whilst the PROCAT dataset is only tailored to predicting finite-length sequences from sets, we cannot rule out the possibility of extending set-to-sequence models to non-finite sets. It is also in principle possible to retrain the discussed models with additional inputs in the form of e.g. embedded personal preferences, making the predicted catalogs tailored to specific individuals, which has been linked to mental health issues related to smartphone addiction [Noë et al., 2019].

In an effort to mitigate this risk, we did not include any user interaction information; doing so could indicate the performance of individual catalogues in terms of user engagement. This information was excluded despite it being likely to signal optimal catalogue structures, as indicated by case studies in the field of ML classification [Ferrari et al., 2020] and clinical decision support [Chen et al., 2020]. As a consequence, the dataset contains no personal information and is GDPR-compliant.

We do not see any clear way for it to exacerbate bias against people of a certain gender, race, sexuality, or who have other protected characteristics. However, it may not be without merit to consider bias that may have been inherent to the marketing decisions made by people who have designed the catalogues contained in the dataset, such as the *pink tax* [Stevens and Shanahan, 2017].

5 Limitations

The PROCAT dataset consists of text in Danish, which has only six million users. However, this can also be seen as a benefit in terms of providing domain-specific, publicly available resources for a

non-privileged language [Kirkedal et al., 2019]. The catalogue ordering problem is independent of language, so we consider this limitation to be of low impact.

An important limitation of PROCAT and learning from human-made product catalogues in general, is that we only have access to one canonical ordering of the offer instances, whereas it is not impossible that other, equally valid catalogues can be constructed from the same input set of offers. In order to mitigate this, we provide the synthetic dataset library, where many valid permutations are available for each input set, increasing the signal to noise ratio.

The benchmark methods provided with PROCAT take a single-step approach. It is not currently clear whether a single step approach to predicting the product catalogue structure in a set-to-sequence formulation is the most viable. Other, multi-stage approaches might circumvent the problem of handling the padding used in the presented version of PROCAT, increasing the signal-to-noise ratio in the dataset. It is possible to use the currently provided raw data for other formulations of the underlying task.

6 Conclusion

We have highlighted the need for and provided a publicly available, easily accessible and reliably maintained product catalogue dataset. The value of the dataset stems from the difficulty of the structure prediction task, which involves representation learning, implicit clustering and permutation learning challenge. This motivates experiments with models capable of predicting complex structures as presented in sections 2.1 and 4.1.

We address the need for such a data source by curating PROCAT – a dataset of over 10,000 expert-designed product catalogues consisting of more than 1.5 million individual product offers, grouped into complementary sections. Additionally, due to the complexity of the underlying data, we also provide a library for generating simplified synthetic catalogues according to chosen clustering and structural rules. The performance of the proposed models is then measured per rule, allowing for a more fine-grained look into what our models have actually learned, through functional tests.

Benchmarks indicate that the PROCAT structure prediction task is considerably difficult. Attention-based models capable of explicitly encoding pairwise and higher order interactions between set elements outperform other set encoders and pure permutation learning models. We believe there are other interesting tasks and methods PROCAT may inspire, though an in-depth exploration is beyond the scope of this dataset paper.

We intend to improve and expand both the PROCAT dataset and the synthetic data generation library in order to facilitate the development of practical solutions in intelligent, privacy-centric product presentation systems.

Acknowledgements

This work was partly supported by an Innovation Fund Denmark research grant (number 9065-00017B) and by Tjek A/S. The authors would like to acknowledge Rasmus Pagh's assistance in model design and benchmark task conceptualization.

References

Qingyao Ai, Keping Bi, Jiafeng Guo, and W Bruce Croft. Learning a deep listwise context model for ranking refinement. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 135–144, 2018.

Steven Bird. NLTK: The natural language toolkit. In *NLTK: The natural language toolkit*, 01 2006. doi: 10.3115/1225403.1225421.

Ji Chen, Sara Chokshi, Roshini Hegde, Javier Gonzalez, Eduardo Iturrate, Yin Aphinyanaphongs, and Devin Mann. Development, implementation, and evaluation of a personalized machine learning algorithm for clinical decision support: Case study with shingles vaccination. *Journal of medical Internet research*, 22(4):e16848, 2020.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning*, 2014.

Baiyun Cui, Yingming Li, Ming Chen, and Zhongfei Zhang. Deep attentive sentence ordering network. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4340–4349, 2018.

Tiago de Melo, Altigran S da Silva, Edleno S de Moura, and Pável Calado. Opinionlink: Leveraging user opinions for product catalog enrichment. *Information Processing & Management*, 56(3): 823–843, 2019.

Anna Ferrari, Daniela Micucci, Marco Mobilio, and Paolo Napoletano. On the personalization of classification models for human activity recognition. *IEEE Access*, 8:32066–32079, 2020.

Zuohui Fu, Yikun Xian, Yaxin Zhu, Yongfeng Zhang, and Gerard de Melo. Cookie: A dataset for conversational recommendation over knowledge graphs in e-commerce. *arXiv preprint arXiv:2008.09237*, 2020.

Jingzhi Guo. Collaborative conceptualisation: towards a conceptual foundation of interoperable electronic product catalogue system design. *Enterprise Information Systems*, 3(1):59–94, 2009.

Tanjim Ul Haque, Nudrat Nawal Saber, and Faisal Muhammad Shah. Sentiment analysis on large scale amazon product reviews. In *2018 IEEE international conference on innovative research and development (ICIRD)*, pages 1–6. IEEE, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Mateusz Jurewicz and Leon Derczynski. Set-to-sequence methods in machine learning: a review. *arXiv preprint arXiv:2103.09656*, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

Andreas Kirkedal, Barbara Plank, Leon Derczynski, and Natalie Schluter. The lacunae of danish natural language processing. In *Proceedings of the 22nd Nordic Conference on Computational Linguistics*, pages 356–362, 2019.

Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753. PMLR, 2019.

Yiu-Chang Lin, Pradipto Das, Andrew Trotman, and Surya Kallumadi. A dataset and baselines for e-commerce product categorization. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*, ICTIR '19, page 213–216, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450368810. doi: 10.1145/3341981.3344237. URL `https://doi.org/10.1145/3341981.3344237`.

Manzhi Liu, Shuai Tan, Mengya Zhang, Gang He, Zhizhi Chen, Zhiwei Fu, and Changjin Luan. Waste paper recycling decision system based on material flow analysis and life cycle assessment: a case study of waste paper recycling from china. *Journal of environmental management*, 255: 109859, 2020.

Lajanugen Logeswaran, Honglak Lee, and Dragomir Radev. Sentence ordering and coherence modeling using recurrent neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Ludmila Lupinacci Amaral. 'absentmindedly scrolling through nothing': liveness and compulsory continuous connectedness in social media. *Media, Culture & Society*, 43:016344372093945, 07 2020. doi: 10.1177/0163443720939454.

Beryl Noë, Liam D Turner, David EJ Linden, Stuart M Allen, Bjorn Winkens, and Roger M Whitaker. Identifying indicators of smartphone addiction through user-app interaction. *Computers in human behavior*, 99:56–65, 2019.

Jonathan W Palmer. Retailing on the www: The use of electronic product catalogs. *Electronic Markets*, 7(3):6–9, 1997.

Liang Pang, Jun Xu, Qingyao Ai, Yanyan Lan, Xueqi Cheng, and Jirong Wen. Setrank: Learning a permutation-invariant ranking model for information retrieval. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 499–508, 2020.

Devi Parikh, Adriana Kovashka, Amar Parkash, and Kristen Grauman. Relative attributes for enhanced human-machine communication. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, 2012.

Kostyantyn Pivnenko, Eva Eriksson, and Thomas Astrup. Waste paper for recycling: Overview and identification of potentially critical substances. *Waste management (New York, N.Y.)*, 45, 03 2015. doi: 10.1016/j.wasman.2015.02.028.

Rodrigo Santa Cruz, Basura Fernando, Anoop Cherian, and Stephen Gould. Visual permutation learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(12):3100–3114, 2018.

Mike Schuster and Kuldip Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45:2673 – 2681, 12 1997. doi: 10.1109/78.650093.

Hadar Serviansky, Nimrod Segol, Jonathan Shlomi, Kyle Cranmer, Eilam Gross, Haggai Maron, and Yaron Lipman. Set2graph: Learning graphs from sets. *Advances in Neural Information Processing Systems*, 33, 2020.

Grace S Shieh. A weighted kendall's tau statistic. *Statistics & probability letters*, 39(1):17–24, 1998.

Katarina Stanoevska-Slabeva and Beat Schmid. Internet electronic product catalogs: an approach beyond simple keywords and multimedia. *Computer Networks*, 32(6):701–715, 2000.

Jennifer L Stevens and Kevin J Shanahan. Structured abstract: Anger, willingness, or clueless? understanding why women pay a pink tax on the products they consume. In *Creating Marketing Magic and Innovative Future Marketing Trends*, pages 571–575. Springer, 2017.

Zhiqing Sun, Jian Tang, Pan Du, Zhi Hong Deng, and Jian Yun Nie. DivGraphPointer: A graph pointer network for extracting diverse keyphrases. In *SIGIR 2019 - Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 755–764, 2019. doi: 10.1145/3331184.3331219.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning based on team reward. *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 3: 2085–2087, 2018. ISSN 15582914.

Nicolas Szilas, Sergio Estupiñán, Monika Marano, and Urs Richle. The study of narrative acts with and for digital media. *Digital Scholarship in the Humanities*, 35(4):904–920, 2020.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, volume 2015-January, pages 2692–2700, 2015.

Oriol Vinyals, S. Bengio, and M. Kudlur. Order matters: Sequence to sequence for sets. *CoRR*, abs/1511.06391, 2016.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, and Mathieu. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019. ISSN 14764687. doi: 10.1038/s41586-019-1724-z. URL `http://dx.doi.org/10.1038/s41586-019-1724-z`.

Tianming Wang and Xiaojun Wan. Hierarchical Attention Networks for Sentence Ordering. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:7184–7191, 2019. ISSN 2159-5399. doi: 10.1609/aaai.v33i01.33017184.

Simone Wirtz-Brückner and Eva-Maria Jakobs. Product catalogs in the face of digitalization. In *2018 IEEE International Professional Communication Conference (ProComm)*, pages 98–106. IEEE, 2018.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Yunjie Calvin Xu, Shun Cai, and Hee-Woong Kim. Cue consistency and page value perception: Implications for web-based catalog design. *Information & Management*, 50(1):33–42, 2013.

Yongjing Yin, Fandong Meng, Jinsong Su, Yubin Ge, Lingeng Song, Jie Zhou, and Jiebo Luo. Enhancing pointer network for sentence ordering with pairwise ordering predictions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9482–9489, 2020.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf`.

Yan Zhang, Adam Prügel-Bennett, and Jonathon Hare. Learning representations of sets through optimized permutations. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

SUPPLEMENTARY MATERIALS
PROCAT: Product Catalogue Dataset for Implicit Clustering, Permutation Learning and Structure Prediction

Mateusz Jurewicz*
Department of Computer Science
IT University of Copenhagen
København, 2300
maju@itu.dk

Leon Derczynski
Department of Computer Science
IT University of Copenhagen
København, 2300
leod@itu.dk

A Appendix

The appendix includes supplementary information including links to the dataset and the code repository for repeated experiments in subsection A.1, as well as the detailed dataset documentation and intended uses in the form of a datasheets for datasets available in subsection A.3.

A.1 Supplementary information and links

The URL to access the dataset is provided below:

`https://doi.org/10.6084/m9.figshare.14709507`

The obtained persistent dereferencable identifier (DOI minted by the data repository) is therefore: 10.6084/m9.figshare.14709507.

Authors bear all responsibility in case of violation of rights. The data is made publicly available under the Attribution-NonCommercial-ShareAlike 4.0 International license (CC BY-NC-SA 4.0). The dataset should not be used for commercial purposes.

Hosting is performed by FigShare, the authors are responsible for maintaining the dataset.

All explanations on how to read the dataset, with examples, is provided via jupyter notebooks as part of the code repository for repeated experiments:

`https://github.com/mateuszjurewicz/procat`

Additionally, the best performing model is made available with the DOI 10.5281/zenodo.4896303 at this hosting address:

`https://zenodo.org/record/4896303#.YLnxgZMzbOQ`

The dataset is intended to be publicly available forever, hence it was uploaded to the FigShare data repository, which also handles its discoverability through structured metadata. For more information, see:

`https://knowledge.figshare.com/publisher/fair-figshare`

A.2 Further notes on dataset diversity

The diversity of the dataset is limited due to the offer text being in Danish. Our intention was to provide a valuable resource for an underrepresented language. One important aspect of the dataset is

*Affiliated with the Tjek A/S Machine Learning Department (København, 1408), contact via mj@tjek.com.

35th Conference on Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks.

Table 1: Global Product Classification of PROCAT Catalogues

Category	Number of Catalogues	%
Food (FBT)	7,456	67.40%
Electronic	5,231	47.28%
Personal Care	5,113	46.22%
Tools	3,311	29.93%
Sports Equipment	2,147	19.41%
Lawn/Garden Supplies	2,039	18.43%
Home Appliances	2,028	18.33%
Baby Care	1,986	17.95%
Household Furniture	1,672	15.11%
Pet Care	1,522	13.76%
Footwear	1,324	11.97%
Toys and Games	1,293	11.69%
Fuels	548	4.95%

that the catalogues come from a wide variety of providers, including cross-border shops that have a significant following in neighboring Scandinavian countries, particularly Sweden and Norway, as well as Germany.

We also provide an overview of commercial categories that the catalogues belong to, following the Global Product Classification (GPC-GS1), with multiple categories per catalogue, in table 1.

Finally, the number of individual retailers that the catalogues belonged to is approximately 2,400 and the total number of unique users who have viewed the catalogues within the app is approximately 2.5 million. Our hope is to represent a broad array of product categories and providers.

A.3 Datasheets for Datasets

The following includes answers to all the questions from the suggested datasheets for datasets framework.

1. **Motivation**
   (a) **For what purpose was the dataset created?**
   The dataset in its current form was created with the purpose of helping solve an industrial challenge of optimal catalogue structure prediction.
   (b) **Who created the dataset (e.g., which team, research group) and on behalf of which entity (e.g., company, institution, organization)?**
   Original raw data collection was performed as part of the day-to-day operations of the company Tjek A/S, which aggregates product catalogues for viewing in a digital format. The curation and preprocessing was performed by the authors of this paper.
   (c) **Who funded the creation of the dataset?**
   The research is funded through an Innovation Fund Denmark research grant that Tjek A/S is a beneficiary of (grant number 9065-00017B).
2. **Composition**
   (a) **What do the instances that comprise the dataset represent (e.g., documents, photos, people, countries)?**
   The instances represent 3 types of entities. The most atomic entity is an *offer*, which represents a specific product with a text heading and description, which often includes its on-offer price. Individual product offers are then grouped into *sections*, which represent pages in a physical catalogue brochure. Finally, an ordered list of sections comprise a single *catalogue*, for which a prediction about its optimal structure is made. This takes the form of permuting the input set of offers into an ordered list, with section breaks marking the start and end of a section.
   (b) **How many instances are there in total (of each type, if appropriate)?**
   The dataset consists of just over 10 thousand catalogs (11063), almost a quarter of a million sections (238256) and over 1.5 million offers (1613686). These are further

grouped into a suggested 80/20 train and test split, with 8850 catalogs in the train set and 2212 in the test set.

(c) **Does the dataset contain all possible instances or is it a sample (not necessarily random) of instances from a larger set?**
The dataset is not a sample, it contains all catalogue instances from the years 2015 - 2019 available for viewing in the Tjek A/S app. No other selection filter was used.

(d) **What data does each instance consist of?**
Each instance consists of both raw data and pre-processed features.
Each offer instance consists of its unique id, its related section and catalogue ids, a text heading and description in both raw form and as word tokens using the nltk tokenizer, the total token count, and finally the full offer text as a vector referencing a vocabulary of 300 thousand word tokens. Additionally, each offer is categorized into a priority class, representing how visually prominent it was in the original catalogue in terms of relative image size (on a 1-3 integer scale).
Each catalogue instance consists of its unique id, an ordered list of associated section ids, and an ordered list of offer ids that comprise the catalogue in question, including section break markers. Additionally, each catalogue instance also includes information in the form of ordered lists of offers as vectors, grouped into sections, their corresponding priority class and the catalogue's total number of offers. Finally a shuffled x of offer vectors (with section breaks) is provided for each catalogue, along with the target y representing the permutation required to restore the original order.

(e) **Is there a label or target associated with each instance?**
Yes, each catalogue instance is pre-processed into a shuffled x of offer vectors and section break markers, along with the target y representing the permutation required to restore the human-designed structure of the original catalogue.

(f) **Is any information missing from individual instances?**
No data is missing.

(g) **Are relationships between individual instances made explicit (e.g., users' movie ratings, social network links)?**
Yes, every offer instance is tied to its section and catalogue via their ids in the appropriate columns of the provided comma-separated files.

(h) **Are there recommended data splits (e.g., training, development/validation, testing)?**
Yes, the entire catalogue set is grouped into a suggested 80/20 train and test split, with 8850 catalogs in the train set and 2212 in the test set. Catalogues were assigned to each group randomly. A validation set can be extracted from the train set based on each researcher's individual preference.

(i) **Are there any errors, sources of noise, or redundancies in the dataset?**
There are no known errors, sources of noise or redundancies in the dataset, however there is a possibility of some degree of overlap between individual offers in terms of the underlying product.

(j) **Is the dataset self-contained, or does it link to or otherwise rely on external resources (e.g., websites, tweets, other datasets)?**
The dataset is self-contained.

(k) **Does the dataset contain data that might be considered confidential (e.g., data that is protected by legal privilege or by doctor patient confidentiality, data that includes the content of individuals' non-public communications)?**
The dataset does not contain data that might be considered confidential.

(l) **Does the dataset contain data that, if viewed directly, might be offensive, insulting, threatening, or might otherwise cause anxiety?**
The dataset does not contain data that the authors would consider offensive, insulting, threatening or causing anxiety.

(m) **Does the dataset relate to people?**
The dataset does not relate to people (thus skipping the remainder of this section's questions).

3. **Collection Process**

(a) **How was the data associated with each instance acquired?**
The data was acquired through a combination of feed readers and custom scraping scripts developed by Tjek A/S. For further details, see the answer to the next question.

(b) **What mechanisms or procedures were used to collect the data (e.g., hardware apparatus or sensor, manual human curation, software program, software API)?**
The scripts read the feeds and scrape a list of stores and PDF catalogs associated with said stores. This provides the basic tooling and processing of the data and communicates this to the company's core API, running the scrapers on a defined schedule as well as on-demand. Following that, a human curation step is performed by the operations department to make sure the obtained data is correct. The data is directly observable.

(c) **If the dataset is a sample from a larger set, what was the sampling strategy (e.g., deterministic, probabilistic with specific sampling probabilities)?**
The dataset is not a sample.

(d) **Who was involved in the data collection process (e.g., students, crowdworkers, contractors) and how were they compensated (e.g., how much were crowdworkers paid)?**
The data collection process was done as part of the day-to-day operations of Tjek A/S, by properly compensated full-time employees.

(e) **Over what timeframe was the data collected?**
The data was collected within the full 4 year period between 2015 and 2019.

(f) **Were any ethical review processes conducted (e.g., by an institutional review board)?**
No.

(g) **Does the dataset relate to people?**
No, thus skipping the remainder of the questions in this section.

4. **Preprocessing / cleaning / labeling**

(a) **Was any preprocessing/cleaning/labeling of the data done (e.g., discretization or bucketing, tokenization, part-of-speech tagging, SIFT feature extraction, removal of instances, processing of missing values)?**
Yes, the raw text features of each offer instance were tokenized using the nltk tokenizer, a vocabulary of word tokens was limited to 300 thousand words and used to obtain offer vectors. Each offer instance was truncated or padded to 30 word tokens, with over 75% of offers consisting of fewer than 24 tokens. Each catalogue instance was truncated or padded to 200 offer instances, with over 75% of catalogues consisting of fewer than 163 offers.
Additionally, to obtain the prominence class per offer per section, signifying the relative size of the offer's image on the page, a proprietary algorithm was used.

(b) **Was the "raw" data saved in addition to the preprocessed/cleaned/labeled data (e.g., to support unanticipated future uses)?**
Yes, raw data is also provided.

(c) **Is the software used to preprocess/clean/label the instances available?**
Yes, the nltk library is available under the Apache License 2.0.

5. **Uses**

(a) **Has the dataset been used for any tasks already?**
The dataset is actively being used to help predict the optimal structure of product catalogues given a provided set of offers, based on their textual description and to recommend complementary offers. It has not been used in prior research.

(b) **Is there a repository that links to any or all papers or systems that use the dataset?**
The repository containing the scripts for repeated experiments will include links to any and all papers using this dataset. For more information, see the appendix subsection A.1.

(c) **What (other) tasks could the dataset be used for?**
The dataset can be used for representation learning through the co-occurrence of offers within the same section, leading to a complementariness-based recommendation

system. It can also be used for learning to cluster a set of offers into a variable number of sections, which is an implicit step in the main task of predicting the entire structure of a catalogue through permutation learning (as it includes the section break markers).

(d) **Is there anything about the composition of the dataset or the way it was collected and preprocessed/cleaned/labeled that might impact future uses?**
It is important to remember that the provided catalogues represent the Danish market between 2015-2019, and thus might not represent patterns that will hold in other societies. This, however, has no bearing on demonstrating a machine learning model's ability to learn structure through joint clustering and permutation learning, which is the intended use of the dataset.

(e) **Are there tasks for which the dataset should not be used?**
The dataset is not meant to be used as a representation of the market for any form of trend prediction.

6. **Distribution**

(a) **Will the dataset be distributed to third parties outside of the entity (e.g., company, institution, organization) on behalf of which the dataset was created?**
The dataset will be made publicly available under the chosen license to any and all parties. For more information see the appendix subsection A.1.

(b) **How will the dataset will be distributed (e.g., tarball on website, API, GitHub)? Does the dataset have a digital object identifier (DOI)?**
The dataset is distributed through a dataset hosting service and has a DOI, for details see the appendix subsection A.1.

(c) **When will the dataset be distributed?**
The dataset will be distributed by the time of the paper's submission.

(d) **Will the dataset be distributed under a copyright or other intellectual property (IP) license, and/or under applicable terms of use (ToU)?**
The dataset will be distributed under the Attribution-NonCommercial-ShareAlike 4.0 International license (CC BY-NC-SA 4.0). The dataset should not be used for commercial purposes.

(e) **Have any third parties imposed IP-based or other restrictions on the data associated with the instances?**
No.

(f) **Do any export controls or other regulatory restrictions apply to the dataset or to individual instances?**
No.

7. **Maintenance**

(a) **Who is supporting/hosting/maintaining the dataset?**
The dataset is hosted by *figshare*, an open access repository where researchers can preserve and share their research outputs, including figures, datasets, images and videos. It is supported by Digital Science & Research Solutions Ltd.
It is maintained by the authors of this paper.

(b) **How can the owner/curator/manager of the dataset be contacted (e.g., email address)?**
Via the emails provided in the contact information above the abstract, repeated here for convenience: maju@itu.dk; leod@itu.dk.

(c) **Is there an erratum?**
There is currently no erratum, it will be added to both the main sharing link and the github repository containing the code for repeated experiments should the need to create an erratum occur.

(d) **Will the dataset be updated (e.g., to correct labeling errors, add new instances, delete instances)?**
If labeling errors are found, they will be corrected. The dataset may be expanded with further instances, depending on the academic interest and number of downloads.

(e) **If the dataset relates to people, are there applicable limits on the retention of the data associated with the instances (e.g., were individuals in question told that their data would be retained for a fixed period of time and then deleted)?**
The dataset does not relate to people.

(f) **Will older versions of the dataset continue to be supported/hosted/maintained?**
Yes, all previous versions of the dataset will continue to be available.

(g) **If others want to extend/augment/build on/contribute to the dataset, is there a mechanism for them to do so?**
Others are encouraged to extend the dataset and can choose to either do so in cooperation with the authors of this paper after contacting them via the provided email addresses or individually in accordance with the chosen license.

Chapter 6

Article 3

Article details:

3. **"Set Interdependence Transformer: Set-to-Sequence Neural Networks for Permutation Learning and Structure Prediction"**
   Mateusz Jurewicz and Leon Derczynski
   Proceedings of the 31st International Joint Conference on Artificial Intelligence, IJCAI-ECAI 2022.

Context. The third published article focuses on experimental work on a number of set-to-sequence tasks. The intent behind it was to highlight the importance and ubiquity of tasks where input takes the form of a varying-cardinality set of elements and the output is some form of order among those elements. Whilst the work does not constitute a complete approach to the Catalog Problem, it did underline the impact of set encoding methods - of which the titular Set Interdependence Transformer (SIT) is an example. As one module of a larger proposed set-to-sequence architecture, SIT's performance also helped showcase the importance of being able to learn relational rules dependent on pairwise and higher-order interactions.

Contributions. Within this IJCAI-ECAI publication, a slight adjustment to the popular Set Transformer model is proposed, which is empirically shown to lend itself well to tasks requiring the learning of interactions between input set elements (as introduced in Chapter 3.2). The developed Set Interdependence Transformer is

then included as the set encoding module within larger set-to-sequence architectures and their performance is tested on a variety of challenges, ranging from the TSP, through simple formal grammars to a sentence ordering dataset, synthetic catalogs and PROCAT. Additionally, a minor ablation study is provided to test whether a smaller number of SIT layers can outperform other set encoding methods paired with the same permutation learning module, when tasked with predicting synthetic catalogs dependent on interactions of an order higher than the number of those layers.

Recent Developments. Given the small amount of time that has passed since this article was published, there isn't a large number of new and relevant research that builds on or provides a new perspective regarding the work presented below. However, an interesting, GNN-based approach to encoding the relations among partial-view point clouds in the context of multi-object manipulation for robotic arms has been proposed by Huang *et al.* (2022), with the goal of empowering the robot to interact with multiple objects at the same time. Whilst the application of their Relational Dynamics GNN (RD-GNN) also shows good empirical results, similarly attributed by the authors to the relational inductive bias of their network and latent space dynamics, a theoretical investigation of the underlying reason for this performance improvement was also left for future work.

6.1 Set Interdependence Transformer: Set-to-Sequence Neural Networks for Permutation Learning and Structure Prediction

In this section, the third published article is included in unchanged form, starting on the next page for ease of reading.

Set Interdependence Transformer: Set-to-Sequence Neural Networks for Permutation Learning and Structure Prediction

Mateusz Jurewicz[1,2] and **Leon Derczynski**[1]
[1] IT University of Copenhagen
[2] Tjek A/S
{maju, leod}@itu.dk

Abstract

The task of learning to map an input *set* onto a permuted *sequence* of its elements is challenging for neural networks. Set-to-sequence problems occur in natural language processing, computer vision and structure prediction, where interactions between elements of large sets define the optimal output. Models must exhibit relational reasoning, handle varying cardinalities and manage combinatorial complexity. Previous attention-based methods require n layers of their set transformations to explicitly represent n-th order relations. Our aim is to enhance their ability to efficiently model higher-order interactions through an additional interdependence component. We propose a novel neural set encoding method called the Set Interdependence Transformer[1], capable of relating the set's permutation invariant representation to its elements within sets of any cardinality. We combine it with a permutation learning module into a complete, 3-part set-to-sequence model and demonstrate its state-of-the-art performance on a number of tasks. These range from combinatorial optimization problems, through permutation learning challenges on both synthetic and established NLP datasets for sentence ordering, to a novel domain of product catalog structure prediction. Additionally, the network's ability to generalize to unseen sequence lengths is investigated and a comparative empirical analysis of the existing methods' ability to learn higher-order interactions is provided.

1 Introduction

There is a wide range of challenges where the objective is to find an optimal mapping from an unordered collection of objects to a permutation. This group of *set-to-sequence* tasks covers combinatorial optimization and structure prediction problems where exhaustive search is often not tractable, lending itself to neural network (NN) approaches.

Set-to-sequence challenges arise in many areas of application. Examples include natural language processing, in the form of sentence and paragraph ordering [Wang and Wan, 2019; Pandey and Chowdary, 2020], text comprehension [Li and Gao, 2020] and discourse coherence maximization [Farag, 2021]; computer vision for relative attribute learning [Santa Cruz and Fernando, 2017] and rigid point cloud registration [Yew and Lee, 2020]; reinforcement learning for managing the combinatorial action space of an agent [Vinyals *et al.*, 2019]. For an overview, see Jurewicz and Derczynski (2021b). We adapt our method to a novel application domain in the form of predicting the structure of digital catalogs.

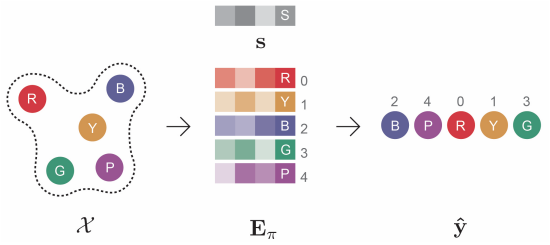


Figure 1: In a set-to-sequence task the initial set ($\mathcal{X}$) is passed to a set encoder, which obtains per-element representations ($\mathbf{E}_\pi$) and a permutation invariant representation ($\mathbf{s}$) of the entire set. A permutation decoder uses them to order the elements into a sequence ($\hat{\mathbf{y}}$).

Set-to-sequence models can be thought of as consisting of two distinct parts (see figure 1). Firstly, the set encoder obtains representations of both the elements of a set individually and of the set in its entirety. Secondly, a permutation learning module uses these two representations to predict a reordered sequence. Each stage presents unique challenges. Set-input methods are required to handle inputs of any dimensionality due to the examples being sets of varying cardinalities [Lee and Lee, 2019]. Additionally, the learned representation of a given set of cardinality n must be identical for all $n!$ possible permutations of the vector that represents it. This permutation invariance can be achieved through a variety of symmetric functions, from simple summation [Zaheer *et al.*, 2017], through self-attention [Lee and Lee, 2019] to the output of a bipartite matching algorithm [Skianis and Konstantinos, 2020]. Solving challenging set-to-sequence problems requires a degree of relational reasoning, thus these functions benefit from being capable of encoding higher-order interac-

[1]Paper accepted for publication in the IJCAI-22 proceedings

tions between set elements [Huang *et al.*, 2020].

Predicting a permutation is also a challenge by itself. One of the main difficulties in dealing with combinatorial objects is that the number of possible output sequences increases factorially in the cardinality of the set. Additionally, the space of all possible permutations is not smooth, preventing direct use of gradient-based methods without a relaxation of the concept [Diallo *et al.*, 2020]. Finally, when the target is a complex structure represented by a permutation, it can be difficult to obtain evaluation and loss functions that are only sensitive to meaningful alterations of this structure.

To address these challenges, we propose a novel set encoding method which, unlike its predecessors, jointly learns the permutation invariant representation of the entire set and the permutation equivariant representations of individual set elements. Whilst methods that rely entirely on a pooled representation of the set have been applied to set-to-sequence problems [Vinyals *et al.*, 2016], their performance drops sharply as the cardinality of the input set increases, compared to methods that obtain both types of representations [Wang and Wan, 2019; Yin *et al.*, 2020]. To our knowledge, no other NN set encoding method learns the representations of set elements and the entire set jointly through an adjusted attention mechanism. Instead, they obtain the permutation equivariant representations of elements and then pool them through various symmetric operations to derive the encoding of the set proper. In this paper, we show empirically that transforming them jointly is advantageous for the purposes of permutation learning and structure prediction.

We propose a complete, 3-part neural network architecture designed for performing the set-to-sequence mapping on inputs of any cardinality, consisting of an initial set encoder, an interdependence encoder and a permutation module. We then showcase its usefulness on a number of challenges, ranging from toy problems such as the Travelling Salesman Problem, through learning context-free and context-sensitive grammars to robust tasks such as sentence ordering and the novel task of catalog structure prediction. We also demonstrate our model's ability to generalize to unseen lengths and empirically prove its ability to learn higher-order interaction rules on a number of easily customized, synthetic structure datasets and via corresponding evaluation functions. All code, hyperparameters and datasets required for repeated experiments are provided in the supplement.

Our main contributions are summarized as follows:

- A novel, fully differentiable set encoding method designed specifically for permutation learning and structure prediction challenges, capable of learning higher-order interactions within sets of any cardinality in a single layer of the proposed transformations.
- A complete set-to-sequence model outperforming state-of-the-art methods on established datasets and within the novel application domain of catalog structure prediction.
- An expanded library for generating synthetic set-to-sequence structure datasets. In addition, we provide easy-to-use tools for obtaining detailed performance reports through customizable metrics, which allow researchers to measure and empirically confirm a model's relational reasoning capabilities.

2 Model

The goal is to transform an input set of any cardinality into the permuted sequence of its elements. To do this, the proposed set-to-sequence model consists of three core components: (i) a basic set encoder, (ii) a novel *interdependence encoder* and (iii) a permutation decoder. The initial encoder uses a learned pooling function to obtain (a) the permutation equivariant representations of individual set elements, and subsequently (b) the permutation invariant representation of the entire set. These two representations are then transformed together in the interdependence encoder, such that higher order interactions between both individual set elements and the set in its entirety can be learned in a single step. Finally, the permutation decoder sequentially selects the elements to form the output sequence by using these two representation via an enhanced pointer attention mechanism. An overview of our complete interdependence architecture is shown in Figure 2.

From a formal standpoint, the model is given a set $\mathcal{X}$ of any cardinality n, consisting of individual set elements represented by fixed-length vectors $\mathbf{x}_i$ of dimensionality d, in the form of a matrix $\mathbf{X}_\pi$ arbitrarily ordered according to some permutation π:

$$\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \sim \mathbf{X}_\pi \in \mathbb{R}^{n \times d} \tag{1}$$

The task is then to sequentially select individual set elements $\mathcal{X}$ in the target order, represented by a vector of indices $\mathbf{y} \in \mathbb{N}^{1 \times n}$ referencing the original placement of set elements in the $\mathbf{X}_\pi$ matrix. This process continues until there are no remaining candidate elements.

2.1 Basic Set Encoder

The initial encoder necessarily consists of learned function f_e for transforming set elements in a permutation equivariant way, and learned function f_s for pooling those element representations ($\mathbf{E}_\pi$) into a permutation invariant embedding of the entire set ($\mathbf{s}$), such that:

$$f_e(\mathbf{X}_\pi) = (\mathbf{e}_{\pi(1)}, \ldots, \mathbf{e}_{\pi(n)}) = \mathbf{E}_\pi \tag{2}$$

$$\forall \pi \in \Pi \; ((f_s \circ f_e)(\mathbf{X}_\pi) = \mathbf{s}) \tag{3}$$

In our proposed model this basic set encoder takes the form of a simplified Set Transformer [Lee and Lee, 2019], which we chose for its ability to explicitly represent inter-item interactions. We denote initial transformer-style attention toward permutation equivariant element representations $\mathbf{E}_\pi$ as:

$$\text{Att}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}})\mathbf{V} \tag{4}$$

where $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ are the projected query, key and value matrices obtained from $\mathbf{X}_\pi$ via weight matrices $\mathbf{W}^Q$, $\mathbf{W}^K$ and $\mathbf{W}^V$; $\sqrt{d_k}$ is the standard transformer normalizing factor.

This operation is repeated in a multi-head fashion for each of the m heads, whose outputs are concatenated and further transformed via learned weight matrix $\mathbf{W}^O$, without positional encoding or dropout:

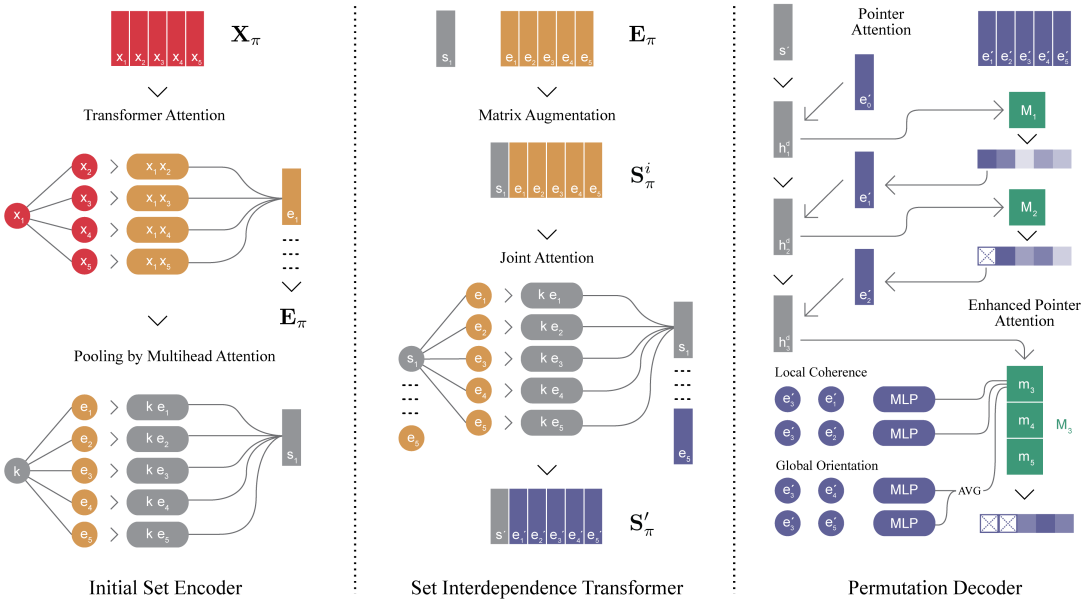


Figure 2: Three stages of the proposed set-to-sequence model. The Initial Set Encoder (left) obtains a permutation equivariant element representation ($\mathbf{E}_\pi$) through transformer-style attention. It then learns the permutation invariant encoding ($\mathbf{s}$) of the entire set via Pooling by Multihead Attention. The Set Interdependence Transformer (middle) augments the per-element matrix with the set encoding into $\mathbf{S}_\pi$ and performs further self-attention transformations, allowing for higher order interactions being modelled between set-elements and the set in its entirety. Finally, a Permutation Decoder (right) is used to sequentially select the next element in the output sequence until there are none left.

$$\mathbf{H}_i = \text{Att}(\mathbf{X}_\pi \mathbf{W}_i^Q, \mathbf{X}_\pi \mathbf{W}_i^K, \mathbf{X}_\pi \mathbf{W}_i^V) \tag{5}$$

$$\mathbf{E}_\pi = \text{Concat}(\mathbf{H}_1, \ldots, \mathbf{H}_m)\mathbf{W}^O \tag{6}$$

To obtain a permutation invariant representation ($\mathbf{s}$) of the set we apply simplified Pooling by Multihead Attention [Lee and Lee, 2019], which performs an attention transformation between a learned seed vector $\mathbf{k} \in \mathbb{R}^{1\times d}$ as the query and $\mathbf{E}_\pi$ as the key and value for each of the m heads:

$$\mathbf{s}_j = \text{Att}(\mathbf{k}_j \mathbf{W}_j^Q, \mathbf{E}_\pi \mathbf{W}_j^K, \mathbf{E}_\pi \mathbf{W}_j^V) \tag{7}$$

$$\text{PMA}(\mathbf{k}, \mathbf{E}_\pi) = \text{Concat}(\mathbf{s}_1, \ldots, \mathbf{s}_m)\mathbf{W}_s^O = \mathbf{s} \tag{8}$$

2.2 Set Interdependence Transformer

At this stage, we could rely on the transformations that lead to $\mathbf{E}_\pi$ to encode the dependencies between elements of $\mathcal{X}$. However, a single layer of such transformations can only explicitly capture *pairwise* relations, as it computes attention between pairs of elements [Lee and Lee, 2019]. Thus it would require up to n stacks to explicitly encode higher-order interactions among an entire set of cardinality n. Our proposed set encoder allows efficient capture of dependencies between set elements and the set in its entirety through adjusted transformer-style attention. We refer to it as the Set Interdependence Transformer, or **SIT**.

The SIT performs attention-based transformations between both individual set elements and the permutation-invariant representation of the set itself in the form of an augmented matrix $\mathbf{S}_\pi$. The initial permutation invariant representation of the entire set $\mathbf{s}$ is treated as an element of a new set:

$$\mathbf{S}_\pi^i = (\mathbf{E}_\pi | \, \mathbf{s}) \tag{9}$$

$$\text{SIT}_i(\mathbf{s}, \mathbf{E}_\pi) = \sigma\left(\frac{(\mathbf{S}_\pi^i \mathbf{W}_i^Q)\,(\mathbf{S}_\pi^i \mathbf{W}_i^K)^\top}{\sqrt{d_\mathbf{s}}}\right)\mathbf{S}_\pi^i \mathbf{W}_i^V \tag{10}$$

$$\text{SIT}_i(\mathbf{s}, \mathbf{E}_\pi) = \mathbf{S}'_\pi = (\mathbf{E}'_\pi | \, \mathbf{s}') \tag{11}$$

where $d_\mathbf{s}$ is a scaling factor equal to the length of the permutation invariant vector representing the entire set, σ is a softmax non-linear function and $\mathbf{W}$ matrices are learned parameters. The above operation can be repeated over any number of heads as described under the inital basic encoder (subsection 2.1).

This method enables encoding of dependencies between individual set elements and the set in its entirety. Even a single SIT layer is capable of modelling higher-order interactions. This is beneficial for tasks over large sets, such as point cloud challenges, where the cardinality is prohibitively high for pairwise transformations.

To obtain the separate set and set-element representations required by a permutation module, we reverse the augmentation and retrieve the transformed $\mathbf{s}'$ vector by its index. Note

that the required permutation invariance of $\mathbf{s}'$ is maintained during the permutation equivariant SIT_i transformations.

2.3 Permutation Decoder

Finally, we use the $\mathbf{E}'_\pi$ and $\mathbf{s}'$ representations in a permutation decoder employing enhanced pointer-style attention [Yin *et al.*, 2020]. This takes the form of an LSTM-based pointer network with two additional mechanisms for pairwise ordering predictions towards improved global and local coherence of the output sequence. Formally, we calculate the conditional probability of a predicted order $\hat{\mathbf{y}}$ as:

$$p_\theta(\hat{\mathbf{y}} \mid \mathcal{X}) = \prod_{i=1}^{n} p_\theta(\hat{y}_i \mid \hat{\mathbf{y}}_{<i}, \mathbf{E}'_\pi, \mathbf{s}') \tag{12}$$

$$p_\theta(\hat{y}_i \mid \hat{\mathbf{y}}_{<i}, \mathcal{X}) = \text{softmax}(\mathbf{v}^\top \tanh(\mathbf{W}_1 \mathbf{h}_i^d + \mathbf{W}_2 \mathbf{M}_i)) \tag{13}$$

$$\mathbf{h}_i^d = \text{LSTM}(\mathbf{h}_{i-1}^d, \mathbf{e}'_{i-1}) \ , \ \mathbf{h}_0^d = \mathbf{s}' \tag{14}$$

where $\mathbf{v}$, $\mathbf{W}_1$ and $\mathbf{W}_2$ are model parameters, n is the set's cardinality, tanh is the hyperbolic tangent nonlinearity, $\mathbf{e}'_{i-1}$ is the embedding of the set element selected at previous step $i-1$ and $\mathbf{h}_i^d$ is the hidden state of the permutation module at current step i. The first hidden state $\mathbf{h}_0^d$ is initialized from the permutation invariant set representation $\mathbf{s}'$. The $\mathbf{M}_i$ matrix provides additional context consisting of two types of information. The first is global orientation relating all remaining unordered set elements to each other and the second is local coherence between previously selected elements and remaining candidates. This context is obtained through "history" and "future" sub-modules from $\mathbf{E}'_\pi$. The sub-modules make pairwise ordering predictions in relation to each candidate element, which are then concatenated to form $\mathbf{M}_i$. For specific implementation details, see Yin *et al.* (2020).

During training, given a batch $\mathcal{B}$ of n examples of the form $(\mathcal{X}_i, \mathbf{y}_i)$, we minimize the loss function:

$$\mathcal{L}(\theta) = -\frac{1}{n} \sum_{(\mathcal{X},\, \mathbf{y}) \in \mathcal{B}} (\log\ p_\theta(\mathbf{y}|\mathcal{X}) + \lambda\ \mathcal{L}_s) \tag{15}$$

where θ is the set of all model parameters and $\lambda = 0.1$ is a hyperparameter that balances the first term of the loss with $\mathcal{L}_s$, a cross-entropy loss of the future and history sub-modules.

3 Experiments

All datasets, hyperparameters and code are freely available and described in detail in the linked supplementary material. The provided code includes any and all data pre-processing and generation steps, where applicable. Full requirements are explicitly listed. Best performance is reported in **bold**, second best is underlined. Reported results are averaged over 3 full training runs, standard deviation is reported after the ± sign.

3.1 Setup

We compare all models on the planar symmetric Travelling Salesman Problem (TSP) [Vinyals *et al.*, 2015] and 3 types of context-free and context-sensitive grammars, as well as the ROCStory sentence ordering dataset [Mostafazadeh and Chambers, 2016] and the PROCAT product catalog dataset [Jurewicz and Derczynski, 2021a], following provided train, test and validation splits. Additionally, we expand upon the synthetic structure dataset from PROCAT and report performance per n-th order rule of interaction required.

We use Kendall's Rank Correlation Coefficient (τ) and Perfect Match Ratio (PMR) as primary metrics, scaled by a factor of a hundred for readability, following convention [Wang and Wan, 2019; Yin *et al.*, 2020; Pandey and Chowdary, 2020]. For TSP we report average tour length (shorter is better) and in both grammars and synthetic structure experiments the percentage of valid predictions per ruleset. For the representations of natural language entities in ROCStory and PROCAT we use the concatenated and averaged output of the last 4 layers of the cased large version of BERT [Devlin *et al.*, 2019], frozen during training to isolate the effect of different set encoding methods on the permutation task performance. The models' exact layer dimensions are given in the supplement, with the number of learnable parameters of each model varying by less than 5% per task. The proposed model consisted of 3-5 layers of SIT transformations over 4 attention heads, with the hidden size of 256. The AdamW [Loshchilov and Hutter, 2017] optimizer was used with weight decay coefficient 1e-2, learning rate (α) 1e-4, dropout rate of 0.1 and batch size 32, for 50 epochs. All experiments were performed on cloud-based GPU instances, with NVIDIA Quadro P6000 graphics cards (24 GB) and 8 CPU cores.

3.2 Baselines

Our proposed model is compared with state-of-the-art set encoders combined with the same permutation component:

- **DeepSets**: *Zaheer et al.* [2017] encodes set elements independently and identically via a fully connected layer and then sums them.
- **Set Transformer**: *Lee et al.* [2019] uses transformer-style attention in a learned pooling by multihead attention function.
- **AttSets**: *Yang et al.* [2020] uses convolutional weighted attention and sum pooling.
- **RepSet**: *Skianis et al.* [2020] obtains permutation invariance through bipartite graph matching.

and with several complete set-to-sequence models:

- **Pointer Network**: *Vinyals et al.* [2015] consists of an RNN set encoder and basic pointer-style attention without pairwise predictions.
- **Read-Process-Write**: *Vinyals et al.* [2016] obtains permutation invariance via an adjusted LSTM network, which performs a number of passes over the input set.
- **ATTOrderNet**: *Vinyals et al.* [2018] combines transformer-style attention for set encoding with layer normalization, average pooling and a basic pointer-style permutation module.
- **Enhanced PtrNet**: *Yin et al.* [2020] introduces the global and local context sub-modules, adding pairwise judgments to the permutation mechanism.

Method	ROCStory		PROCAT		Synthetic $\|y\| = 30$	
	PMR	Kendall's τ	PMR	Kendall's τ	PMR	Kendall's τ
DeepSets (2017)	33.81 ± 4.55	62.41 ± 4.2	15.21 ± 2.69	35.15 ± 3.8	25.61 ± 1.90	40.94 ± 2.7
Set Transformer (2019)	41.94 ± 1.29	73.15 ± 1.9	21.03 ± 0.98	42.74 ± 2.6	30.23 ± 1.86	44.71 ± 2.9
AttSets (2020)	42.51 ± 1.45	74.81 ± 2.3	19.24 ± 1.30	38.44 ± 1.7	31.48 ± 2.04	46.20 ± 3.2
RepSet (2020)	42.47 ± 1.61	73.39 ± 1.7	22.72 ± 1.19	41.30 ± 2.5	34.84 ± 1.99	47.95 ± 3.4
PointerNet (2015)	28.73 ± 3.91	59.72 ± 6.2	02.90 ± 1.17	16.85 ± 3.4	17.33 ± 3.41	32.66 ± 3.1
Read-Process-Write (2016)	20.38 ± 7.22	51.78 ± 8.5	03.54 ± 1.06	21.11 ± 2.9	21.40 ± 2.07	36.28 ± 4.3
ATTOrderNet (2018)	41.14 ± 2.10	73.02 ± 2.0	17.33 ± 2.31	37.47 ± 3.2	28.02 ± 2.40	42.67 ± 3.4
Enhanced PointerNet+ (2020)	44.32 ± 1.25	76.43 ± 1.3	22.09 ± 1.57	42.53 ± 1.9	34.34 ± 1.39	48.14 ± 2.9
Set Interdep. Transformer	**47.00 ± 0.89**	**79.86 ± 0.9**	**25.61 ± 1.81**	**46.41 ± 1.3**	**37.16 ± 1.01**	**52.03 ± 2.2**

Table 1: Precise Match Ratio (PMR) and Kendall's Rank Correlation Coefficient (τ) results for ROCStory, PROCAT and the Synthetic task.

Method	TSP cardinality		
	$n = 10$	$n = 15$*	$n = 20$*
Held–Karp	2.97	3.87	4.42
Random	4.48	7.32	8.96
DeepSets	3.27 ± 0.12	4.37 ± 0.19	5.12 ± 0.14
SetTrans	3.00 ± 0.02	3.99 ± 0.05	4.55 ± 0.08
AttSets	3.06 ± 0.06	4.02 ± 0.13	4.64 ± 0.19
RepSet	3.03 ± 0.05	4.13 ± 0.04	4.59 ± 0.11
PtrNet	3.18 ± 0.26	4.85 ± 0.69	5.83 ± 0.85
RPW	3.67 ± 0.10	4.92 ± 0.04	4.82 ± 0.11
ATTOrdNet	3.10 ± 0.06	4.53 ± 0.15	5.63 ± 0.39
PtrNet+	3.01 ± 0.04	3.92 ± 0.06	4.52 ± 0.09
SIT	**2.98 ± 0.01**	**3.90 ± 0.03**	**4.46 ± 0.06**

Table 2: Performance on the TSP in terms of average tour length.

3.3 Core Results

As shown in Table 2, our proposed model outperforms baselines on the TSP by predicting shorter average paths and with smaller standard deviation. The improvement is more pronounced as input set cardinalities increasingly differ from those seen during training, unseen-cardinality sets being marked with an asterisk. Specifically, the proposed model predicts paths that are shorter than the second-best by 0.02 and 0.06 when $n = 15$ and $n = 20$ respectively, exhibiting increased ability to generalize to unseen path node counts, even for sets of twice the cardinality.

Overall results on ROCStory and PROCAT are shown in Table 1. Our proposed model, SIT, outperforms the state-of-the-art on both datasets. PMR scores are increased by 2.68 and 2.89, and τ by 3.43 and 3.67, over second-best performances on the ROCStory and PROCAT datasets respectively.

We also compare all methods on synthetically-generated structures of length 30, following the default n-th order rulesets. Compared to the best benchmark, our method offers an improvement of 2.32 (PMR) and 1.89 (τ). Table 3 presents the average percentage of predicted sequences that adhered to the rules underlying two context sensitive and one context free grammar, in the form of the Dyck language [Yu *et al.*, 2019] consisting of $n \in [2, 100]$ pairs of {} and (). The simplest grammar ($a^n b^n c^n$, n sampled uniformly from [1..100]) was able to be fully learned by most models using the enhanced pointer network as their permutation module, our proposed method among them. With regards to the more challenging $a^n b^k c^{nk}$ context-sensitive grammar (n and k sampled uniformly from [1..25]) our method outperformed the second best by 1.24 % and by 1.56 % in the case of the Dyck language. Performance in terms of learning higher-order interaction rulesets is discussed in Section 3.4.

The main improvement of the proposed model architecture stems from the ability to relate the input elements to the set in its entirety, supplementing its capacity to encode higher-order interactions in the set representation, from which the first hidden state of the permutation decoder is initialized. Compared with the baseline set encoders and full set-to-sequence models, the combination of the SIT set encoder and the enhanced pointer network consistently outperforms on a plethora of tasks where the target permutation depends on the input set's composition. Additionally, in order to explicitly confirm the proposed model's ability to learn higher-than-pairwise interactions between set elements in a single layer of SIT transformation, we designed a comparative experiment on the expanded synthetic structure prediction task. Results are presented in Table 4 and discussed in the next section.

3.4 Relational Reasoning Study

To empirically compare a model's capacity to learn n-th order interactions between set elements, we expanded the synthetic structure library provided by Jurewicz and Derczynski (2021a). This task requires ordering a set of tokens into a target structure, defined by customizable rulesets which depend on n-th order relations between input set elements. Each benchmark model consisted of 5 stacked layers of its respective attention transformations (except the Set Transformer, which was tested with both 4 and 5 layers). The hypothesis is that even with learning only pairwise interactions per-layer, benchmark models should be able to learn the highest-order ruleset (5th). By comparison, the tested versions of our proposed set-to-sequence model had 2, 3 and 4 attention-based encoding layers, requiring them to learn higher-order inter-

Method	Formal Grammars		
	$a^n b^n c^n$	$a^n b^k c^{nk}$	Dyck
DeepSets	97.17 ± 2.9	94.52 ± 2.4	79.84 ± 2.3
SetTrans	**100.0 ± 0.0**	96.62 ± 0.8	92.16 ± 1.5
AttSets	98.24 ± 1.3	96.12 ± 1.1	92.31 ± 1.2
RepSet	**100.0 ± 0.0**	97.64 ± 0.9	93.22 ± 1.4
PtrNet	79.31 ± 5.3	75.85 ± 4.7	58.85 ± 6.7
RPW	86.41 ± 1.5	81.74 ± 1.2	61.13 ± 5.3
ATTOrdNet	97.75 ± 1.1	94.17 ± 1.0	84.31 ± 3.8
PtrNet+	**100.0 ± 0.0**	96.82 ± 0.7	93.51 ± 1.2
SIT	**100.0 ± 0.0**	**98.88 ± 0.6**	**95.07 ± 1.0**

Table 3: Results for 2 context-sensitive and 1 context-free grammar (the Dyck language). Scores are between 0-100, reflecting what proportion of predicted sequences was grammatical.

Method	n-th Order Relation Ruleset		
	$n = 3$	$n = 4$	$n = 5$
DeepSets	53.37 ± 6.1	40.12 ± 4.0	21.13 ± 0.2
SetTrans (4)	92.41 ± 1.6	92.90 ± 1.4	91.74 ± 1.7
SetTrans (5)	94.66 ± 1.9	93.38 ± 1.3	92.93 ± 1.5
AttSets	93.84 ± 2.8	92.93 ± 1.1	86.44 ± 1.3
RepSet	91.29 ± 3.4	90.84 ± 2.4	89.73 ± 2.9
PtrNet	41.26 ± 5.3	31.96 ± 4.8	15.23 ± 4.2
RPW	45.11 ± 2.0	36.31 ± 1.6	16.12 ± 2.5
ATTOrdNet	82.31 ± 4.4	67.07 ± 2.0	0.12 ± 1.4
PtrNet+	89.58 ± 3.9	87.22 ± 3.2	86.94 ± 2.8
SIT (2 layers)	93.83 ± 3.6	89.01 ± 2.6	86.79 ± 1.9
SIT (3 layers)	**98.73 ± 0.8**	93.44 ± 2.3	92.72 ± 1.8
SIT (4 layers)	98.48 ± 1.2	**97.52 ± 0.9**	**96.10 ± 1.6**

Table 4: Synthetic structure prediction scores per ruleset type, split by order of interaction required. On a scale of 0-100, reflecting the proportion of valid predicted structures with regards to each ruleset.

actions in a single SIT layer. Further details are given in the supplementary material.

As shown in Table 4, the 3-layer version of the proposed model has the highest score on the 3rd-order ruleset by a margin of 4.07. However, even the 2-layer model outperforms all but two benchmark methods, each consisting of 5 stacked layers. On the 4th-order ruleset the best result is obtained through the proposed model's 4-layer version; second-best performance is attained by the 3-layer model, showing its ability to learn higher-than-pairwise interactions in a single SIT layer. Similarly, on the 5th-order ruleset ($n = 5$ in the table) the 4-layer model outperforms the best benchmark by 3.07 percentage points, with a 5-layer Set Transformer followed by the same permutation component as our proposed set-to-sequence model achieving second best performance. As an ablation study, we include results for a 4-layer Set Transformer, the only difference between it and our 4-layer set-to-sequence model being the SIT matrix augmentation. Its presence accounts for a 4.36 performance increase.

4 Related Work

The earliest neural set-to-sequence model was proposed by Hopfield and Tank (1985). It introduced a constrained version of the set-to-sequence problem, in which the input set must be of a fixed cardinality. However, sets of varying cardinality require different neural architectures, making it limited in application. Kernels between sets have also been proposed to allow kernel methods, such as Support Vector Machines, to tackle set-input problems [Lyu, 2005]. These require a two-step approach where representation learning is separate.

Set-input NN methods, such as DeepSets [Zaheer *et al.*, 2017], obtain a permutation equivariant representation of the input set by encoding each element in an identical and independent way and summing them into a permutation invariant representation. The main limitation is an inability to explicitly model higher order interactions, which are lost during pooling. This is addressed by the Set Transformer [Lee and Lee, 2019], using self-attention; the AttSets [Yang and Wang, 2020], though weighted pooling with learned, feature-specific weights; and RepSet [Skianis and Konstantinos, 2020], by solving a series of network flow problems.

Complete set-to-sequence NN models include the Pointer Network [Vinyals *et al.*, 2015], later improved by its authors to obtain a permutation invariant representation of the set [Vinyals *et al.*, 2016], ATTOrderNet [Cui *et al.*, 2018] and the Enhanced Pointer Network, which requires the model to continuously make pairwise order predictions, resulting in increased sequence coherence [Yin *et al.*, 2020]. There also exists a plethora of other NN order prediction methods, ranging from listwise ranking approaches [Ai *et al.*, 2018] in the field of information retrieval (which, however, presume an existence of a query for which the relevant order is predicted), to permutation matrix approaches [Zhang *et al.*, 2019], which do not handle sets of varying cardinalities. Finally, reinforcement learning has also been applied to combinatorial optimization problems in conjunction with pointer-based attention [Kool *et al.*, 2018]. However, many interesting tasks, such as sentence ordering and complex structure prediction, do not provide the kind of fine-grained signal ideally required by these methods.

5 Conclusion

In this paper, we propose a novel set encoding method for permutation learning and structure prediction. The Set Interdependence Transformer (SIT) is capable of effectively learning higher-order interactions between input set elements, which is crucial for tasks requiring a degree of relational reasoning over large sets of varying cardinality. Our method can take input sets of any size and generalizes well to unseen lengths of the target output sequence. It is easily modularized and can be combined with an attention-based permutation decoder to form a complete set-to-sequence model. Experiments show that this architecture achieves state-of-the-art results on combinatorial optimization challenges, on NLP tasks such as sentence ordering, and in the novel domain of complex structure prediction in the context of product catalogs.

Acknowledgements

This work was partly supported by an Innovation Fund Denmark research grant (number 9065-00017B) and Tjek A/S.

References

[Ai *et al.*, 2018] Qingyao Ai, Keping Bi, Jiafeng Guo, and W Bruce Croft. Learning a deep listwise context model for ranking refinement. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 135–144, 2018.

[Cui *et al.*, 2018] Baiyun Cui, Yingming Li, Ming Chen, and Zhongfei Zhang. Deep attentive sentence ordering network. In *Proc. EMNLP*, pages 4340–4349, 2018.

[Devlin *et al.*, 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proc. NAACL*, 2019.

[Diallo *et al.*, 2020] Aïssatou Diallo, Markus Zopf, and Johannes Fürnkranz. Permutation learning via lehmer codes. In *ECAI 2020*, pages 1095–1102. IOS Press, 2020.

[Farag, 2021] Youmna Farag. *Neural approaches to discourse coherence: modeling, evaluation and application*. PhD thesis, University of Cambridge, 2021.

[Hopfield and Tank, 1985] John J Hopfield and David W Tank. "neural" computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.

[Huang *et al.*, 2020] Qian Huang, Horace He, Abhay Singh, Yan Zhang, Ser-Nam Lim, and Austin Benson. Better set representations for relational reasoning. *Proc. NeurIPS*, 34, 2020.

[Jurewicz and Derczynski, 2021a] Mateusz Jurewicz and Leon Derczynski. Procat: Product catalogue dataset for implicit clustering. *Proc. NeurIPS: Track on Datasets and Benchmarks*, 35, 2021.

[Jurewicz and Derczynski, 2021b] Mateusz Jurewicz and Leon Derczynski. Set-to-sequence methods in machine learning: a review. *Journal of Artificial Intelligence Research*, 71:885–924, 2021.

[Kool *et al.*, 2018] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *Proc. ICLR*, 2018.

[Lee and Lee, 2019] Juho Lee and Yoonho Lee. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753. PMLR, 2019.

[Li and Gao, 2020] Liangcheng Li and Feiyu Gao. An end-to-end ocr text re-organization sequence learning for rich-text detail image comprehension. In *ECCV*, pages 85–100. Springer, 2020.

[Loshchilov and Hutter, 2017] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. In *CoRR, abs/1711.05101, 2017.*, 2017.

[Lyu, 2005] Siwei Lyu. A kernel between unordered sets of data: The gaussian mixture approach. In *European Conference on Machine Learning*, pages 255–267. Springer, 2005.

[Mostafazadeh and Chambers, 2016] Nasrin Mostafazadeh and Nathan Chambers. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proc. NAACL*, pages 839–849, 2016.

[Pandey and Chowdary, 2020] Divesh Pandey and C Ravindranath Chowdary. Modeling coherence by ordering paragraphs using pointer networks. *Neural Networks*, 126:36–41, 2020.

[Santa Cruz and Fernando, 2017] Rodrigo Santa Cruz and Basura Fernando. Deeppermnet: Visual permutation learning. In *Proc. CVPR*, pages 3949–3957, 2017.

[Skianis and Konstantinos, 2020] Skianis and Konstantinos. Rep the set: Neural networks for learning set representations. In *International conference on artificial intelligence and statistics*, pages 1410–1420. PMLR, 2020.

[Vinyals *et al.*, 2015] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Proc. NeurIPS*, volume 2015-January, pages 2692–2700, 2015.

[Vinyals *et al.*, 2016] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. In *Proc. ICLR*, pages 1–11, 2016.

[Vinyals *et al.*, 2019] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, and Mathieu. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[Wang and Wan, 2019] Tianming Wang and Xiaojun Wan. Hierarchical Attention Networks for Sentence Ordering. *AAAI*, 33:7184–7191, 2019.

[Yang and Wang, 2020] Bo Yang and Sen Wang. Robust attentional aggregation of deep feature sets for multi-view 3d reconstruction. *International Journal of Computer Vision*, 128(1):53–73, 2020.

[Yew and Lee, 2020] Zi Jian Yew and Gim Hee Lee. Rpm-net: Robust point matching using learned features. In *IEEE/CVF*, pages 11824–11833, 2020.

[Yin *et al.*, 2020] Yongjing Yin, Fandong Meng, Jinsong Su, Yubin Ge, Lingeng Song, Jie Zhou, and Jiebo Luo. Enhancing pointer network for sentence ordering with pairwise ordering predictions. In *AAAI*, volume 34, pages 9482–9489, 2020.

[Yu *et al.*, 2019] Xiang Yu, Ngoc Thang Vu, and Jonas Kuhn. Learning the dyck language with attention-based seq2seq models. In *ACL Workshop BlackboxNLP*, pages 138–146, 2019.

[Zaheer *et al.*, 2017] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets. In *Proc. NeurIPS*, volume 30, 2017.

[Zhang *et al.*, 2019] Yan Zhang, Adam Prügel-Bennett, and Jonathon Hare. Learning representations of sets through optimized permutations. In *Proc. ICLR*, 2019.

SUPPLEMENTARY MATERIAL 1
Set Interdependence Transformer: Set-to-Sequence Neural Networks for Permutation Learning and Structure Prediction

Mateusz Jurewicz[1,2] and **Leon Derczynski**[1]
[1] IT University of Copenhagen
[2] Tjek A/S
{maju, leod}@itu.dk

1 Introduction

This document constitutes the first of the two main supplementary sources for the paper. It contains complete information regarding each of the presented experimental results as well as a more detailed description of the synthetic structures dataset and its corresponding evaluation functions.

The other source of information is the code base enabling other researchers to reproduce the results. The code also contains full configuration files specifying the entirety of model architectures, layer dimensions and training hyperparameters. Full requirements regarding utilized libraries are provided to make it easier to recreate the original software environment. Hardware information can be found in the main paper. The reported Perfect Match Ratio (PMR) and Kendall's Rank Correlation Coefficient (τ) are scaled by a factor of a 100 for increased readability, following convention from Cui *et al.* (2018) and Yin *et al.* (2020).

2 Experiments

2.1 Travelling Salesman Problem

TSP is an NP-hard problem, which arises in many areas of theoretical computer science. Solutions to this problem comprise algorithms that find usage in DNA sequencing and microchip design [Vinyals *et al.*, 2015]. In this specific setting, the task is to take as input a set of n 2-dimensional points $\mathcal{X} \sim \mathbf{X} \in \mathbb{R}^{n \times 2}$, represented by their coordinates on a Euclidean plane, and output their predicted permutation $\mathbf{y}'_\pi \in \mathbb{N}^n$ such that the lines drawn between these points in the predicted order form the shortest possible circuit, exemplified by the target permutation $\mathbf{y}_\pi$. Point coordinates are sampled uniformly from the range of $[0, 1]$.

The proposed set-to-sequence model is compared with a number of joint set encoding and permutation learning methods, random solutions that consist of valid paths without repetitions and with the paths obtained via the Bellman–Held–Karp algorithm [Hansen and Krarup, 1974] as the target on which the models were trained. Three datasets were generated, each with a different cardinality $n \in (10, 15, 20)$ of the input sets of points. Models were only trained on the dataset consisting of sets of 10 elements, but their performance was also tested on the other cardinalities to gauge the ability to generalize to unseen lengths. The $n = 10$ training dataset consisted of 20K examples, the validation set of 5K and each test set, for all 3 cardinalities, consisted of 3K examples.

	TSP cardinality		
Method	$n = 10$	$n = 15$*	$n = 20$*
Held–Karp	2.97	3.87	4.42
Random	4.48	7.32	8.96
DeepSets	3.27 ± 0.12	4.37 ± 0.19	5.12 ± 0.14
SetTrans	<u>3.00 ± 0.02</u>	3.99 ± 0.05	4.55 ± 0.08
AttSets	3.06 ± 0.06	4.02 ± 0.13	4.64 ± 0.19
RepSet	3.03 ± 0.05	4.13 ± 0.04	4.59 ± 0.11
PtrNet	3.18 ± 0.26	4.85 ± 0.69	5.83 ± 0.85
RPW	3.67 ± 0.10	4.92 ± 0.04	4.82 ± 0.11
ATTOrdNet	3.10 ± 0.06	4.53 ± 0.15	5.63 ± 0.39
PtrNet+	3.01 ± 0.04	<u>3.92 ± 0.06</u>	<u>4.52 ± 0.09</u>
Ours	**2.98 ± 0.01**	**3.90 ± 0.03**	**4.46 ± 0.06**

Table 1: Performance on the TSP in terms of average tour length.

Each tested model had approximately 5 million trainable parameters, corresponding dimensions of hidden layer weights and number of layers. For specific details we refer the reader to the provided code and configuration files. The proposed model consisted of 3 layers of SIT transformations over 4 attention heads. The AdamW [Loshchilov and Hutter, 2017] optimizer was used in all cases, with weight decay coefficient 1e-2, learning rate (α) 1e-4, dropout rate of 0.1 and batch size 32, for 50 epochs. The reported results are averaged over full training runs with reported standard deviation. In the case of TSP the only metric used is the average predicted tour length over the entire test set for each cardinality. As shown in table 1, our proposed model outperforms the baselines by predicting shorter average paths with lower standard deviation, the difference becoming more pronounced the more the input set cardinality differed from the ones seen during training. Specifically, the proposed model predicts paths that are shorter by 0.02 and 0.06 when $n = 15$ and $n = 20$ respectively, exhibiting increased ability to generalize to unseen lengths.

2.2 ROCStory

ROCStory is a popular sentence ordering dataset consisting of 98,162 common-sense stories with 50 words per story

Method	ROCStory		PROCAT		Synthetic $\|y\| = 30$	
	PMR	Kendall's τ	PMR	Kendall's τ	PMR	Kendall's τ
DeepSets (2017)	33.81 ± 4.55	62.41 ± 4.2	15.21 ± 2.69	35.15 ± 3.8	25.61 ± 1.90	40.94 ± 2.7
Set Transformer (2019)	41.94 ± 1.29	73.15 ± 1.9	21.03 ± 0.98	42.74 ± 2.6	30.23 ± 1.86	44.71 ± 2.9
AttSets (2020)	42.51 ± 1.45	74.81 ± 2.3	19.24 ± 1.30	38.44 ± 1.7	31.48 ± 2.04	46.20 ± 3.2
RepSet (2020)	42.47 ± 1.61	73.39 ± 1.7	22.72 ± 1.19	41.30 ± 2.5	34.84 ± 1.99	47.95 ± 3.4
PointerNet (2015)	28.73 ± 3.91	59.72 ± 6.2	02.90 ± 1.17	16.85 ± 3.4	17.33 ± 3.41	32.66 ± 3.1
Read-Process-Write (2016)	20.38 ± 7.22	51.78 ± 8.5	03.54 ± 1.06	21.11 ± 2.9	21.40 ± 2.07	36.28 ± 4.3
ATTOrderNet (2018)	41.14 ± 2.10	73.02 ± 2.0	17.33 ± 2.31	37.47 ± 3.2	28.02 ± 2.40	42.67 ± 3.4
Enhanced PointerNet+ (2020)	44.32 ± 1.25	76.43 ± 1.3	22.09 ± 1.57	42.53 ± 1.9	34.34 ± 1.39	48.14 ± 2.9
Ours (2021)	**47.00 ± 0.89**	**79.86 ± 0.9**	**25.61 ± 1.81**	**46.41 ± 1.3**	**37.16 ± 1.01**	**52.03 ± 2.2**

Table 2: Precise Match Ratio (PMR) and Kendall's Rank Correlation Coefficient (τ) results for ROCStory, PROCAT and the Synthetic task.

on average [Mostafazadeh and Chambers, 2016], publicly available at the following link: https://cs.rochester.edu/nlp/rocstories.

Each story contains exactly 5 sentences. Following common practice established by Wang and Wan (2019) and Yin *et al.* (2020), the provided dataset split is used to get the training, testing and validation datasets of 78K, 9.8K and 9.8K stories per set. The input is a set of five sentences in random order π, in English, with the canonical order $\mathbf{y}$ provided as the learning target. In our experiment, we use the concatenated and averaged output of the last 4 layers of the cased large version of BERT [Devlin *et al.*, 2019] to obtain fixed-length vector representations of each sentence. The language model is frozen during training to isolate the effect of different set encoding methods on the permutation task performance.

Each tested model had approximately 7 million trainable parameters, corresponding dimensions of hidden layer weights and number of layers. For specific details we refer the reader to the provided code and configuration files. The proposed model consisted of 5 layers of SIT transformations over 4 attention heads. The AdamW [Loshchilov and Hutter, 2017] optimizer was used, with weight decay coefficient 1e-2, learning rate (α) 1e-4, dropout rate of 0.1 and batch size 32, for 50 epochs. The reported results are averaged over 3 full training runs with reported standard deviation. The performance is reported in terms of the Perfect Match Ratio and the Kendall's τ Rank Correlation Coefficient, both of which are frequently used metrics for the sentence ordering challenge [Cui *et al.*, 2018; Wang and Wan, 2019; Yin *et al.*, 2020]. The overall results on the two main ROCStory and PROCAT datasets are shown in table 2. Our proposed model outperforms the state-of-the-art on both datasets. The PMR scores are increased by 2.68 and 2.89 and the τ by 3.43 and 3.67 over second best performances on the ROCStory and PROCAT datasets respectively.

2.3 PROCAT

PROCAT is a product catalog structure dataset consisting of over 1.5 million product offers composed into over 10,000 human-designed catalogs. It contains relative visual prominence information and the exact split of offers into complementary sections, which are then ordered into a full catalog [Jurewicz and Derczynski, 2021]. It is publicly available at the following link: https://doi.org/10.6084/m9.figshare.14709507.

Much like ROCStory, it lends itself to set-to-sequence challenges. A permutation learning objective is formed by the original order of offers and sections within a catalog. Each section can consist of a variable number of offers and each catalog of a variable number of sections, thus requiring greater flexibility than ROCStory. The input is formed by the text features of individual offers and their random order (along with section break markers) in the matrix representing the input set. The target is their actual order in the original catalog, in the proper section split defined by the section break markers. In our experiment, we used the concatenated and averaged output of the last 4 layers of the cased large version of BERT [Devlin *et al.*, 2019], publicly available under this link, to obtain fixed-length vector representations of each offer. The language model is frozen during training to isolate the effect of different set encoding methods on the permutation task performance. The split into training and test sets follows the one provided by the authors of the dataset (80/20).

Each tested model had approximately 7 million trainable parameters, corresponding dimensions of hidden layer weights and number of layers. For specific details we refer the reader to the provided code and configuration files. The proposed model consisted of 5 layers of SIT transformations over 4 attention heads. The AdamW [Loshchilov and Hutter, 2017] optimizer was used, with weight decay coefficient 1e-2, learning rate (α) 1e-4, dropout rate of 0.1 and batch size 32, for 50 epochs. The reported results are averaged over 3 full training runs with reported standard deviation. The performance is reported in terms of the Perfect Match Ratio and the Kendall's τ Rank Correlation Coefficient, both of which are frequently used metrics for permutation learning challenges [Cui *et al.*, 2018; Wang and Wan, 2019; Yin *et al.*, 2020]. The overall results are shown jointly in table 2. The proposed model outperforms the state-of-the-art on the PROCAT dataset. The PMR score is increased by 2.89 and the τ by 3.67 over second-best tested performance.

2.4 Formal Grammars

Formal grammars are a well-studied concept from the field of formal language theory, lending itself to the set-to-sequence framing as a machine learning task [Nakamura and Imada, 2011]. A grammar describes how to form strings from a language's alphabet that are valid according to this language's syntax. In our experiments we tackle two context-sensitive and one context-free grammar, as originally categorized by Chomsky (1956). The task is to take a set of terminal symbols that can be composed into a grammatical string, or in other words ordered into their original index sequence $\mathbf{y}$ which was shuffled to obtain the randomly ordered input set matrix $\mathbf{X}_\pi$.

We generate data from the first grammar, $a^n b^n c^n$, by sampling n uniformly from [1,100]. For the second grammar, $a^n b^k c^{nk}$, both n and k are sampled uniformly from [1,25], with the maximum sequence length being 675. For the Van Dyck we only form sequences consisting of 4 terminal symbols:'{', '}', '(' and ')', of random length sampled uniformly from [4,50]. The Van Dyck language requires that each pair of parenthesis and square brackets is properly closed. Each training set consisted of 5K randomly shuffled arrays representing $\mathbf{X}_\pi$, each test and validation set consisted of 1K examples.

Each tested model had approximately 5 million trainable parameters, corresponding dimensions of hidden layer weights and number of layers. For specific details we refer the reader to the provided code and configuration files. The progressive masking is removed from the permutation modules for this task. The proposed model consisted of 3 layers of SIT transformations over 4 attention heads. The AdamW [Loshchilov and Hutter, 2017] optimizer was used, with weight decay coefficient 1e-2, learning rate (α) 1e-4, dropout rate of 0.1 and batch size 32, for 50 epochs. The reported results are averaged over 3 full training runs with reported standard deviation. The performance is reported in terms of the percentage of predicted grammatical sequences.

In table 3 we present the average percentage of predicted sequences that adhered to the rules underlying two context sensitive grammars and one context free grammar, in the form of the Van Dyck language. The simplest grammar ($a^n b^n c^n$) was able to be fully learned by most models utilizing the enhanced pointer network as their permutation module, our proposed method among them. With regards to the more challenging $a^n b^k c^{nk}$ context-sensitive grammar our method outperformed the second best by 1.24 percentage points and by 1.56 in the case of the Van Dyck language.

2.5 Synthetic Structures

The synthetic structure datasets are generated via an expanded version of the library provided by Jurewicz and Derczynski (2021), which is publicly available under the following link: https://github.com/mateuszjurewicz/procat.

Each 1-dimensional structure is generated through a set of customizable rulesets, which define how the atomic tokens that comprise each structure can be ordered. In the reported experiments we generate structures consisting of 30 instances of the 5 atomic token types, represented by five colours: blue, yellow, red, green and purple. And additional token type represents section breaks.

	Grammars % valid		
Method	$a^n b^n c^n$	$a^n b^k c^{nk}$	Van Dyck
DeepSets	97.17 ± 2.9	94.52 ± 2.4	79.84 ± 2.3
SetTrans	**100.0 ± 0.0**	96.62 ± 0.8	92.16 ± 1.5
AttSets	98.24 ± 1.3	96.12 ± 1.1	92.31 ± 1.2
RepSet	**100.0 ± 0.0**	97.64 ± 0.9	93.22 ± 1.4
PtrNet	79.31 ± 5.3	75.85 ± 4.7	58.85 ± 6.7
RPW	86.41 ± 1.5	81.74 ± 1.2	61.13 ± 5.3
ATTOrdNet	97.75 ± 1.1	94.17 ± 1.0	84.31 ± 3.8
PtrNet+	**100.0 ± 0.0**	96.82 ± 0.7	93.51 ± 1.2
Ours	**100.0 ± 0.0**	**98.88 ± 0.6**	**95.07 ± 1.0**

Table 3: Results for 2 context-sensitive and 1 context-free grammar, the Van Dyck language, consisting of {} and () pairs. On a scale of 0-100, reflecting the proportion of predicted grammatical sequences.

	n-th Order Relation Ruleset		
Method	$n = 3$	$n = 4$	$n = 5$
DeepSets	53.37 ± 6.1	40.12 ± 4.0	21.13 ± 0.2
SetTrans (4)	92.41 ± 1.6	92.90 ± 1.4	91.74 ± 1.7
SetTrans (5)	94.66 ± 1.9	93.38 ± 1.3	92.93 ± 1.5
AttSets	93.84 ± 2.8	92.93 ± 1.1	86.44 ± 1.3
RepSet	91.29 ± 3.4	90.84 ± 2.4	89.73 ± 2.9
PtrNet	41.26 ± 5.3	31.96 ± 4.8	15.23 ± 4.2
RPW	45.11 ± 2.0	36.31 ± 1.6	16.12 ± 2.5
ATTOrdNet	82.31 ± 4.4	67.07 ± 2.0	0.12 ± 1.4
PtrNet+	89.58 ± 3.9	87.22 ± 3.2	86.94 ± 2.8
Ours (2 layers)	93.83 ± 3.6	89.01 ± 2.6	86.79 ± 1.9
Ours (3 layers)	**98.73 ± 0.8**	93.44 ± 2.3	92.72 ± 1.8
Ours (4 layers)	98.48 ± 1.2	**97.52 ± 0.9**	**96.10 ± 1.6**

Table 4: Synthetic structure prediction scores per ruleset type, split by order of interaction required. On a scale of 0-100, reflecting the proportion of valid predicted structures with regards to each ruleset.

Rulesets define the valid composition of sections and their order. An example of a valid section composition might be one that consists only of 3 blue, yellow or red tokens. An example of a valid section order might be always starting the sequential structure with an all-red section or always ending it with an all-blue one. These are intended to represent a more abstract simplification of the rules that govern actual product catalog structures.

Which ruleset should be applied when predicting a synthetic structure depends on higher order relational interactions between the elements of the input set of tokens. A pairwise interaction rule would trigger if the input set contains a blue and a yellow token. A third-order interaction rule, by comparison, would have to depend on the presence or absence of 3 atomic token types and so forth. A real life example of a 3rd order interaction between product offers comes in the form of an input set including beef and French cheese offers. With just the pairwise interaction between the two, they don't necessarily form a good pairing to go together in the same

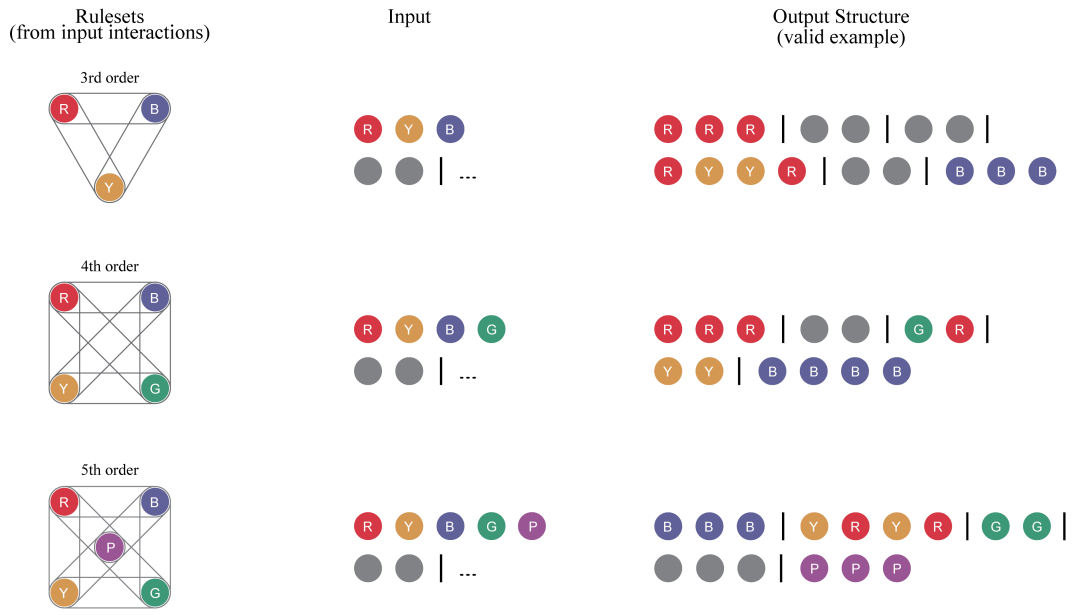


Figure 1: Synthetic structures and n-th order interaction rulesets. Rulesets (left) represent higher-order interactions that determine the validity of an output structure. Which ruleset is applicable is dependent on the input set composition (middle). For example, if a red, yellow and blue token are present in the input set, the valid output structure (right) has to start with an all-red section, end on an all-blue section and can only combine red and yellow tokens together in a 50/50 split. Inclusion of a 4th special token (second row) introduces a 4th-order interaction which changes the underlying ruleset and, in consequence, the target output structure.

section in a complementary way. However, if an offer for red wine is also present in the input set, the 3rd order interaction between these three products ties them together into a popular catalog section.

Given a predicted structure, based on a specific input set, we can easily check whether it is valid in relation to the appropriate ruleset. Thus we can report what percentage of predicted structures were valid as per rulesets requiring the model to learn n-th order interactions. This allows us to approximate a model's relational reasoning capability. For a visual explanation see figure 1.

Reported results are split between tables 2 and 4. In the former, we compare the PMR and τ on a synthetically generated structures dataset of length 30, following the default rulesets, where our model outperformed the best benchmarks by 2.32 and 1.89 respectively. In all reported cases the training dataset consists of 10K examples generated via the default rulesets including 3rd, 4th and 5th order interactions, and the test dataset consists of 5K examples.

The results presented in table 4 require further explanation. In order to assess the proposed model's ability to learn n-th order relational rules without performing n pairwise attention operations through a stack of n attention layers, we construct versions of our set-to-sequence model with $< n$ set encoding layers. Specifically, the tested versions of the proposed architecture consist of a single permutation equivariant, transformer-style layer for obtaining element representations followed by m SIT layers. Here, $m \in (1, \ldots, n-2)$ so that the proposed models always consists of at least one less layers $(m+1)$ than the number (n) required assuming only pairwise interactions per layer.

For specific dimensions of hidden layers we refer the reader to the provided code and configuration files. The proposed models consisted of m layers of SIT transformations over 4 attention heads. The AdamW [Loshchilov and Hutter, 2017] optimizer was used, with weight decay coefficient 1e-2, learning rate (α) 1e-4, dropout rate of 0.1 and batch size 32, for 50 epochs. The reported results are averaged over 3 full training runs with reported standard deviation.

Table 4 shows the percentage of predicted synthetic structures that were valid under the relevant ruleset. Each column refers to rulesets depending on n-th order relational interactions, where $n \in (3, 4, 5)$. The baselines in this case are specifically versions of the original models consisting of exactly 5 layers (with one exception) responsible for pairwise attention transformations. These are contrasted with three versions of the proposed model, with 1, 2 or 3 layers of the proposed SIT set encoder, plus 1 layer of the initial permutation equivariant attention, for a total of 2, 3 and 4 layers (see bottom rows of table 4). Thus the proposed models are always at a disadvantage in terms of the number of stacked attention operations, requiring the SIT encoder to learn higher

than pairwise interactions.

As seen in table 4, the 3-layer version of the proposed model has the highest score on the 3rd order ruleset by a margin of 4.07 compared to the best benchmark. However, even the 2-layer version outperforms all but two benchmark methods, each consisting of 5 stacked layers. On the 4th order ruleset the best result is obtained through the proposed model's 4-layer version, but the second best performance is attained by the 3-layer version, showcasing its ability to learn higher-than-pairwise interactions in a single SIT layer. Similarly on the 5th order ruleset ($n = 5$ in the table) the 4-layer version outperforms the best benchmark by 3.07 percentage points, with a 5-layer Set Transformer followed by the same permutation module as our proposed set-to-sequence model achieving second best performance. As an ablation study, we additionally include results for a 4-layer version of the Set Transformer (4), where the only difference between our 4-layer set-to-sequence model is the removal of the SIT matrix augmentation. Its presence accounts for a 4.36 performance increase.

References

[Chomsky, 1956] Noam Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

[Cui *et al.*, 2018] Baiyun Cui, Yingming Li, Ming Chen, and Zhongfei Zhang. Deep attentive sentence ordering network. In *Proc. EMNLP*, pages 4340–4349, 2018.

[Devlin *et al.*, 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proc. NAACL*, 2019.

[Hansen and Krarup, 1974] Keld Helbig Hansen and Jakob Krarup. Improvements of the held—karp algorithm for the symmetric traveling-salesman problem. *Mathematical Programming*, 7(1):87–96, 1974.

[Jurewicz and Derczynski, 2021] Mateusz Jurewicz and Leon Derczynski. Procat: Product catalogue dataset for implicit clustering. *Proc. NeurIPS: Track on Datasets and Benchmarks*, 35, 2021.

[Loshchilov and Hutter, 2017] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. In *CoRR, abs/1711.05101, 2017.*, 2017.

[Mostafazadeh and Chambers, 2016] Nasrin Mostafazadeh and Nathan Chambers. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proc. NAACL*, pages 839–849, 2016.

[Nakamura and Imada, 2011] Katsuhiko Nakamura and Keita Imada. Towards incremental learning of mildly context-sensitive grammars. In *2011 10th International Conference on Machine Learning and Applications and Workshops*, volume 1, pages 223–228. IEEE, 2011.

[Vinyals *et al.*, 2015] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Proc. NeurIPS*, volume 2015-January, pages 2692–2700, 2015.

[Wang and Wan, 2019] Tianming Wang and Xiaojun Wan. Hierarchical attention networks for sentence ordering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7184–7191, 2019.

[Yin *et al.*, 2020] Yongjing Yin, Fandong Meng, Jinsong Su, Yubin Ge, Lingeng Song, Jie Zhou, and Jiebo Luo. Enhancing pointer network for sentence ordering with pairwise ordering predictions. In *AAAI*, volume 34, pages 9482–9489, 2020.

Chapter 7

Article 4

Article details:

4. **"Clustering and Ordering Variable-Sized Sets: The Catalog Problem"**
   Mateusz Jurewicz and Leon Derczynski
   Under review as a conference paper in the *Proceedings of the 11th International Conference on Learning Representations*, ICLR 2023.

Context. At the time of writing of the fourth article, significant progress was becoming apparent in the field of supervised neural clustering. Models such as Deep Amortized Clustering [21], Neural Clustering Process [20] and the Attentive Clustering Process [22] were being applied to synthetic tasks, such as clustering mixtures of 2D Gaussians, and real-world challenges like graph community detection. This aligned very well with the remaining aspects of the Catalog Problem that were challenging to accurately address with set-to-sequence methods and their modifications, two of which are proposed and tested in this article. Namely, these aspects revolved around avoiding the prediction of in-section order and gaining the ability to output a varying, input-dependent number of clusters. Both of these were addressed by the aforementioned clustering methods, which appeared to lack only the ability to order clusters, present in set-to-sequence methods. A natural step was to combine and further improve upon them, based on their limitations that became apparent in experiments, particularly in relation to learning cluster-level cardinality constraints.

Contributions. This paper proposed the full Neural Ordered Clusters (NOC) neural architecture, capable of taking input sets of varying cardinality and predicting an input-dependent, ordered, partitional clustering in a supervised manner. NOC combines powerful neural clustering techniques, further enhanced through the addition of the Set Interdependence Transformer. SIT is utilized to obtain fixed-length vector representations of the input set, each individual cluster and all predicted clusters at various stages of the NOC architecture, such that they encode higher-order interactions. An additional step is added to the clustering stage, where a cardinality threshold of the current cluster is predicted and used to adjust the initial cluster assignments, enabling NOC to predict clusters that abide by the compositional rules explicitly present in the synthetic catalogs and implicitly in the PROCAT dataset. The article also introduces and defines the Catalog Problem and tests NOC's performance on datasets that exemplify it, showing improved performance in comparison to a wide array of baselines.

Recent Developments. Given that the article is still under review at the time of writing, there has not been ample time for new methods to emerge. However, between the submission of this article and the submission of this thesis an interesting model has been proposed by Liu *et al.* (2022). Introduced under the name of Deep amortized Relational Model (DaRM), this technique uses a generative process to exploit both pairwise and higher-order interactions between input elements in both inter-group (structural) and intra-group (compositional) relations. Involving both GNNs and generation of cluster prototypes (through selection of pioneer nodes), this technique represents the cutting edge in community detection and resembles certain aspects of NOC. DaRM's inherently hierarchical nature could possibly lend itself to being extended to challenges like the Catalog Problem in future work.

7.1 Clustering and Ordering Variable-Sized Sets: The Catalog Problem

CLUSTERING AND ORDERING VARIABLE-SIZED SETS: THE CATALOG PROBLEM

Anonymous authors
Paper under double-blind review

ABSTRACT

Prediction of a **varying number** of **ordered clusters** from sets of **any cardinality** is a challenging task for neural networks, combining elements of set representation, clustering and learning to order. This task arises in many diverse areas, ranging from medical triage and early discharge, through machine part management and multi-channel signal analysis for petroleum exploration to product catalog structure prediction. This paper focuses on the latter, which exemplifies a number of challenges inherent to adaptive ordered clustering, referred to further as the eponymous *Catalog Problem*. These include learning variable cluster constraints, exhibiting relational reasoning and managing combinatorial complexity. Despite progress in both neural clustering and set-to-sequence methods, no joint, fully differentiable model exists to-date. We develop such a modular architecture, referred to further as Neural Ordered Clusters (NOC), enhance it with a specific mechanism for learning cluster-level cardinality constraints, and provide a robust comparison of its performance in relation to alternative models. We test our method on three datasets, including synthetic catalog structures and PROCAT, a dataset of real-world catalogs consisting of over 1.5 M products, achieving state-of-the-art results on a new, more challenging formulation of the underlying problem, which has not been addressed before. Additionally, we examine the network's ability to learn higher-order interactions and investigate its capacity to learn both compositional and structural rulesets.

1 INTRODUCTION

The ability to group members of a set and order these groups is key to many important real-world decision-making processes. It finds applications ranging from supply chain management (Wenzel et al., 2019) to prioritization in medical triage (Miles et al., 2020; Buchard et al., 2020). Other application domains include petroleum exploration (Rabiller et al., 2010), business process analytics (Le et al., 2014), parallelization and machine part management (Bakkelund, 2022) and product catalog structuring (Jurewicz & Derczynski, 2022), where the goal is to take a set of products and work out how to group them together and order these groups to form a coherent product catalog. We term this problem of simultaneously grouping and ordering a set of items the Catalog Problem.

This paper defines the Catalog Problem and presents an investigation into neural network approaches to it. To this end we introduce a fully-differentiable, deep learning (DL) model architecture that addresses the Catalog Problem. In it, sets of items are clustered into groups, and an ordering between groups is established. All of this is achieved in a *supervised* manner. While clustering methods are often unsupervised (Aljalbout et al., 2018; Ronen et al., 2022), the meaningful ordering of clusters often requires more knowledge than is available from the instance representation alone.

Similarly, learning to order is often framed as a supervised learning task (Vinyals et al., 2015; Yin et al., 2020; Shi, 2022). Referred to further as *set-to-sequence* (S2S), this area and its corresponding methods inspire the cluster-ordering aspect of our proposed Neural Ordered Clusters (NOC) model. Both neural clustering and set-to-sequence models have limitations. Element-wise neural clustering methods require $O(n)$ passes over the input set of cardinality n[1]. Cluster-wise and attention-based

[1] $O(n)$ can be prohibitive with large input sets ($n >= 1000$), which is often the case in many interesting set-input problems such as 3D point cloud tasks (Qi et al., 2017; Ge et al., 2018; Zhao et al., 2021).

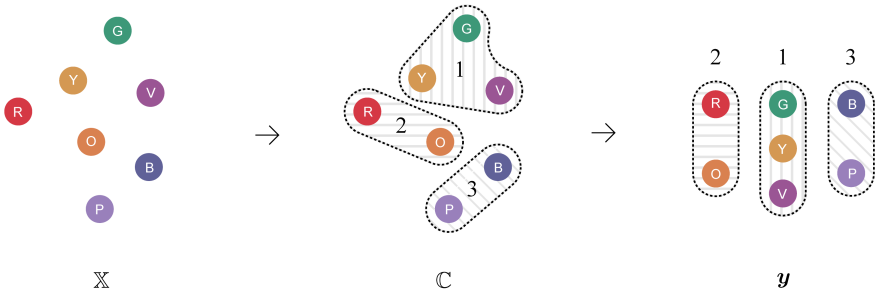


Figure 1: The Catalog Problem. From left to right: a set of input elements ($\mathbb{X}$); partitional clustering of those elements ($\mathbb{C} = \{\mathbb{C}_1, \mathbb{C}_2, \mathbb{C}_3\}$); and a target ordering over those clusters ($\mathbf{y} = (\mathbb{C}_2, \mathbb{C}_1, \mathbb{C}_3)$), left to right. A candidate model has to perform all these tasks using information about inter-element relations and intra-cluster relations in order to characterise a cluster, and inter-cluster relations to generate the final, ordered clustering.

models are more computationally efficient, but exhibit a limited ability to learn cluster cardinality constraints (Pakman et al., 2020), integral to both the prototypical Catalog Problem and its practical instantiations. Set-to-sequence methods, on the other hand, are effective at learning constraints (Zhu et al., 2021) and generalizing to unseen distributions (Wen, 2022). However, they are limited by their inability to predict an input-dependent number k of clusters without major adjustments (Fernández-González, 2022), two of which are proposed in Section 5.1. Nonetheless, these S2S variants suffer from noisy in-cluster order and cascading first-choice costs (Gan et al., 2020; Vial et al., 2022).

To address these challenges, we implement a unified clustering and cluster ordering method. NOC is capable of predicting ordered, partitional cluster assignments for elements of sets of varying cardinality. It infers a flexible, input-dependent number of diverse clusters, maintains $O(k)$ complexity and utilizes a jointly learned representation of set elements to find the target cluster order. Unlike existing neural clustering methods, it exhibits the ability to learn cluster cardinality constraints through supervision. To our knowledge, no other neural-based method exists to address such challenges in an end-to-end, jointly trainable way, instead performing clustering and ordering as two separate tasks, sometimes with the separate addition of a representation learning step (Aljalbout et al., 2018). All code, hyper-parameters and datasets required for reproducing our results are made available and detailed via the appendix. To summarize, our contributions are as follows:

- Firstly, we introduce the Catalog Problem, a novel joint clustering and cluster ordering problem over sets of elements, which is a challenging variant of the set-to-sequence domain with multiple aspects that are not handled by existing neural methods. We exemplify and tackle this problem on three datasets, including a real-world dataset of over 1.5 M products grouped and ordered into product catalogs by human experts.
- Secondly, we propose a novel, fully differentiable, joint neural clustering and cluster ordering model, Neural Ordered Clusters (NOC), capable of predicting an adaptive, input-dependent number of ordered, partitional clusters from sets of varying cardinality.
- Thirdly, we provide a robust comparison of existing and proposed neural methods on the Catalog Problem using synthetic & real-world datasets, providing insights into the models' capacity to learn higher-order relational rules of cluster composition and ordered structure.

2 The Catalog Problem

Many problems require predicting an adaptive, input-dependent number of partitional clusters from sets of varying cardinality and consequently ordering these clusters according to a target preference. These include but are not limited to vitally important challenges related to triage-like tasks in the medical, military and crisis relief contexts (Kennedy et al., 1996), which are only beginning to be tackled via machine learning methods in a supervised or semi-supervised settings (Buchard et al., 2020). We refer to such challenges through the umbrella term of the titular Catalog Problem.

Multiple datasets from various areas of application either lend themselves directly to this formulation or can be adjusted to it, such as the PROCAT dataset which contains a supervision target in the form of the human-designed structure of product catalogs (Jurewicz & Derczynski, 2021) or the MIMIC-III dataset of electronic health records (Johnson et al., 2016). Other curated datasets exist in the field of large-scale image preference grouping (Chang et al., 2016), mortality risk ranking (Kwon et al., 2018) and early warning systems for influenza (Espino et al., 2003), with further supervised datasets being potentially obtainable in such areas as grouping and ordering sets of tasks for parallel runs on a finite number of processors (Bakkelund, 2022), placement in physical design of integrated circuits (Lin et al., 2018) and graph clustering (Tian et al., 2014).

in the Catalog Problem the input is an unordered set of unique elements $\mathbb{X} = \{x_1, \ldots, x_n\}$, with a varying cardinality $n = |\mathbb{X}|$. The target output is a sequence of clusters $\mathbf{y} = (\mathbb{C}_1, \ldots, \mathbb{C}_k)$, where k is an input-dependent number of clusters and each cluster is defined by the elements assigned to it, whose number can differ per cluster. For example, given an input set $\mathbb{X} = \{x_1, x_2, x_3, x_4, x_5\}$ the target can take the form of the sequence $\mathbf{y} = (\{x_3, x_4\}, \{x_1\}, \{x_2, x_5\})$. This example thus requires the prediction of the following set of clusters: $\mathbb{C} = \{\mathbb{C}_1, \mathbb{C}_2, \mathbb{C}_3\}$, such that if $\mathbb{C}_1 = \{x_1\}$, $\mathbb{C}_2 = \{x_3, x_4\}$ and $\mathbb{C}_3 = \{x_2, x_5\}$ then $\mathbf{y} = (\mathbb{C}_2, \mathbb{C}_1, \mathbb{C}_3)$. All elements must be assigned to a cluster, an element can only belong to one cluster and empty clusters are not allowed.

The problem requires learning a parameterized function ρ_θ, capable of taking the input set, finding the optimal number of clusters to partition it into (k from the supervision target), assigning each element to one of k clusters and ordering these clusters. Thus $\rho_\theta(\mathbb{X}) = \hat{\mathbf{y}}$, with k being predicted indirectly. The optimal assignments and order of clusters is known from examples in the datasets. This is a general problem that, as is shown by experiments later in this paper, is non-trivial. For a visual explanation, see Figure 1. Although the Catalog Problem is so named because it models the task of creating a catalog of items, e.g. products, no specific application is prescribed; the problem only defines input and output types and a relation between these two. The difficulty lies in learning the relationships between both input elements and groups thereof. This difficulty can be compounded by the uniqueness of input elements, making learning representations difficult, due to the scarcity of distributional information.

2.1 Related Work

There have been many machine learning (ML) approaches to clustering with some notion of order, albeit often aimed at preventing the impact of this order on the clusters (Fisher et al., 1992). In the more common, *unsupervised* setting these range from hierarchical clustering (Johnson, 1967; Chu, 1974), through ordinal clustering (Janowitz, 1978) and incremental conceptual clustering (Fisher, 1987) to Markov clustering (Van Dongen, 2000) and other, more recent methods (Ankerst et al., 1999; Turowski et al., 2020). Certain unsupervised clustering methods without the ordering element, like affinity propagation (Frey & Dueck, 2007; Vlasblom & Wodak, 2009), are also capable of outputting an adaptive, input-dependent number of partitional clusters.

Closer to the supervised setting of interest, there have been attempts to leverage instance labels to augment k-means (Ergun et al., 2022), improve the interpretability thereof (Peng et al., 2022) and to cluster labelled data to facilitate permutation learning (Lee & Kim, 2020). Similarly, contrastive clustering utilizes soft labels to maximize the similarities of positive pairs while minimizing those of negative ones (Li et al., 2021), in an approach reminiscent of the pairwise order prediction modules that resulted in increased performance on strictly set-to-sequence tasks (Yin et al., 2020). However, these supervised clustering methods do not yield an ordering of clusters.

3 Background

We identify three classes of neural approaches to solving aspects of the Catalog Problem: set representation; neural clustering; and ordering through pointer attention. Firstly, learning permutation invariant **set representations** that can encode higher-order interactions is vital, due to the complex relational factors among set elements that determine the target output. Deep learning advances in set representation focus primarily on being able to effectively learn such relations, starting with *Deep Sets* (Zaheer et al., 2017), through the *Set Transformer* (Lee et al., 2019) to modifications thereof (Girgis et al., 2021; Jurewicz & Derczynski, 2022). These methods can be used for both

encoding elements and representing clusters. In the Set Transformer, given an unordered set ($\mathbb{X}$), we obtain the representations of set elements ($\boldsymbol{E}_\pi$) and subsequently the entire input set ($\boldsymbol{s}$) via:

$$\boldsymbol{E}_\pi = \text{MAB}(\mathbb{X}, \mathbb{X}) = \text{LN}(\boldsymbol{H} + \phi(\boldsymbol{H})), \text{ where } \boldsymbol{H} = \text{LN}(\boldsymbol{X} + \text{MHA}(\boldsymbol{X}_q, \boldsymbol{X}_k, \boldsymbol{X}_v)), \quad (1)$$

$$\mathbf{s} = \text{PMA}(\mathbf{E}_\pi) = \text{MAB}(\mathbf{r}, \mathbf{E}_\pi), \quad (2)$$

Here, multihead, intra-set attention (denoted as MHA) is performed by casting the input set $\mathbb{X}$ to query, key, and value matrices $\boldsymbol{X}_q, \boldsymbol{X}_k, \boldsymbol{X}_v$ according to an arbitrary permutation π, and adding a residual connection as defined by Vaswani et al. (2017), without positional encoding. This operation is incorporated into a multihead self-attention block (MAB) by the inclusion of a row-wise feed-forward neural network (NN) ϕ, with layer normalization (LN) after each block (Ba et al., 2016), resulting in a permutation equivariant[2] matrix of per-element representations ($\boldsymbol{E}_\pi$). These are then aggregated into a permutation invariant representation of the entire set ($\boldsymbol{s}$) by performing pooling by multihead attention (PMA) between the per-element representations and a learned seed vector $\boldsymbol{r}$. These operations are used extensively in our method for encoding both the initial input set and the predicted clusters.

Secondly, supervised **neural clustering** obtains per-element cluster assignments ($\hat{c}_i$) through a number of modular functions parameterized by NNs. These networks leverage set representation methods to encode the set of currently available, unassigned elements ($\boldsymbol{U}_j$), each previously completed cluster ($\boldsymbol{g}_j$) and consequently all clustered elements jointly ($\boldsymbol{G}_j$), at each step j. This is paired with an algorithm for selecting the next j-th cluster (if clusterwise) or element (if pointwise) to be considered until nothing remains to be assigned.

In the $O(k)$ clusterwise formulation each cluster assignment is the output of another NN (ρ), in the form of the probability of each unassigned, encoded element ($\boldsymbol{x}_i$) belonging to the current j^{th} cluster ($p_\theta(\hat{c}_i = j)$), conditioned on these representations and trained in a teacher-forced manner, with loss calculated only for the elements belonging to the current cluster. In the attention-based, clusterwise framework of the *Attentive Clustering Process* (Pakman et al., 2020) a random anchor element ($\boldsymbol{x}_a$) is obtained at each step j, along with a vector ($\boldsymbol{z}_j$) sampled from a Gaussian latent variable, which represents the features of the j^{th} cluster and is trained as a conditional variational autoencoder (Sohn et al., 2015).

$$p(\boldsymbol{z}_j \mid \mathbb{X}_j) = \mathcal{N}(\boldsymbol{z}_j \mid \mu(\boldsymbol{x}_a, \boldsymbol{U}_j, \boldsymbol{G}_j), \sigma(\boldsymbol{x}_a, \boldsymbol{U}_j, \boldsymbol{G}_j)), \quad (3)$$

where $\mathbb{X}_j$ is used as a shorthand for the state of the input set $\mathbb{X}$ at step j, referring to which elements have been assigned to which clusters, μ and σ are MLPs which take as input the j^{th} anchor element ($\mathbf{x}_a$), the representations of all currently unassigned points ($\mathbf{U}_j$) and all previously predicted clusters ($\mathbf{G}_j$) and output the means and standard deviations for each dimension of the $\mathbf{z}_j$ vector, on which the final, per-element assignment probabilities are conditioned for the current j-th cluster:

$$p_{\theta,i}(\hat{c}_i = j \mid \mathbb{X}_j) = \text{sigmoid}(\rho(\boldsymbol{x}_i, \boldsymbol{x}_a, \boldsymbol{z}_j, \boldsymbol{U}_j, \boldsymbol{G}_j)). \quad (4)$$

Thirdly, **pointer attention** can be used to select a single element from a set of any cardinality n, common in set-to-sequence NNs. At each step $m \in \{1, \ldots, k\}$ it outputs an attention vector ($\boldsymbol{a}_m$) over all obtained clusters $\mathbb{C}$. As the clusters are selected sequentially, this represents their predicted order, with highest attention value pointing to the index of the m-th cluster in that order:

$$\boldsymbol{a}_m = \text{softmax}(\boldsymbol{v}^\top \tanh(\boldsymbol{W}_1 \mathbb{C} + \boldsymbol{W}_2 \boldsymbol{h}_m^d)), \quad (5)$$

where $\boldsymbol{v}$, $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are model parameters, $\tanh$ is the hyperbolic tangent nonlinearity, and $\boldsymbol{h}_m^d$ is customarily the hidden state of the pointer network at current selection step m. The first hidden state $\boldsymbol{h}_0^d$ can be initialized from the permutation invariant set representation $\boldsymbol{s}$. In our context, this in principle enables us to sequentially select predicted clusters according to their learned target order.

[2]For a formal proof, see Section 3.1 and supplementary material of Lee et al. (2019).

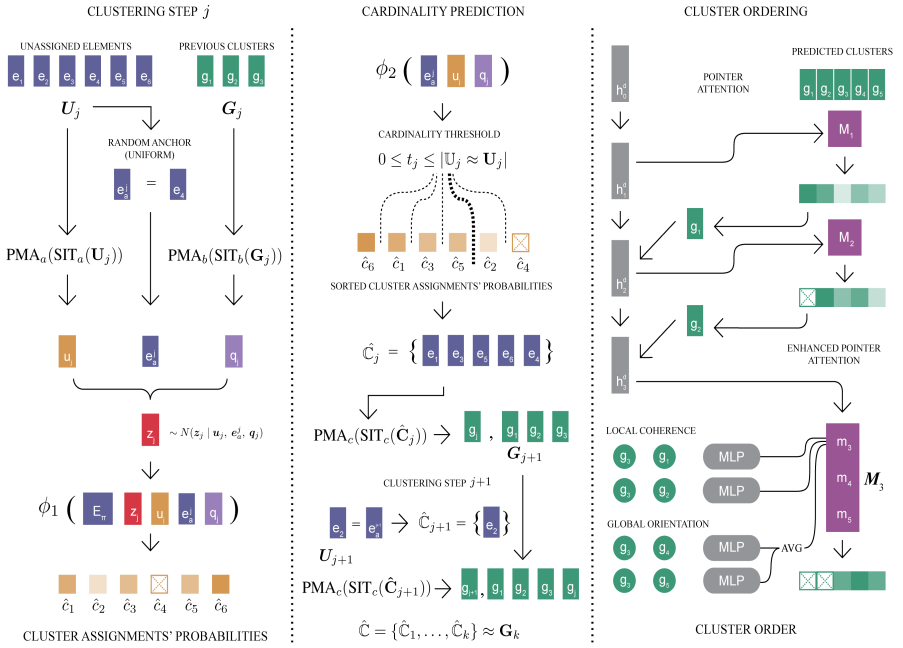


Figure 2: NOC. Starting at the top of the leftmost panel, at clustering step j the representations of unassigned elements ($\boldsymbol{U}_j$), previously created clusters ($\boldsymbol{G}_j$) and a random anchor element ($\boldsymbol{e}_a^j$) are used to make initial cluster assignments ($\hat{c}_{1-6}$). In the middle panel the current cardinality (t_j) is predicted and used to adjust the j^{th} cluster, which is then transformed via $\text{SIT}_c(\hat{\boldsymbol{C}}_j)$ into its embedded representation $\boldsymbol{g}_j$, which becomes part of the $\boldsymbol{G}_{j+1}$ matrix and is used during the remaining clustering steps. In the rightmost panel, after k iterations of the NOC_1 and NOC_2 steps, the predicted clusters are ordered via NOC_3's Enhanced Pointer attention (A.2).

4 The Neural Ordered Clusters Model

Existing methods do not, to the best of our knowledge, directly address the catalog problem of joint clustering and cluster ordering. To this end, we investigate a set of novel and adapted methods to apply to this problem. This section introduces the proposed Neural Ordered Clusters (NOC) model. NOC consists of three modular parts, each with a corresponding loss factor. These components take the form of partitional neural clustering, per-cluster cardinality prediction, and cluster ordering via pointer attention. The learned representations of elements and the set in its entirety are transformed by each of these modules and continuously adjusted during training in a fully differentiable way. For an overview of the NOC architecture, we refer the reader to Figure 2.

The **first step** is to obtain a partitional clustering (NOC_1). We propose to achieve this through an adjusted neural clustering module, building on the process described in equations 3 and 4. First, we utilize the *Set Interdependence Transformer*, or SIT (Jurewicz & Derczynski, 2022), to obtain both the representations of the individual elements ($\boldsymbol{E}_\pi$) and the permutation-invariant representation of the entire set ($\boldsymbol{s}$). SIT consists of a stack of transformer attention layers described in equations 1 and 2, except that the second layer's input takes the form of an augmented matrix, in which the vector representation of the set is concatenated to $\boldsymbol{E}_\pi$ as if it was an additional set element $\boldsymbol{e}_i$. This is intended to enable learning of higher-order interactions in fewer layers. At each cluster prediction step j the representations of unassigned elements ($\boldsymbol{u}_j$) and all previously completed clusters ($\boldsymbol{q}_j$) are obtained through a stack of SIT and PMA transformations and used to make cluster assignments $\hat{c}_i$ per unassigned element ($\mathbf{e}_i$):

$$\text{NOC}_1(\mathbf{e}_i, \mathbb{X}_j) = p_\theta(\hat{c}_i = j \mid \mathbb{X}_j) = \text{sigmoid}(\text{concat}(\phi_1(\mathbf{e}_i, \mathbf{e}_a^j, \mathbf{z}_j, \mathbf{u}_j, \mathbf{q}_j))). \tag{6}$$

The **second step** (NOC_2) is to adjust the cluster assignments via the predicted cardinality t_j of the j-th cluster. At each step a function, parameterized by a fully-connected neural network ϕ_2, is used to predict the cardinality of the current cluster as a regression task. The obtained cardinality, conditioned on the available elements and previously predicted clusters, is used as a threshold for the maximum number of elements to assign to current cluster. If the number of elements assigned by NOC_1 to $\hat{\mathbb{C}}_j$ exceeds this threshold, the elements with lower values of the predicted probabilities $p_\theta(\hat{c}_1 = j) = p_\theta(\boldsymbol{e}_1 \in \hat{\mathbb{C}}_j) \approx \hat{c}_i$ of the i^{th} element belonging to j^{th} cluster are excluded from it:

$$t_j = \text{NOC}_2(\hat{\mathbb{C}}_j, \mathbb{X}_j) = \phi_2(\text{concat}(\boldsymbol{e}_a^j, \boldsymbol{u}_j, \boldsymbol{q}_j)), \tag{7}$$

$$\hat{\mathbb{C}}_j = \begin{cases} \hat{\mathbf{C}}^j, & \text{if } |\hat{\mathbb{C}}_j| \leq t_j \\ \hat{\mathbf{C}}^j_{1:t_j}, & \text{otherwise} \end{cases}, \tag{8}$$

Steps one and two are repeated until we have obtained k partitional clusters $(\hat{\mathbb{C}}_1, \ldots \hat{\mathbb{C}}_k)$ with individual cardinalities. Set-to-sequence methods expect fixed-length vector representations, therefore SIT and PMA are used to pool each predicted cluster ($\text{PMA}_c(\text{SIT}_c(\hat{\mathbb{C}}_j)) = \boldsymbol{g}_j$). In the **third and final stage** of NOC_3 an Enhanced Pointer Network (Yin et al., 2020) is used to output an attention vector $\boldsymbol{a}_m$ at each step $m \in \{1, \ldots, k\}$. The highest attention value points to the cluster to be placed at m-th position in the output sequence of ordered clusters:

$$\boldsymbol{a}_m = \text{softmax}(\boldsymbol{v}^\top \tanh(\boldsymbol{W}_1 \boldsymbol{M}_m + \boldsymbol{W}_2 \boldsymbol{h}_m^d)) \; ; \; \boldsymbol{h}_m^d = \text{LSTM}(\boldsymbol{h}_{m-1}^d, \boldsymbol{g}_{m-1}). \tag{9}$$

This largely resembles the process outlined in Equation 5, with the exception of matrix $\boldsymbol{M}_m$, specific to the Enhanced Pointer Network, explained in more detail in appendix A.2. Together, these three elements of NOC allow for the prediction of an input-dependent number k of partitional clusters with varying, learned cardinalities. This learning is achieved through a weighted sum of the loss factors from each of the three stages of NOC, with teacher-forcing (Williams & Zipser, 1989). The full algorithm is provided in Appendix A.1.

5 Experiments

The Catalog Problem presents an interesting type signature, where while the input — as in S2S — is an unordered set, the output is a more complex structure that is the result of clustering and ordering. In this section we examine multiple approaches to the Catalog Problem, including baseline methods adapted to this output structure as well the NOC model, evaluating over both synthetic and real-world datasets. All datasets, hyperparameters and code are freely available and described in detail in Appendix A.5. The provided code includes all data pre-processing and generation steps.

The models' exact layer dimensions are given in Appendix A.5, with the number of learnable parameters of each model varying by less than 5% per task. The AdamW (Loshchilov & Hutter, 2017) optimizer was used with weight decay coefficient 1e-3, learning rate (α) 1e-4, dropout rate of 0.05 and batch size 64, for 50–100 epochs. Experiments were performed on cloud-based GPU instances, with NVIDIA Quadro P6000 graphics cards (24 GB) and 8 CPU cores. To represent natural language entities in Section 5.4 we use the concatenated and averaged output of the last 4 layers of the cased, large version of BERT (Devlin et al., 2019), frozen during training to isolate the effect of compared clustering and permutation methods on the final performance.

The best performance is reported in **bold** and second best is underlined. Reported results are averaged over three full training runs, standard deviation is reported after the ± sign. We use V-Measure (Rosenberg & Hirschberg, 2007) and Kendall's Rank Correlation Coefficient (τ) as the primary clustering and permutation metrics respectively, scaled by a factor of a hundred for readability, following convention (Wang & Wan, 2019; Pandey & Chowdary, 2020).

Table 1: Clustering and permutation results on all three datasets

	2D Gaussians		Procedural Catalogs		PROCAT	
Method	V-Measure	Kendall's τ	V-Measure	Kendall's τ	V-Measure	Kendall's τ
NCP + S2S	91.52 ± 3.30	75.31 ± 4.5	63.12 ± 4.12	74.82 ± 5.1	25.42 ± 5.14	21.94 ± 4.3
CCP + S2S	93.94 ± 2.13	83.88 ± 4.2	79.41 ± 3.76	81.10 ± 3.9	37.41 ± 3.10	25.24 ± 4.0
ACP + S2S	96.63 ± 1.82	90.13 ± 3.7	87.66 ± 3.91	85.73 ± 3.2	41.38 ± 3.88	31.73 ± 3.1
S2S-B	89.37 ± 4.21	95.89 ± 2.3	78.39 ± 1.64	92.13 ± 2.0	39.01 ± 3.35	44.39 ± 3.7
S2S-C	92.45 ± 2.01	93.41 ± 2.1	75.83 ± 4.91	91.55 ± 3.3	36.71 ± 4.26	40.22 ± 4.2
NOC	**97.81 ± 0.92**	**98.40 ± 0.5**	**96.13 ± 1.28**	**95.84 ± 0.9**	**52.84 ± 3.15**	**56.67 ± 2.8**

5.1 Baselines

We present two groups of baselines for addressing the Catalog Problem. **i) Neural clustering methods with an added set-to-sequence module:** the module takes the predicted clusters and outputs their order via attention-based pointing. These methods include the pointwise Neural Clustering Process (NCP), the Clusterwise Clustering Process (CCP) and the Attentive Clustering Process (ACP) developed by Pakman et al. (2020) and Wang et al. (2021). **ii) Proposed variants of the set-to-sequence architecture:** these S2S variants enhance the pointer mechanism with the notion of predicting ordered *clusters*, as opposed to ordered *elements*. The first variant, called S2S-B (for "break"), adds a secondary decision of whether or not to start a new cluster in parallel to the selection of the set element to be placed next in the predicted sequence. The second variant, called S2S-C (for "clusterwise"), uses a threshold mechanism to select multiple elements forming a single cluster at each step. For details, see Appendix A.4.

5.2 Ordered Mixtures of Gaussians

This dataset consists of 2D coordinates for a number of points, generated from a mixture of a finite number of Gaussian distributions. The points should be clustered and the clusters ordered by distance from origin (as they are in the supervision target). Following convention from probabilistic models for clustering (McLachlan & Basford, 1988), we introduce a random variable c_i signifying the cluster to which each data point x_i is assigned. The generation process creates a random number of clusters k, each with their own parameter vector $\boldsymbol{\mu}_j$ controlling the distribution of the j-th cluster. For comparison with prior work (Pakman et al., 2020), we use a Chinese Restaurant Process with a single modification — the addition of a target order of the clusters, based on the Euclidean distance of their *centroids* from the origin point. An example of the joint prediction of per-element cluster assignments and predicted cluster order can be seen in Figure 3. The predicted order is denoted through colour gradient, with a bright red to deep blue and violet scale. In the figure, three separate predictions are displayed, one from the ACP model, one from a modification of set-to-sequence methods in the form of S2S-B and finally one from the proposed NOC model.

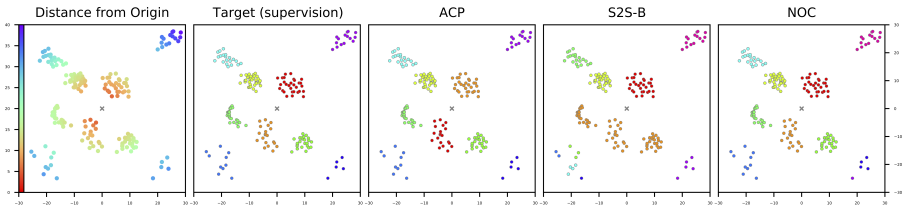


Figure 3: Example of predicted ordered clusters. Target (supervision) shows clusters and their order through colour, red being closest to the origin point (marked with a gray ×), dark blue and violet being furthest. Heat map (leftmost) indicates distance for individual points. The ACP prediction exhibits good clustering, but errs in the ordering (mistaken red and orange clusters). S2S-B exhibits good ordering, but incorrect clustering in the bottom-left quarter. NOC (ours) is closest to the target.

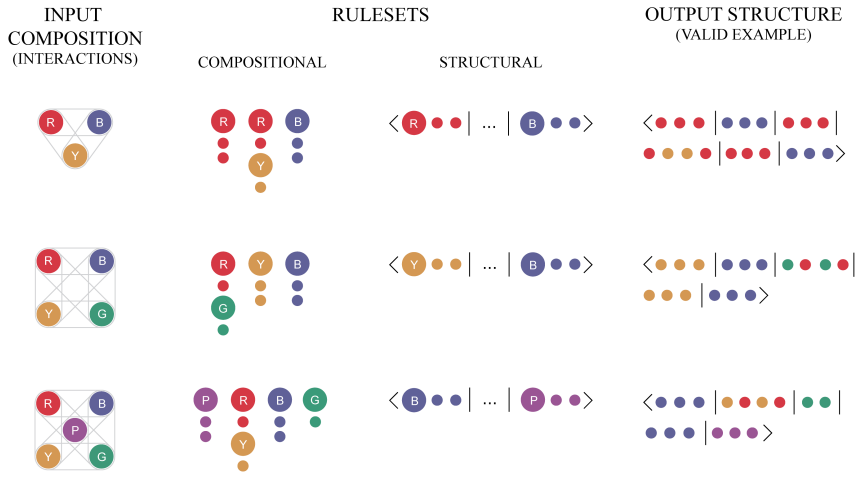


Figure 4: Procedurally generated synthetic catalogs. Interactions between elements of the input (left) define the compositional and structural rules (middle), which inform the generation of these synthetic datasets. The actual input is a multiset of n offer tokens, the leftmost panel shows only which types of tokens were present in it. Compositional rules define valid sections, structural rules define valid section order, which in the figure is represented only by what section should be first and last. A successful model should learn these rules from supervised exposure to the resulting synthetic datasets, and then be able to order new sets of elements according to the learned rules. One valid example is given for each input (right, wrapped over 2 lines). See the second paragraph of Section 5.3 for a written description of the compositional and structural ruleset portrayed in the top row.

As shown in the rightmost column of Table 1, NOC outperforms other methods on both the clustering task, according to V-Measure, and the cluster ordering task, measured with Kendall's τ. Specifically it improves by +1.18 points over the second-best clustering method (ACP) and +2.51 over the second-best set-to-sequence method (S2S-B). Its performance appears relatively consistent, showing a smaller standard deviation over three full training runs.

5.3 Procedurally Generated Catalogs

The second experiment uses synthetic catalogs. These catalogs consist of varying-length sequences of clusters of elements, with repetition. Elements are colour-coded. These catalogs form the supervised training targets $\boldsymbol{y}_i$, with the unordered multiset of available atomic elements forming the inputs $\mathbb{X}_i$. The correct composition of individual sections and the structure of the overall catalog, in the form of the order of its sections, depends on n-th order interactions between the input elements.

For procedural generation, these interactions are formalized as compositional (intra-cluster) and structural (inter-cluster) rules. A simplified example of a compositional rule would be: *"if the input set contains only red, blue and yellow elements, a section containing red and yellow elements in 1:1 ratio is a valid section"*. An example of a structural rule could specify that (given the same input) the catalog has to begin with an all-red section or end on an all-blue one (top row of Figure 4). Compositional rules also include upper cardinality constraints for valid sections. We use the tool provided by Jurewicz & Derczynski (2022) to generate catalogs.

As shown in Table 2, neural clustering methods appear to be better at composing valid catalog sections but struggle with ordering sections into valid catalogs. This is indicated through two metrics – a *compositional score*, which is the percentage of predicted sections that were valid in accordance

with the applicable n-th order ruleset, and a *structural score*, which is the percentage of valid predicted catalog structures (i.e. section ordering). By contrast, the adapted S2S models outperform neural clustering methods at correctly ordering sections, as measured via the structural score. NOC outperforms both methods on each of the two scores. This improvement is also reflected on the same test set in the more general but related V-Measure and Kendall's τ, shown in Table 1, where NOC surpasses the next-best models by +8.47 and +3.71 percentage points.

Among the sections predicted by neural clustering methods (NCP, CCP, ACP), the predominant error (present in 74% of invalid sections) stemmed from incorrect cardinality, even though the models correctly predict the composition (15%) and ratio (11%) of elements to include. This error occurs despite the presence of mechanisms that could, in principle, allow for the learning of max-cardinality constraints: NCP constructs clusters element-by-element, further transforming the candidate cluster at each element's addition; CCP and ACP obtain a representation of the current candidate cluster before assigning candidate elements.

NOC overcomes this limitation through the addition of a cluster-level cardinality prediction mechanism and corresponding loss. It outperforms the second best method on the section composition task by +11.96, +13.01 and +15.65 percentage points with regards to the 3rd, 4th and 5th order relational ruleset respectively. It also performs better with regards to the structural score, offering a smaller but consistent improvement over the S2S-C and S2S-B methods by +4.54, +3.59 and +3.61 points, with respect to increasing n-th order rulesets.

Table 2: Results over procedurally generated catalogs, by n-th order relational ruleset

	Compositional score			Structural score		
Method	$n = 3$	$n = 4$	$n = 5$	$n = 3$	$n = 4$	$n = 5$
NCP + S2S	64.13 ± 3.9	55.81 ± 4.6	51.82 ± 5.2	56.49 ± 4.0	51.87 ± 5.1	49.70 ± 6.8
CCP + S2S	75.40 ± 3.2	71.49 ± 4.3	65.11 ± 4.5	70.21 ± 3.5	68.39 ± 4.7	66.55 ± 5.4
ACP + S2S	87.05 ± 1.7	81.33 ± 1.9	76.83 ± 2.2	81.09 ± 2.2	76.34 ± 3.4	73.86 ± 3.8
S2S-B	84.99 ± 0.5	82.90 ± 0.7	74.82 ± 0.6	92.33 ± 1.5	89.83 ± 2.1	87.31 ± 2.0
S2S-C	82.03 ± 1.8	78.74 ± 2.1	72.13 ± 2.4	92.49 ± 1.6	87.41 ± 2.2	85.05 ± 2.3
NOC	**99.01 ± 0.3**	**95.91 ± 0.4**	**92.48 ± 0.4**	**97.03 ± 0.9**	**93.42 ± 1.0**	**90.92 ± 1.2**

5.4 PROCAT

The final experiment was performed on the PROCAT dataset (Jurewicz & Derczynski, 2021), using its provided training and testing split. All models were trained on approximately 9K product catalogs and tested on a separate set of 2K catalogs. Unlike the benchmarks provided with the PROCAT dataset, this formulation of the task mirrors the Catalog Problem exactly, with no information about the target number of sections (clusters) being available to the models. Individual elements were transformed into vector representations via a pre-trained, frozen language model as described at the beginning of Section 5, removing its effect on the variation in performance on the downstream task. Figure 5 displays a sample catalog predicted by NOC from a PROCAT input set of product offers.

As the two rightmost columns of Table 1 show, the PROCAT structure prediction task is more difficult than the previous tasks. The best results in terms of both the clustering quality (via V-Measure) and section order (measured indirectly via Kendall's τ with regards to element order) are approximately 40% below the corresponding scores on the procedural task in its default configuration. One possible explanation stems from the existence of a higher number of reasonable substitutions for each element in any given section from the entire input set of initially available products. While also present in the procedural catalogs, this challenge becomes harder to overcome as the cardinality of the input increases from tens in the procedural case to hundreds in PROCAT.

NOC outperforms both neural clustering methods and the adjusted set-to-sequence models. While the overall pattern of neural clustering methods outperforming S2S-B and S2S-C in V-Measure is upheld, it is less pronounced (+2.37 points between ACP and S2S-B on PROCAT compared to +9.27 and +4.18 on the procedural and 2D Gaussian task respectively). The adjusted set-to-sequence models continue to outperform NCP, CCP and ACP on the ordering aspect of the task, with a margin

Figure 5: PROCAT. An example of three sequential sections predicted by the NOC as part of a larger catalog, from an input set of products from the PROCAT dataset. The prediction groups elements into complementary sections (the three pages shown above) and orders them into a rendered catalog.

of +12.66 points. NOC yields the best performance in terms of both partitional clustering and ordering, exceeding the relevant second-best methods by +11.46% and +12.28% respectively.

6 Conclusion

The posited Catalog Problem consists of learning to group elements and to order the groups. It poses a more difficult challenge than its individual components. Our work defined benchmark tasks representing this problem and presented approaches for them, including both adjusted baselines and a candidate approach, Neural Ordered Clusters (NOC). Existing neural clustering methods appear ineffective at learning cluster-level cardinality constraints. Our method offers an improvement in this area through its cardinality-prediction module. NOC outperforms adjusted S2S methods in terms of both clustering quality and accuracy of the predicted cluster order, indicating that structuring models to address adaptive ordered clustering leads to improved performance over standard S2S prediction.

Nevertheless, the complexity and fluidity of intra- and inter-cluster relations result in the Catalog Problem remaining significantly more challenging than S2S processing. We considered a predictive solution to the catalog problem, where we trained the model to yield a single "ground truth" human-generated catalog given a set of products. Future work could consider a fully generative formulation of the problem that respects an unlimited number of valid solutions for both clustering and ordering.

ETHICS STATEMENT

Given the e-commerce context of the third presented dataset, we must highlight the wider problem of *endless scroll* user interfaces in product presentation apps and social media (Lupinacci Amaral, 2020). Although the PROCAT dataset is tailored to the prediction of cluster sequences of finite lengths, we cannot rule out the possibility of extending the proposed adaptive clustering and cluster ordering models to non-finite sets. It is also in principle possible to retrain the proposed models with additional inputs such as embedded personal preferences, making the predicted catalogs tailored to specific individuals, which has previously been linked to mental health issues in relation to smartphone addiction (Noë et al., 2019).

As with many machine learning systems, the results are not perfect, and sub-optimal predictions from NOC could silently disadvantage an end-user; for example a business may produce catalogs that don't make it easier for the reader to discover relevant, cost-saving offers, or an individual may receive an inaccurate medical analysis (in the case of the hypothesized medical triage use case).

Applying this tool may impact the employment of people performing creative catalog-related tasks, and further, might not even do the task as well as them. Product catalog design is considered something of an art among its practitioners, and there may be deep interactions not clearly evinced in training data that are lost by transiting the ownership of the catalog construction task from human subject matter experts to a machine learning model. Attempting to completely replace a human at this task may lead to both unsatisfactory and marginalizing results Birhane (2021).

We do not see any direct way for the presented methods to exacerbate bias against people of a certain gender, race, sexuality, or who have other protected characteristics. However, bias inherent to the marketing decisions made by people who have designed the catalogues contained in the PROCAT dataset, will be propagated by models trained on it. Negative biases in this particular scenario include as the *pink tax* (Stevens & Shanahan, 2017). In general, learning from socially-biased data and making predictions based on it will propagate those biases Buolamwini (2017); Raji (2020).

REPRODUCIBILITY STATEMENT

In order to ensure reproducibility all code and datasets needed for repeated experiments have been made freely available, as described in detail in Appendix A.5 as part of the provided supplementary materials. The anonymized code repository includes a comprehensive readme.md file describing the necessary steps to set up the execution environment, download, generate and preprocess the datasets and run each of the experiments discussed in Section 5. The exact hyperparameters per experiment are stated both in the Appendix (A.5.1, A.5.2, A.5.3) and in the provided configuration files in the linked code repository. Additionally, a detailed description of the NOC algorithm is provided (1) to ensure that the method can be reimplemented, if necessary.

REFERENCES

Elie Aljalbout, Vladimir Golkov, Yawar Siddiqui, Maximilian Strobel, and Daniel Cremers. Clustering with deep learning: Taxonomy and new methods. *arXiv preprint arXiv:1801.07648*, 2018.

Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. *ACM Sigmod record*, 28(2):49–60, 1999.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Daniel Bakkelund. Order preserving hierarchical agglomerative clustering. *Machine Learning*, 111 (5):1851–1901, 2022.

Abeba Birhane. The impossibility of automating ambiguity. *Artificial Life*, 27(1):44–61, 2021.

Albert Buchard, Baptiste Bouvier, Giulia Prando, Rory Beard, Michail Livieratos, Dan Busbridge, Daniel Thompson, Jonathan Richens, Yuanzhao Zhang, Adam Baker, et al. Learning medical triage from clinicians using deep q-learning. *arXiv e-prints*, pp. arXiv–2003, 2020.

J Buolamwini. Limited vision: The undersampled majority. In *AI Now*. MIT Media Lab, 2017.

Huiwen Chang, Fisher Yu, Jue Wang, Douglas Ashley, and Adam Finkelstein. Automatic triage for a photo series. *ACM Transactions on Graphics (TOG)*, 35(4):1–10, 2016.

Kenneth C Chu. Applications of artificial intelligence to chemistry. use of pattern recognition and cluster analysis to determine the pharmacological activity of some organic compounds. *Analytical chemistry*, 46(9):1181–1187, 1974.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proc. NAACL*, 2019.

Jon Ergun, Zhili Feng, Sandeep Silwal, David P Woodruff, and Samson Zhou. Learning-augmented k-means clustering. *arXiv preprint arXiv:2110.14094*, 2022.

Jeremy U Espino, William R Hogan, and Michael M Wagner. Telephone triage: a timely data source for surveillance of influenza-like diseases. In *AMIA Annual Symposium Proceedings*, volume 2003, pp. 215. American Medical Informatics Association, 2003.

Daniel Fernández-González. Multitask pointer network for multi-representational parsing. *Knowledge-Based Systems*, 236:107760, 2022.

Douglas Fisher, Ling Xu, and Nazih Zard. Ordering effects in clustering. In *Machine Learning Proceedings 1992*, pp. 163–168. Elsevier, 1992.

Douglas H Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine learning*, 2(2):139–172, 1987.

Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

Chao Gan, Ruida Zhou, Jing Yang, and Cong Shen. Cost-aware cascading bandits. *IEEE Transactions on Signal Processing*, 68:3692–3706, 2020.

Liuhao Ge, Yujun Cai, Junwu Weng, and Junsong Yuan. Hand pointnet: 3d hand pose estimation using point sets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8417–8426, 2018.

Roger Girgis, Florian Golemo, Felipe Codevilla, Martin Weiss, Jim Aldon D'Souza, Samira Ebrahimi Kahou, Felix Heide, and Christopher Pal. Latent variable sequential set transformers for joint multi-agent motion prediction. In *International Conference on Learning Representations*, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Melvin F Janowitz. An order theoretic model for cluster analysis. *SIAM Journal on Applied Mathematics*, 34(1):55–72, 1978.

Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

Stephen C Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

Mateusz Jurewicz and Leon Derczynski. PROCAT: Product catalogue dataset for implicit clustering, permutation learning and structure prediction. In *Proceedings of the conference on Neural Information Processing Systems: Datasets and Benchmarks Track*, 2021.

Mateusz Jurewicz and Leon Derczynski. Set interdependence transformer: Set-to-sequence neural networks for permutation learning and structure prediction. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2022.

Katharyn Kennedy, Richard V Aghababian, Lucille Gans, and C Phuli Lewis. Triage: techniques and applications in decisionmaking. *Annals of emergency medicine*, 28(2):136–144, 1996.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

Joon-myoung Kwon, Youngnam Lee, Yeha Lee, Seungwoo Lee, Hyunho Park, and Jinsik Park. Validation of deep-learning-based triage and acuity score using a large national dataset. *PloS one*, 13(10):e0205836, 2018.

Mai Le, Detlef Nauck, Bogdan Gabrys, and Trevor Martin. Sequential clustering for event sequences and its impact on next process step prediction. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pp. 168–178. Springer, 2014.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer. In *International Conference on Machine Learning*, 2019.

Seon-Ho Lee and Chang-Su Kim. Deep repulsive clustering of ordered data based on order-identity decomposition. In *International Conference on Learning Representations*, 2020.

Yunfan Li, Peng Hu, Zitao Liu, Dezhong Peng, Joey Tianyi Zhou, and Xi Peng. Contrastive clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 8547–8555, 2021.

Yibo Lin, Meng Li, Yuki Watanabe, Taiki Kimura, Tetsuaki Matsunawa, Shigeki Nojima, and David Z Pan. Data efficient lithography modeling with transfer learning and active data selection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(10): 1900–1913, 2018.

Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. In *CoRR, abs/1711.05101, 2017.*, 2017.

Ludmila Lupinacci Amaral. 'absentmindedly scrolling through nothing': liveness and compulsory continuous connectedness in social media. *Media, Culture & Society*, 43:016344372093945, 07 2020. doi: 10.1177/0163443720939454.

Geoffrey J McLachlan and Kaye E Basford. *Mixture models: Inference and applications to clustering*, volume 38. M. Dekker New York, 1988.

Jamie Miles, Janette Turner, Richard Jacques, Julia Williams, and Suzanne Mason. Using machine-learning risk prediction models to triage the acuity of undifferentiated patients entering the emergency care system: a systematic review. *Diagnostic and prognostic research*, 4(1):1–12, 2020.

Beryl Noë, Liam D Turner, David EJ Linden, Stuart M Allen, Bjorn Winkens, and Roger M Whitaker. Identifying indicators of smartphone addiction through user-app interaction. *Computers in human behavior*, 99:56–65, 2019.

Ari Pakman, Yueqi Wang, Catalin Mitelut, JinHyung Lee, and Liam Paninski. Neural clustering processes. In *International Conference on Machine Learning*, pp. 7455–7465. PMLR, 2020.

Divesh Pandey and C Ravindranath Chowdary. Modeling coherence by ordering paragraphs using pointer networks. *Neural Networks*, 126:36–41, 2020.

Xi Peng, Yunfan Li, Ivor W Tsang, Hongyuan Zhu, Jiancheng Lv, and Joey Tianyi Zhou. Xai beyond classification: Interpretable neural clustering. *Journal of Machine Learning Research*, 23 (6):1–28, 2022.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.

Philippe Rabiller, Peter Boles, and Boualem Boashash. Ordered clustering: A way to simplify analysis of multichannel signals. In *10th International Conference on Information Science, Signal Processing and their Applications (ISSPA 2010)*, pp. 237–242. IEEE, 2010.

Deborah Raji. How our data encodes systematic racism. *MIT Technology Review*, 10, dec 2020.

Meitar Ronen, Shahaf E Finder, and Oren Freifeld. Deepdpm: Deep clustering with an unknown number of clusters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9861–9870, 2022.

Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, pp. 410–420, 2007.

Chenxu Shi. Pointer network solution pool: Combining pointer networks and heuristics to solve tsp problems. In *2022 3rd International Conference on Computer Vision, Image and Deep Learning & International Conference on Computer Engineering and Applications (CVIDL & ICCEA)*, pp. 1236–1242. IEEE, 2022.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015.

Jennifer L Stevens and Kevin J Shanahan. Structured abstract: Anger, willingness, or clueless? understanding why women pay a pink tax on the products they consume. In *Creating Marketing Magic and Innovative Future Marketing Trends*, pp. 571–575. Springer, 2017.

Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. Learning deep representations for graph clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

Krzysztof Turowski, Jithin K Sreedharan, and Wojciech Szpankowski. Temporal ordered clustering in dynamic networks. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pp. 1349–1354. IEEE, 2020.

Stijn Marinus Van Dongen. *Graph clustering by flow simulation*. PhD thesis, Utrecht University, 2000.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Daniel Vial, Sujay Sanghavi, Sanjay Shakkottai, and R Srikant. Minimax regret for cascading bandits. *arXiv preprint arXiv:2203.12577*, 2022.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

James Vlasblom and Shoshana J Wodak. Markov clustering versus affinity propagation for the partitioning of protein interaction graphs. *BMC bioinformatics*, 10(1):1–14, 2009.

Daniel Wallach and Bruno Goffinet. Mean squared error of prediction as a criterion for evaluating and comparing system models. *Ecological modelling*, 44(3-4):299–306, 1989.

Tianming Wang and Xiaojun Wan. Hierarchical Attention Networks for Sentence Ordering. *AAAI*, 33:7184–7191, 2019. ISSN 2159-5399. doi: 10.1609/aaai.v33i01.33017184.

Yueqi Wang, Yoonho Lee, Pallab Basu, Juho Lee, Yee Whye Teh, Liam Paninski, and Ari Pakman. Amortized probabilistic detection of communities in graphs. *arXiv preprint arXiv:2010.15727*, 2021.

Jiaqi Wen. Spectral-pointer network: Pre-sort leads the pointer network to elude the tsp vortex. In *2022 3rd International Conference on Computer Vision, Image and Deep Learning & International Conference on Computer Engineering and Applications (CVIDL & ICCEA)*, pp. 578–586. IEEE, 2022.

Hannah Wenzel, Daniel Smit, and Saskia Sardesai. A literature review on machine learning in supply chain management. In *Artificial Intelligence and Digital Transformation in Supply Chain Management: Innovative Approaches for Supply Chains. Proceedings of the Hamburg International Conference of Logistics (HICL), Vol. 27*, pp. 413–441. Berlin: epubli GmbH, 2019.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Yongjing Yin, Fandong Meng, Jinsong Su, Yubin Ge, Lingeng Song, Jie Zhou, and Jiebo Luo. Enhancing pointer network for sentence ordering with pairwise ordering predictions. In *AAAI*, volume 34, pp. 9482–9489, 2020.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 16259–16268, 2021.

Yutao Zhu, Kun Zhou, Jian-Yun Nie, Shengchao Liu, and Zhicheng Dou. Neural sentence ordering based on constraint graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 14656–14664, 2021.

A APPENDIX

A.1 NOC ALGORITHM

In this section of the appendix we outline the progression over all three stages of the proposed Neural Ordered Clusters method. Steps 1-23 jointly describe the processing within NOC_1 and NOC_2 (steps 12-16 specifically for the latter), as presented in Section 4. The third module, NOC_3 is shown in steps 24-30. We begin with an unordered set $\mathbb{X} \in \mathbb{R}^d$ (of any cardinality), and assume that it has at least two elements, which can then potentially belong to separate clusters. This set is represented as a matrix of d-dimensional elements, ordered according to some arbitrary permutation π into $\boldsymbol{X}_\pi$. The initial, intermediate output comes in the form of individual clusters of elements at each j-th iteration, which ultimately form the set of all predicted clusters ($\hat{\mathbb{C}} = \{\hat{\mathbb{C}}_1, \ldots, \hat{\mathbb{C}}_k\}$). Each candidate cluster goes through a final cardinality prediction step, resulting in the threshold value of t_j, through which some elements may be excluded from their original cluster. Finally, an Enhanced Pointer Network (EPN) performs $k-1$ iterations, selecting a single cluster to be placed next in the final output sequence $\hat{\boldsymbol{y}}$ by the index of the highest value in the predicted attention vector $\boldsymbol{a}_m$.

Algorithm 1 Neural Ordered Clusters

Require: $|\mathbb{X}| = n \geq 2$ ▷ At least two elements, otherwise single cluster
1: $\boldsymbol{E}_\pi \leftarrow \text{SIT}(\boldsymbol{X}_\pi \sim \mathbb{X}),\ j \leftarrow 1$
2: $r \leftarrow n - 1$ ▷ Track number of unassigned elements
3: $\boldsymbol{e}_a^j \leftarrow \boldsymbol{E}_\pi$ ▷ Randomly chosen anchor for initial cluster
4: $\boldsymbol{U}_j \leftarrow \text{SIT}(\mathbb{E} \setminus \{\boldsymbol{e}_j^a\})$ ▷ Initialize unassigned representations
5: $\boldsymbol{u}_j \leftarrow \text{PMA}(\boldsymbol{U}_j)$ ▷ Vector representation of all unassigned elements
6: $\boldsymbol{q}_j \leftarrow \varnothing$ ▷ No previous clusters
7: **while** $r > 1$ **do**
8: $\quad \boldsymbol{z}_j \leftarrow \mathcal{N}(\boldsymbol{z}_j \mid \mu(\boldsymbol{e}_j^a, \boldsymbol{u}_j, \boldsymbol{q}_j), \sigma(\boldsymbol{e}_j^a, \boldsymbol{u}_j, \boldsymbol{q}_j))$
9: $\quad$ **for** $i \leftarrow 1 \ldots r$ **do**
10: $\quad\quad \hat{c}_i \leftarrow \phi_1(\boldsymbol{e}_i, \boldsymbol{e}_a^j, \boldsymbol{z}_j, \boldsymbol{u}_j, \boldsymbol{q}_j)$ ▷ j-th cluster assignments per element
11: $\quad$ **end for**
12: $\quad \hat{\boldsymbol{C}}_j \leftarrow \boldsymbol{E}^\pi_{i:\hat{c}_i = j}$ ▷ Cluster j from assignments (sorted)
13: $\quad t_j \leftarrow \phi_2(\boldsymbol{e}_a^j, \boldsymbol{u}_j, \boldsymbol{q}_j)$ ▷ Predict cluster cardinality
14: $\quad$ **if** $|\hat{\mathbb{C}}_j| \leq t_j$ **then**
15: $\quad\quad \hat{\mathbb{C}}_j \leftarrow \hat{\boldsymbol{C}}_j$
16: $\quad$ **else**
17: $\quad\quad \hat{\mathbb{C}}_j \leftarrow \hat{\boldsymbol{C}}_{1:t_j}$ ▷ Adjust j-th cluster's cardinality
18: $\quad$ **end if**
19: $\quad j \leftarrow j + 1$
20: $\quad \boldsymbol{e}_a^j \leftarrow \boldsymbol{E}_\pi$ ▷ Randomly chosen anchor for next cluster
21: $\quad \boldsymbol{U}_j \leftarrow (\mathbb{U}_{j-1} \setminus \hat{\mathbb{C}}_j)$ ▷ Update unassigned representations
22: $\quad \boldsymbol{g}_{j-1} \leftarrow \text{PMA}(\text{SIT}(\hat{\boldsymbol{C}}_{j-1}))$ ▷ Vector representation of previous cluster $\hat{\mathbb{C}}_{j-1}$
23: $\quad \boldsymbol{q}_j \leftarrow \hat{\mathbb{C}} \approx \{\boldsymbol{g}_1, \ldots, \boldsymbol{g}_{j-1}\}$ ▷ Update preceding clusters' representations
24: $\quad r \leftarrow r - |\hat{\mathbb{C}}_{j-1}| - 1$ ▷ Adjust number of unassigned elements
25: **end while**
26: $\boldsymbol{h}_1^d \leftarrow \text{PMA}(\text{SIT}(\boldsymbol{G}_j \approx \hat{\mathbb{C}}))$ ▷ First hidden state from all clusters
27: $\hat{\boldsymbol{y}} = (\varnothing_1, \ldots, \varnothing_j)$ ▷ Final prediction placeholder
28: **for** $m \leftarrow 1 \ldots k-1$ **do**
29: $\quad \boldsymbol{a}_m, \boldsymbol{h}_{m+1}^d \leftarrow \text{EPN}(\boldsymbol{G}_j, \boldsymbol{h}_m^d)$ ▷ Enhanced pointer attention over k predicted clusters
30: $\quad l \leftarrow \arg\max(\boldsymbol{a}_m)$
31: $\quad \hat{y}_m = \hat{\mathbb{C}}_l$ ▷ Next cluster by highest attention index
32: **end for**

A.2 ENHANCED POINTER NETWORK

In all reported experiments we use the same set-to-sequence module, the Enhanced Pointer Network Yin et al. (2020), which is a pointer-attention based method inspired by the popular Pointer Network

Vinyals et al. (2015). It offers a performance improvement by leveraging two additional mechanisms for pairwise ordering predictions towards improved global and local coherence of the output sequence. Formally, the conditional probability of a predicted order $\hat{\boldsymbol{y}}$ is calculated as:

$$p_\theta(\hat{\boldsymbol{y}} \mid \mathbb{C}) = \prod_{j=1}^{k} p_\theta(\hat{y_j} \mid \hat{\boldsymbol{y}}_{<j}, \boldsymbol{G}_k, \boldsymbol{s}_c) \; ; \; \boldsymbol{s}_c = \text{PMA}(\text{SIT}(\boldsymbol{G}_k)) \tag{10}$$

$$p_\theta(\hat{y_j} \mid \hat{\boldsymbol{y}}_{<j}, \mathbb{C}) = \text{softmax}(\boldsymbol{v}^\top \tanh(\boldsymbol{W}_1 \boldsymbol{h}_j^d + \boldsymbol{W}_2 \boldsymbol{M}_j)) \tag{11}$$

$$\boldsymbol{h}_j^d = \text{LSTM}(\boldsymbol{h}_{j-1}^d, \boldsymbol{g}_{j-1}) \; , \; \boldsymbol{h}_0^d = \boldsymbol{s}_c \tag{12}$$

where $\boldsymbol{v}$, $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are model parameters, k is the total number of clusters, tanh is the hyperbolic tangent nonlinearity, $\boldsymbol{g}_{j-1}$ is the fixed-length embedding of the cluster selected at the preceding step $j-1$ and $\boldsymbol{h}_j^d$ is the hidden state of the permutation module at current step i. The first hidden state $\boldsymbol{h}_0^d$ is initialized from the permutation invariant set representation of all previously predicted clusters $\boldsymbol{s}_c$, obtained via SIT and PMA. The $\boldsymbol{M}_j$ matrix provides additional context consisting of 2 kinds of information. The first is global orientation relating all remaining unordered clusters to one another. The second is local coherence between previously selected clusters and remaining candidates. This contextual information is obtained via HISTORY and FUTURE sub-modules from the original matrix of all cluster representations ($\boldsymbol{G}_k \approx \mathbb{C}$), which form the elements to be ordered. These two sub-modules output pairwise ordering predictions in relation to each candidate cluster, which are then combined to form $\boldsymbol{M}_j$. For exact implementation details, we refer the reader to Yin et al. (2020), but also provide more detail regarding these two sub-modules below.

The FUTURE sub-module calculates the probability of every currently (at j^{th} decoder step) unordered element x_u appearing before and after every other unordered element $x_{u\prime}$, denoted here as $p_\theta(\textit{before}, \textit{after} \mid x_u, x_{u\prime})$, which is represented by a softmaxed vector consisting of two elements ($\boldsymbol{p}_{u,u\prime} \in \mathbb{R}^2$) obtained in the following way:

$$\boldsymbol{z}_{u,u\prime} = \text{ReLU}(\boldsymbol{W}_c \times \text{ReLU}(\boldsymbol{W}_a \boldsymbol{e}_u + \boldsymbol{W}_b \boldsymbol{e}_{u\prime})), \tag{13}$$

$$\boldsymbol{p}_{u,u\prime} = \text{softmax}(\boldsymbol{W}_d \times \boldsymbol{z}_{u,u\prime}), \tag{14}$$

where $\times$ denotes matrix multiplication, ReLU() is the rectified linear unit nonlinear activation function, $\boldsymbol{e}_u$ and $\boldsymbol{e}_{u\prime}$ are the embedded representations of the unordered element x_u and $x_{u\prime}$ respectively (obtained by the encoder). Bias terms have been omitted for readability. All pairs of unordered elements are processed in this way and then transformed into a single vector $\boldsymbol{m}_{f,u}$, per unordered element:

$$\boldsymbol{m}_{f,u} = \frac{1}{|\mathbb{X}_u|} \left(\sum_{x_{u\prime} \in \mathbb{X}_u} [\boldsymbol{z}_{u,u\prime} \mid \boldsymbol{p}_{u,u\prime}] \right), \tag{15}$$

where $\mathbb{X}_u$ denotes the set of unordered elements except x_u and $[\boldsymbol{z}_{u,u\prime} \mid \boldsymbol{p}_{u,u\prime}]$ is a concatenation of the two listed vectors. This enables the model to exploit global relative orientation of other unordered elements when considering x_u.

The HISTORY sub-module performs analogous transformations for each unordered element x_u, but this time with regards to two previously ordered elements x_{j-1} and x_{j-2} (placed in the output sequence at the $j-1^{\text{th}}$ and $j-2^{\text{th}}$ index during preceding decoder steps). For x_{j-1} this takes the form of:

$$\boldsymbol{z}_{u,j-1} = \text{ReLU}(\boldsymbol{W}_c \times \text{ReLU}(\boldsymbol{W}_a \boldsymbol{e}_u + \boldsymbol{W}_b \boldsymbol{e}_{j-1})), \tag{16}$$

$$\boldsymbol{p}_{u,j-1} = \text{softmax}(\boldsymbol{W}_d \times \boldsymbol{z}_{u,j-1}). \tag{17}$$

The $\boldsymbol{z}_{u,j-1}$ and $\boldsymbol{p}_{u,j-2}$ vectors are obtained analogously, but through different learned $\boldsymbol{W}_*$ weight matrices (which the authors of the method justify by the fact that they refer to a relative distance of two places, instead of a single one). These are then used to obtain the $\boldsymbol{m}_{h1}$ and $\boldsymbol{m}_{h2}$ vectors by concatenating the corresponding $\boldsymbol{z}$ and $\boldsymbol{p}$ vectors, which are taken to encode left-side, local coherence (local referring to the fact that it's just two elements to the left of the current candidate that are being considered, as opposed to the entire left-side context encoded in the decoder's hidden state $\boldsymbol{h}_j^d$). Finally, the $\boldsymbol{m}_{f,u}$ vector (obtained by the FUTURE sub-module) and the $\boldsymbol{m}_{h1}$ and $\boldsymbol{m}_{h2}$ vectors (obtained by the HISTORY module) are concatenated into the $\boldsymbol{m}_u$ vector, which encodes both relative orientations of other unordered elements with respect to x_u and measures local coherence between two previously ordered elements and x_u. Such $\boldsymbol{m}_u$ vectors are obtained for all unordered elements at each j^{th} decoder step and packed into the $\boldsymbol{M}_j$ matrix.

A.3 NOC Training

NOC is trained in a teacher-forced way through a combined, weighted loss. The first loss factor comes from NOC_1 and is equivalent to the clustering loss of CCP (Pakman et al., 2020). At every j^{th} clustering step NOC_1 outputs a vector of cluster assignment probabilities for all unassigned set elements ($\boldsymbol{o}_j = (o_j^1, \ldots, o_j^m)$, where m is the number of remaining elements, which have not been assigned to any of the preceding clusters):

$$o_j^i = \text{NOC}_1(\boldsymbol{e}_i, \mathbb{X}_j) = p_\theta(x_i \in \hat{\mathbb{C}}_j) = p_\theta(\hat{c}_i = j), \tag{18}$$

where $\mathbb{X}_j$ represents the input set $\mathbb{X}$ at the j^{th} clustering step with regards to which elements have been assigned to which clusters, $\boldsymbol{e}_i$ is the embedded representation of the element x_i and θ represents all trainable model parameters. As per Equation 6, the $\boldsymbol{o}_j$ vector is obtained using the sigmoid activation function, making each o_j^i be a real number within the $[0, 1]$ range, which are treated as probabilities. During training the labelled examples are used to optimize an evidence lower bound (ELBO) thereof, by minimizing the expected KL divergence as described in the appendix of the paper introducing CCP (Pakman et al., 2020).

The second loss factor comes from NOC_2, which predicts a cardinality threshold for each j^{th} cluster as a positive integer ($t_j \in \mathbb{Z}^+$). This is compared to the target cardinality of the current cluster $|\mathbb{C}_j|$ (identified by the anchor element's cluster assignment) to obtain the mean squared error (Wallach & Goffinet, 1989). The third loss factor comes from NOC_3 and is equivalent to the Enhanced Pointer Network's loss. Given a batch $\mathbb{B}$ of m examples of the form $(\mathbb{X}, \boldsymbol{y})$:

$$\mathcal{L}(\mathbf{y}; \theta) = -\frac{1}{m} \sum_{(\mathbb{X},\, \mathbf{y}) \,\in\, \mathbb{B}} (\log\, p_\theta(\mathbf{y} \mid \mathbf{X}_\pi) + \lambda\, \mathcal{L}_{FH}), \tag{19}$$

where θ is the set of all model parameters and λ is a hyperparameter that balances the first term of the loss with $\mathcal{L}_{FH}$, a cross-entropy loss calculated from the pairwise predictions made by these FUTURE and HISTORY sub-modules (as outlined in Appendix A.2). For a visual explanation, the reader is referred to Figure 2 from the original paper by Yin et al. (2020). These three loss factors are weighted and the combined loss is minimized using the stochastic Adam optimizer (Kingma & Ba, 2015).

A.4 Set-to-Sequence Baselines

In this subsection, a more detailed description of the proposed S2S variants is given. The **S2S-B variant** utilizes pointer attention to select individual remaining set elements at each step, following the convention of Pointer Networks Vinyals et al. (2015) and their enhancements (Yin et al., 2020). What distinguishes S2S-B from these models is an added prediction target which requires making $n - 1$ binary decisions, where n is the cardinality of the input set. At each step of the predicted permutation sequence, S2S-B indicates whether the currently selected element should be the last one of the current, open cluster. If so, this would indicate a *"break"* in the sequence, reminiscent of a page break in a product catalog. Once the last available element is reached, any remaining opened

clusters are closed by default, hence $n-1$. All previously pointed-to elements since the last break are considered members of the current open cluster.

The S2S-B model is thus capable of predicting a clustering where each element is assigned its own cluster and one where all elements belong to a single cluster. It is guaranteed to assign a cluster to every single element and can handle varying cardinality input sets, like all pointer networks. The first difficulty faced due to this particular modification stems from highly skewed class distribution. Namely, we never complete (or break) a cluster after each element. This is mitigated via a class-weighted binary cross-entropy loss function:

$$\mathcal{L}_{\text{BCE-w}}(\theta) = -\frac{1}{m}\sum_{i=1}^{m}(w_b \times y_m \times \log(\hat{y}_m) + (1-y_m) \times \log(1-\hat{y}_m) \tag{20}$$

Where m is the number of training examples, w_b is the adjusted weight for the positive class, and y_i and $\hat{y}_i$ are the target and prediction respectably. This loss factor is then scaled and added to the negative loss likelihood loss used to train the pointer selection mechanism. The main disadvantage of this model is that it predicts meaningless in-cluster order, making the loss signal noisy. The order of elements within each cluster is meaningless within the confines of the presented Catalog Problem.

To mitigate this disadvantage, a second variant was developed. Referred to as **S2S-C** (for "cluster-wise"), this model predicts the entire next cluster of elements at each step, instead of pointing to a single next element in the output sequence. Instead of performing n transformation steps in a loop over the entire input set, it outputs an attention vector over all available elements until there are none left. Thus it is also bound between assigning all elements to a single cluster or every element to its own cluster, much like S2S-B, guaranteeing cluster assignment for each element of the input set.

In order to predict clusters of adaptive, input-dependent cardinality, the formula for obtaining the pointer-attention vector over available elements had to be adjusted. The softmax operator was replaced with the sigmoid function (σ) and a threshold (t_a) of 0.5 was adopted. At each step $j \in \{1, 2 \ldots, n\}$ every element with a corresponding attention value (a_i^j) above the threshold is thus assigned to the next cluster:

$$\boldsymbol{a}_i = \sigma(\boldsymbol{v}^\top \tanh(\boldsymbol{W}_2\boldsymbol{E}_\pi + \boldsymbol{W}_1\boldsymbol{h}_i^d)) \tag{21}$$

$$\hat{y}_i^j = \begin{cases} 0, & \text{if } a_i^j < t_a \\ 1, & \text{otherwise} \end{cases} \tag{22}$$

During training, the S2S-C model was teacher-forced (Williams & Zipser, 1989) to prevent the cascading impact of incorrect initial cluster assignment on subsequent computation steps, which is a known challenge in certain areas of machine learning, such as the multi-armed bandit problem (Gan et al., 2020). This is not a departure from the other tested models (with the exception of S2S-B), as all neural clustering baselines are also teacher-forced during training, as per author implementations of the papers that originally introduced them.

A.5 Code, Datasets and Parameters

The code required for all three of the main experiments can be found in a fully anonymized repository under the following link:

`https://github.com/anonymous-paper-submissions/neural-ordered-clusters`

Follow the instructions provided in the readme.md document to set up the necessary environment locally, via the requirements.txt file listing all necessary packages and their versions.

In the following sections we describe each dataset in more detail, including how to download or generate it. All datasets are freely available under publicly accessible links. Additionally, each section contains the specific hyperparameters used for repeated experiments as well as the exact number of layers and parameters per tested NOC model.

A.5.1 ORDERED MIXTURES OF 2D GAUSSIANS

Data. The dataset for predicting ordered clusters of 2D Gaussians (based on their distance from the origin point) is synthetically generated when running the experiment via the linked run_gauss2D.py file. The full, default configuration is given in the parser arguments (nothing should require adjustment to run the equivalent experiment). This includes a default seed, which should help ensure repeatability. In the provided experiments we generate 30K batches of 64 examples each, for a total of just under 2 million individual training examples for a full run. Each example is a set of 5 to 100 individual points characterized by their coordinates, generated through the Chinese Restaurant Process with dispersion parameter α set to 0.7 for all experiments. Unlike the batch generation process used by Pakman et al. (2020), we generate batches with diverse number of clusters and cluster cardinalities in each example.

Hyperparameters. The training regimen includes a learning rate adjustment from 1e-4 to 5e-5 at the 15K-th batch and 1e-5 at the 20K-th batch. The AdamW (Loshchilov & Hutter, 2017) optimizer was used with a weight decay coefficient of 1e-3. Additionally, the default weights per loss factor are provided. The main clustering loss factor λ_c is equal to 1.0, the cluster ordering loss factor is set to λ_o = 4.0 and the cardinality prediction loss factor λ_k = 3e-3. A 100 inferences samples is generated by default during validation, final metrics being calculated for the clustering prediction with the highest probability.

Model parameters. The NOC model with reported performance had over 12mil trainable parameters. The element and individual cluster encoding functions, each consisting of three stacked SIT layers followed by a PMA layer, had the input and hidden dimensions of 128. The function pooling all clusters consisted of two stacked SIT layers followed by a PMA layer, also with 128 dimensions. The NOC_1 clustering module consistently uses a Parametric Rectified Linear Unit (PReLU) as the nonlinearity He et al. (2015).

A.5.2 PROCEDURALLY GENERATED CATALOGS

Data. The dataset for predicting the cluster composition (sections of offer tokens) and structure (order of these sections) of synthetic catalogs is automatically generated when running the linked run_synthetic.py experiment script with default parser arguments. This script loads the provided configuration file synthetic_rulesets.json which specifies all compositional and structural rulesets to which the generated synthetic catalogs will adhere. In all reported experiments we refer to this default set of rulesets, but encourage researchers to treat it as an easy-to-edit, flexible configuration that can be adjusted for other exploratory experiments.

For the experiments, we generate 300K synthetic catalogs for the training set and 75K for the validation and test sets (split into 15 data-loaders). Each example consists of 35-50 offer tokens, each batch consists of 64 examples with varied number of clusters and cluster cardinalities in each batch. The NOC model is trained over 250K batch iterations, the equivalent of 50 epochs.

Hyperparameters. The procedurally generated catalog training regimen includes a learning rate adjustment from 1e-4 to 5e-5 at the 100K-th batch iteration and 1e-5 at the 200K-th. The AdamW (Loshchilov & Hutter, 2017) optimizer was used with a weight decay coefficient of 1e-3. Additionally, the default weights per loss factor are provided. The main clustering loss factor λ_c is equal to 1.0, the cluster ordering loss factor is set to λ_o = 15.0 and the cardinality prediction loss factor λ_k = 0.1. A hundred inferences samples is generated by default during validation, final metrics being calculated for the clustering prediction with the highest probability.

Model parameters. The NOC model with reported performance had over 18mil trainable parameters. The element and individual cluster encoding functions, each consisting of four stacked SIT layers followed by a PMA layer, had the input and hidden dimensions of 128. The function pooling all clusters consisted of three stacked SIT layers followed by a PMA layer, also with 128 dimensions. The NOC_1 clustering module consistently uses a Parametric Rectified Linear Unit (PReLU) as the nonlinearity He et al. (2015).

A.5.3 PROCAT

Data. The PROCAT dataset is freely available under the following link:

https://figshare.com/articles/dataset/PROCAT_Product_Catalogue_Dataset_for_Implicit_Clustering_Permutation_Learning_and_Structure_Prediction/14709507

We follow the provided train - test split of 8K - 2K catalogs and all pre-processing steps from the original paper (Jurewicz & Derczynski, 2021). The provided section break tokens are removed in the pre-processing to enable the prediction of input-dependent number of sections. Elements are by default truncated to 512 dictionary tokens for the language-specific BERT model, available in the linked hugging face repository and the suggested max-offer threshold of 200 per catalog is followed. Batches of 64 catalogs are used. The proposed NOC model is trained for 12.5K batch-iterations, the equivalent of 100 epochs.

Hyperparameters. The PROCAT training regimen includes a learning rate adjustment from 1e-4 to 5e-5 at the 5K-th batch iteration and 1e-5 at the 10K-th. The AdamW (Loshchilov & Hutter, 2017) optimizer was used with a weight decay coefficient of 1e-3. Additionally, the default weights per loss factor are provided. The main clustering loss factor λ_c is equal to 1.0, the cluster ordering loss factor is set to λ_o = 10.0 and the cardinality prediction loss factor λ_k = 0.5. A hundred inferences samples is generated by default during validation, final metrics being calculated for the clustering prediction with the highest probability.

Model parameters. The NOC model with reported performance had 23mil trainable parameters (not including the BERT model, which was frozen during training). The element and individual cluster encoding functions, each consisting of five stacked SIT layers followed by a PMA layer, had the input and hidden dimensions of 128. The function pooling all clusters consisted of four stacked SIT layers followed by a PMA layer, also with 128 dimensions. The NOC_1 clustering module consistently uses a Parametric Rectified Linear Unit (PReLU) as the nonlinearity (He et al., 2015).

Chapter 8

Conclusion

This final chapter provides a brief summary and discussion of the main aspects of the presented research, followed by an outline of potential future work and applications to related challenges.

8.1 Summary and Discussion

This thesis began with a closer look at the industrial context of the included research. This was centered around two specific research questions, one theoretical, the other applied. Namely, the questions of how neural networks can be utilized to predict ordered, partitional clusters from sets of elements and how such methods can be applied to the problem of predicting product catalog structure from sets of available product offers. Catalog structure was defined as either a permutation or an ordered, partitional clustering of the available set elements. This, in turn, led to the formulation of the titular Catalog Problem as an umbrella term for challenges that require taking as input sets of varying cardinality and predicting an input-dependent number of ordered, partitional clusters in accordance with target preference.

The majority of the considerations related to the second research question stem from either the specifics of the Incito service, which defined the proposed models' output format, or from the reality of existing product catalogs, as evidenced by PROCAT. This curated dataset formed one of the larger initial deliverables of this project.

Incito requires a nested list of product offer ids, which it then renders into an electronic product catalog, taking into consideration the available screen size, relevant product images, length of associated text and other factors. This defined the input as varying-cardinality sets of product offers, whose features consist primarily of their textual descriptions. The target output, in turn, became an ordered sequence of subsets of product offers, representing the final catalog structure. The in-section order of product offers is disregarded by the Incito rendering service due to the scarcity of available screen space, thus there is no requirement to predict it. Examples of catalogs predicted by the proposed model and subsequently rendered by the Incito service are given in Figure 3.6.

Such and other examples of sets of product offers and their target structure are taken from the provided PROCAT dataset. Consisting of over ten thousand individual catalogs designed by human experts, this data forms a consistent mainstay of the experiments presented in included publications. Comprised of over one and a half million product offers composed into almost a quarter of a million of complementary sections, PROCAT is a substantial and diverse source of information. Spanning 15 GPC-GS1 commercial categories of products, it also provides a large corpus in an underrepresented language (Danish). To the best of my knowledge PROCAT remains the sole dataset enabling the prediction of the structure of electronic product catalogs.

Key aspects of the underlying challenge offered by the PROCAT dataset include the varied number of product offers per catalog and the complex relational nature of interactions between them, which according to subject matter experts defines the composition of sections and their final ordering. Additionally, substantial difficulty stems from having to provide not just a reasonable grouping or a prioritized list of products by their relevance, but an engaging catalog narrative. This manifests through seemingly interchangeable product offers appearing spread across multiple pages, at different places within the catalog structure. It also hints at the problem of there being many equally valid possible structures that could be predicted from

the same input set. Unfortunately, within PROCAT there is only one given target per set of product offers, defining a single way in which that corresponding catalog's content could be structured.

With these considerations in mind, two approaches to tackling the Catalog Problem were applied. The first, incomplete approach was simpler and did not address every aspect of the Catalog Problem. It is also referred to as the set-to-sequence approach, in relation to a larger field of research and corresponding family of methods, where the goal is to take a set of varying cardinality and predict a permutation of its elements. As such, this approach does not predict an input-dependent number of partitional clusters. Instead, it requires one to provide this number a priori and predicts in-section order through selecting special section-break tokens at various places in the output sequence.

The second, complete approach to the Catalog Problem described within this work came in the form of supervised neural ordered clustering. It combines elements of each of the three families of neural methods that were presented in Chapter 2. Namely, it utilizes set encoding methods to obtain meaningful, fixed-length representations of the input set. Subsequently, it uses supervised neural clustering techniques to obtain a partitional clustering of the set elements, conditioned on this embedding. Finally, this approach employs set-to-sequence, pointer attention methods to predict an order over the clusters. In principle, this respects every aspect of the Catalog Problem.

The core of the presented contributions revolves around finding ways to enhance various aspects of these two approaches. The simpler, set-to-sequence approach inspired a literature review outlining different set encoding and permutation learning methods in the field of neural networks, as included in Chapter 4. The hope is that this publication provided a comprehensive entry point to researchers interested in this and related challenges, guiding them towards the methods that are best suited to their context of application.

On top of the curated dataset of real-world product catalogs, a library for generat-

ing simpler, synthetic catalogs has been provided. Both of these are introduced in the second included article, in Chapter 5. The procedurally generated catalogs received separate attention within this work in Chapter 3.1. Together, these datasets - one real, one synthetic - presented the scientific community with data, evaluation metrics and initial benchmarks for a practical approach to the Catalog Problem. In particular, the flexible, easily customizable configuration schema that guides the generation of the aforementioned synthetic catalogs opens the door for an empirical evaluation of a given model's capacity to learn the kinds of higher-order relational rules that guided the design of catalogs from PROCAT. Furthermore, one can gain insight into model performance separately in terms of section composition and catalog structure prediction or even on a per-rule basis. Additionally, synthetic catalogs address the problem of there being only one canonical structure available per a catalog's constituent product offer set, as one can generate a large number of equally valid synthetic catalogs from the same underlying input in the form of n atomic product offer tokens.

Another contribution came in the form of the Set Interdependence Transformer (SIT), a proposed set encoding method designed to enhance the original Set Transformer's [11] ability to encode interactions between elements and the set in its entirety. This variant was inspired by the conversations with SMEs in relation to what actually determines the composition and order of sections in real-world product catalogs. The experts pointed in the direction of pairwise and higher-order interactions among the entirety of the available set. SIT appears to make learning of these interactions more effective by treating the entire set as its own element and performing transformer-style attention operations [118] over all such elements, via a chosen number of layers.

This modification appears to enable SIT to learn structural and compositional rules dependent on n^{th} order interactions among set elements in fewer than n layers, as evidenced by the performance of set-to-sequence models that employ it as their set encoding module, on a wide array of tasks. Presented in Chapter 6, these included the Travelling Salesman Problem [193], Formal Grammars [194] and the Dyck Lan-

guage [19], the sentence ordering dataset ROCStory [135] as well as the synthetic catalogs and PROCAT. In almost all cases SIT resulted in an increase in performance, suggesting its applicability to tasks where interactions among a large number of set elements play a pivotal role. However, these are purely empirical results and both their reproduction and a firm theoretical grounding of why this occurs remain as subjects for future work, as discussed in the next section.

The final contribution of the presented work came in the form of the Neural Ordered Clusters (NOC) model. Markedly the first neural architecture capable of addressing the Catalog Problem in all its aspects, NOC can take a set of any cardinality and predict an unknown number of ordered, partitional clusters, without in-section order. NOC utilizes SIT as its set encoder to obtain both the fixed-length vector representation of the initial input set, of each predicted cluster and of all the predicted clusters in their entirety (which are themselves sets of varying cardinalities). It expanded upon existing neural clustering methods by adding a per-cluster cardinality prediction mechanism, which is used to adjust the initial cluster assignments. The last stage of NOC involves set-to-sequence techniques to predict a permutation of the predicted clusters, specifically through the application of an Enhanced Pointer Network (EPN) [17].

Altogether, these elements allowed the NOC algorithm to achieve top performance on the task of ordering Mixtures of 2D Gaussians by their distance from the origin point, the synthetic catalogs and PROCAT, as presented in the final article included in Chapter 7. To my knowledge NOC is the only method capable of tackling the Catalog Problem without alterations. It outperforms both preceding neural clustering architectures combined with permutation-learning modules and two proposed modifications to set-to-sequence models that are capable of predicting and ordering an input-dependent number of partitional clusters. Thus, a trained NOC model can be used to take sets of product offers from a given PROCAT catalog and then have its prediction fed to the Incito service to generate full catalogs that render well on any

device, as shown in Figure 3.6. However, NOC is not a fully generative model, instead predicting a single ordered clustering per input set.

8.2 Future Work

There is a number of directions that would be interesting to explore in the course of future research. First significant direction could take the form of exploring alternative formulations and approaches to the Catalog Problem itself. Two such alternatives were briefly mentioned in Chapter 2. One of them would see the output of the proposed models take the form of a graph. This could be either a mixed graph [195] or a directed one, where unidirectional edges define the order of catalog sections and bidirectional edges connect elements belonging to the same section, page, cluster or community. This direction was left largely unexplored due to time constraints but there is a significant and active branch of research in the area of deep generative NNs which could provide the foundation for future work along these lines [70, 71]. Particularly the Graph Recurrent Attention Networks (GRANs) would appear to constitute an avenue of investigation into prediction of ordered communities, due to the sequential way in which they predict blocks of nodes and edges, as seen in the works of Liao *et al.* (2019), Shah and Koltun (2020) and Jin *et al.* (2020). For a recent overview with a particular focus on generative models for the related topic of molecular discovery, the work of Bilodeau *et al.* (2022) can be recommended.

Another alternative approach to the Catalog Problem comes in the form of Reinforcement Learning. RL has been successfully applied to a large number of combinatorial optimization problems [96–99], including the aforementioned Vehicle Routing Problem [100], which is a generalization of the Travelling Salesman Problem, which was used in the presented experimental work. This progress has been extended to real-world applications in such fields as transportation systems [102], traffic signal control [103] and robotics [104], further suggesting its applicability to the industrial challenge of predicting engaging product catalogs. However, RL methods generally

benefit from a clear, fine-grained reward function that would provide a learning signal at each time step and the PROCAT dataset does not appear to provide such a measure. This difficulty is related to the previously mentioned, hypothesized existence of multiple equally valid, alternative ways to compose and structure the same set of product offers into a catalog. Nonetheless, other developments in the field appear to alleviate this problem. For example, the Exploratory Combinatorial Optimization DQN (ECO-DQN) [197] proposed by Barrett *et al.* suggests that the agent should seek to continuously improve the solution by learning to *explore at test time*, which resulted in greater ability to generalize to unseen graph sizes and structures. This and other RL methods could potentially be used to learn from actual user interactions within the Tjek app and continuously adjust the initially predicted catalog structure. Even more aligned with the original area of application is the Learn2Assemble RL model proposed by Funk *et al.* (2022) which learns to assemble entire 3D structures through a combination of multi-head attention graph representation, Q-learning and Monte Carlo Tree Search. Nonetheless, the scale of the problem that Learn2Assemble was tested on does not match the scale of the PROCAT dataset (up to 22 blocks compared to hundreds of product offers in a single catalog).

A different direction of future efforts could come in the form of improving on existing work. Starting with the provided library for generating synthetic catalogs, it could be further expanded to include other types of compositional and structural rules. These could include sections consisting of three or more types of atomic product offers in more complicated ratios or multiple valid structural orders of sections per input set composition. It could also be beneficial to investigate other ways of ensuring that only n^{th} order interactions are required to be learned to predict catalogs that are valid according to a specific rule. Existing datasets in the relational reasoning space, such as Sort-of-CLEVR [188], adjusted Kinetics [191] and datasets of simulated physical mass-spring systems constructed via the MuJoCo physics engine [199] clearly distinguish between relational and non-relational patterns, but are less clear when it

comes to the degree or order of the relational rule that needs to be learned. For example the Sort-of-CLEVR visual question answering dataset presents pictures of up to six elements, but having to answer a question about which element is farthest from another might only require knowledge of five pairwise relations.

Furthermore, research efforts could be devoted to improving upon the two presented neural architectures. The Set Interdependence Transformer could benefit from a firm theoretical analysis of its ability to encode higher-order interactions and the Neural Ordered Clusters could be reformulated as a fully generative model in accordance with the nature of the PROCAT dataset, as discussed in Chapter 3.3.2. Set-to-sequence models sometimes use beam search to obtain multiple predictions per input set, which I have abstained from due to the work of Meister *et al.* (2020) which showed that exact maximum a posteriori decoding of NLP generators frequently leads to low-quality results. Nonetheless, in principle nothing prevents the application of a fully generative approach in the cluster ordering stage and such a model's ability to suggest multiple, perhaps equally probable catalog alternatives could be useful for A/B testing in the downstream application. This could be further combined with the learning signal from direct user interactions, as hinted at earlier in this Chapter.

Finally, the presented work could be extended to other, related challenges or combined with novel ML techniques to provide a better experience to the end user. The generated catalogs could be tailored to the specific user's preferences in a private yet personalized way through federated learning [201]. A real-world challenge with business applications that can be structured in a way reminiscent of the Catalog Problem is the selection of news and other types of articles for digital media outlets. Similarly, certain recent developments in the area of using ML to construct curriculums for students suggest that this can be approached through grouping topics that need to be learned together, but in any order, possibly tailored to the individual student, and then ordering these groups to enable hierarchical learning [202, 203]. Furthermore, deep learning methods are being applied to medical triage datasets in

a semi-supervised manner [204]. I believe the applicability of presented methods to these and other emerging areas speaks to the usefulness of NN-based, supervised ordered clustering with regards to new and exciting challenges.

References

[11] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh, "Set transformer: A framework for attention-based permutation-invariant neural networks," in *International conference on machine learning*, PMLR, 2019, pp. 3744–3753.

[17] Y. Yin *et al.*, "Enhancing pointer network for sentence ordering with pairwise ordering predictions," in *AAAI*, vol. 34, 2020, pp. 9482–9489.

[19] X. Yu, N. T. Vu, and J. Kuhn, "Learning the dyck language with attention-based seq2seq models," in *ACL Workshop BlackboxNLP*, 2019, pp. 138–146.

[70] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," *arXiv:1803.03324*, 2018.

[71] A. Grover, A. Zweig, and S. Ermon, "Graphite: Iterative generative modeling of graphs," in *International conference on machine learning*, PMLR, 2019, pp. 2434–2444.

[72] R. Liao *et al.*, "Efficient graph generation with graph recurrent attention networks," *Advances in neural information processing systems*, vol. 32, 2019.

[73] S. A. Shah and V. Koltun, "Auto-decoding graphs," *arXiv:2006.02879*, 2020.

[74] W. Jin, R. Barzilay, and T. Jaakkola, "Hierarchical generation of molecular graphs using structural motifs," in *International conference on machine learning*, PMLR, 2020, pp. 4839–4848.

[96] X. Chen and Y. Tian, "Learning to perform local rewriting for combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[97] Q. Ma, S. Ge, D. He, D. Thaker, and I. Drori, "Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning," *arXiv e-prints*, arXiv–1911, 2019.

[98] K. Li, T. Zhang, and R. Wang, "Deep reinforcement learning for multiobjective optimization," *IEEE transactions on cybernetics*, vol. 51, no. 6, pp. 3103–3114, 2020.

[99] Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim, "Learning improvement heuristics for solving routing problems..," *IEEE transactions on neural networks and learning systems*, 2021.

[100] W. Kool, H. van Hoof, J. Gromicho, and M. Welling, "Deep policy dynamic programming for vehicle routing problems," in *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, Springer, 2022, pp. 190–213.

[102] M. Veres and M. Moussa, "Deep learning for intelligent transportation systems: A survey of emerging trends," *IEEE Transactions on Intelligent transportation systems*, vol. 21, no. 8, pp. 3152–3168, 2019.

[103] X. Zang, H. Yao, G. Zheng, N. Xu, K. Xu, and Z. Li, "Metalight: Value-based meta-reinforcement learning for traffic signal control," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 1153–1160.

[104] L. Brunke *et al.*, "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, pp. 411–444, 2022.

[118] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[135] N. Mostafazadeh *et al.*, "A corpus and cloze evaluation for deeper understanding of commonsense stories," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 839–849.

[188] A. Santoro *et al.*, "A simple neural network module for relational reasoning," *Advances in neural information processing systems*, vol. 30, 2017.

[191] W. Kay *et al.*, "The kinetics human action video dataset," *arXiv:1705.06950*, 2017.

[193] B. Gavish and S. C. Graves, "The travelling salesman problem and related problems," 1978.

[194] G. Jäger and J. Rogers, "Formal language theory: Refining the chomsky hierarchy," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 367, no. 1598, pp. 1956–1970, 2012.

[195] K. Sadeghi and S. Lauritzen, "Markov properties for mixed graphs," *Bernoulli*, vol. 20, no. 2, pp. 676–696, 2014.

[196] C. Bilodeau, W. Jin, T. Jaakkola, R. Barzilay, and K. F. Jensen, "Generative models for molecular discovery: Recent advances and challenges," *Wiley Interdisciplinary Reviews: Computational Molecular Science*, e1608, 2022.

[197] T. Barrett, W. Clements, J. Foerster, and A. Lvovsky, "Exploratory combinatorial optimization with reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 3243–3250.

[198] N. Funk, G. Chalvatzaki, B. Belousov, and J. Peters, "Learn2assemble with structured representations and search for robotic architectural construction," in *Conference on Robot Learning*, PMLR, 2022, pp. 1401–1411.

[199] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ international conference on intelligent robots and systems*, IEEE, 2012, pp. 5026–5033.

[200] C. Meister, R. Cotterell, and T. Vieira, "If beam search is the answer, what was the question?" In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 2173–2185.

[201] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, "Federated learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 13, no. 3, pp. 1–207, 2019.

[202] R. Ball, L. Duhadway, K. Feuz, J. Jensen, B. Rague, and D. Weidman, "Applying machine learning to improve curriculum design," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2019, pp. 787–793.

[203] M. E. Webb *et al.*, "Machine learning for human learners: Opportunities, issues, tensions and threats," *Educational Technology Research and Development*, vol. 69, no. 4, pp. 2109–2130, 2021.

[204] A. Buchard *et al.*, "Learning medical triage from clinicians using deep q-learning," *arXiv e-prints*, arXiv–2003, 2020.

Bibliography

[1] M. Jurewicz and L. Derczynski, "PROCAT: Product catalogue dataset for implicit clustering, permutation learning and structure prediction," in *Thirty-fifth Conference on Neural Information Processing Systems. Datasets and Benchmarks Track*, 2021.

[20] A. Pakman, Y. Wang, C. Mitelut, J. Lee, and L. Paninski, "Neural clustering processes," in *International Conference on Machine Learning*, PMLR, 2020, pp. 7455–7465.

[21] J. Lee, Y. Lee, and Y. W. Teh, "Deep amortized clustering," *arXiv:1909.13433*, 2019.

[22] Y. Wang *et al.*, "Amortized probabilistic detection of communities in graphs," *arXiv:2010.15727*, 2020.

[23] S. Gershman and N. Goodman, "Amortized inference in probabilistic reasoning," in *Proceedings of the annual meeting of the cognitive science society*, vol. 36, 2014.

[24] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[205] K. Cho *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *EMNLP*, 2014.

[2] X. Jin and J. Han, "Partitional clustering," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 766–766, ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_631. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_631.

[3] Miljøministeriet, *Danskernes papirforbug i kraftigt fald*, Miljøstyrelsen, Ed., Dec. 2018. [Online]. Available: https://mst.dk/service/nyheder/nyhedsarkiv/2018/sep/danskernes-papirforbrug-i-kraftigt-fald/.

[4] K. Dixon, *How much do you save in carbon emissions by being a good energy customer?* GoodEnergy, Ed., Sep. 2020. [Online]. Available: https://www.goodenergy.co.uk/good-stats-on-carbon-saving/.

[5] UN, "Kyoto protocol," *United Nations Framework Convention on Climate Change. Available online: http://unfccc. int/kyoto_protocol/items/2830.php*, 1997.

[6] S. Wirtz-Brückner and E.-M. Jakobs, "Product catalogs in the face of digitalization," in *2018 IEEE International Professional Communication Conference (ProComm)*, IEEE, 2018, pp. 98–106.

[7] H. M. Zahera and M. Sherif, "Probert: Product data classification with fine-tuning bert model.," in *MWPD@ ISWC*, 2020.

[8] S. Bird, "NLTK: The natural language toolkit," in *NLTK: The natural language toolkit*, Jan. 2006. DOI: 10.3115/1225403.1225421.

[9] Y. C. Xu, S. Cai, and H.-W. Kim, "Cue consistency and page value perception: Implications for web-based catalog design," *Information & Management*, vol. 50, no. 1, pp. 33–42, 2013.

[10] M. Zaheer, S. Kottur, S. Ravanbhakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 3394–3404.

[11] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh, "Set transformer: A framework for attention-based permutation-invariant neural networks," in *International conference on machine learning*, PMLR, 2019, pp. 3744–3753.

[12] Y. Zhang, J. Hare, and A. Prügel-Bennett, "Fspool: Learning set representations with featurewise sort pooling," in *International Conference on Learning Representations*, 2019.

[13] B. Yang, S. Wang, A. Markham, and N. Trigoni, "Robust attentional aggregation of deep feature sets for multi-view 3d reconstruction," *International Journal of Computer Vision*, vol. 128, no. 1, pp. 53–73, 2020.

[14] K. Skianis, G. Nikolentzos, S. Limnios, and M. Vazirgiannis, "Rep the set: Neural networks for learning set representations," in *International conference on artificial intelligence and statistics*, PMLR, 2020, pp. 1410–1420.

[15] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *Advances in neural information processing systems*, vol. 28, 2015.

[16] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," *arXiv:1511.06391*, 2016.

[17] Y. Yin *et al.*, "Enhancing pointer network for sentence ordering with pairwise ordering predictions," in *AAAI*, vol. 34, 2020, pp. 9482–9489.

[18] K. H. Hansen and J. Krarup, "Improvements of the held—karp algorithm for the symmetric traveling-salesman problem," *Mathematical Programming*, vol. 7, no. 1, pp. 87–96, 1974.

[19] X. Yu, N. T. Vu, and J. Kuhn, "Learning the dyck language with attention-based seq2seq models," in *ACL Workshop BlackboxNLP*, 2019, pp. 138–146.

[25] H. Wenzel, D. Smit, and S. Sardesai, "A literature review on machine learning in supply chain management," in *Artificial Intelligence and Digital Transformation in Supply Chain Management: Innovative Approaches for Supply Chains. Proceedings of the Hamburg International Conference of Logistics (HICL), Vol. 27*, Berlin: epubli GmbH, 2019, pp. 413–441.

[26] J. Miles, J. Turner, R. Jacques, J. Williams, and S. Mason, "Using machine-learning risk prediction models to triage the acuity of undifferentiated patients entering the emergency care system: A systematic review," *Diagnostic and prognostic research*, vol. 4, no. 1, pp. 1–12, 2020.

[27] P. Rabiller, P. Boles, and B. Boashash, "Ordered clustering: A way to simplify analysis of multichannel signals," in *10th International Conference on Information Science, Signal Processing and their Applications (ISSPA 2010)*, IEEE, 2010, pp. 237–242.

[28] M. Le, D. Nauck, B. Gabrys, and T. Martin, "Sequential clustering for event sequences and its impact on next process step prediction," in *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Springer, 2014, pp. 168–178.

[29] M. Jurewicz and L. Derczynski, "Set interdependence transformer: Set-to-sequence neural networks for permutation learning and structure prediction," in *31st International Joint Conference on Articial Intelligence, IJCAI-ECAI 2022*, 2022.

[30] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure," in *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, 2007, pp. 410–420.

[31] Q. Huang, H. He, A. Singh, Y. Zhang, S.-N. Lim, and A. Benson, "Better set representations for relational reasoning," *Proc. NeurIPS*, vol. 34, 2020.

[32] R. R. Larson, J. McDonough, P. O'Leary, L. Kuntz, and R. Moon, "Cheshire ii: Designing a next-generation online catalog," *Journal of the American Society for information Science*, vol. 47, no. 7, pp. 555–567, 1996.

[33] S. Q. Yang and K. Wagner, "Evaluating and comparing discovery tools: How close are we towards next generation catalog?" *Library hi tech*, 2010.

[34] D. Allison, "Information portals: The next generation catalog," *Journal of web librarianship*, vol. 4, no. 4, pp. 375–389, 2010.

[35] J. Emanuel, "Usability of the vufind next-generation online catalog," *Information technology and libraries*, vol. 30, no. 1, pp. 44–52, 2011.

[36] A. M. Keller and M. R. Genesereth, "Multi-vendor catalogs: Smart catalogs and virtual catalogs," in *EDI FORUM-OAK PARK-*, THE EDI GROUP, LTD., vol. 9, 1996, pp. 87–93.

[37] J. B. Baty and R. M. Lee, "Intershop: Enhancing the vendor/customer dialectic in electronic shopping," *Journal of Management Information Systems*, vol. 11, no. 4, pp. 9–31, 1995.

[38] J. W. Palmer, "Retailing on the www: The use of electronic product catalogs," *Electronic Markets*, vol. 7, no. 3, pp. 6–9, 1997.

[39] D.-M. Lincke and B. Schmid, "Mediating electronic product catalogs," *Communications of the ACM*, vol. 41, no. 7, pp. 86–88, 1998.

[40] P. Schubert, "The participatory electronic product catalog: Supporting customer collaboration in e-commerce applications," *Electronic Markets*, vol. 10, no. 4, pp. 229–236, 2000.

[41] S. Handschuh, B. F. Schmid, and K. Stanoevska-Slabeva, "The concept of a mediating electronic product catalog," *Electronic Markets*, vol. 7, no. 3, pp. 32–35, 1997.

[42] E. Callahan and J. Koenemann, "A comparative usability evaluation of user interfaces for online product catalog," in *Proceedings of the 2nd ACM Conference on Electronic Commerce*, 2000, pp. 197–206.

[43] A. Kobsa, "An empirical comparison of three commercial information visualization systems," in *IEEE Symposium on Information Visualization, 2001. INFOVIS 2001.*, IEEE, 2001, pp. 123–130.

[44] P. Markellou, M. Rigou, and S. Sirmakessis, "Product catalog shopping cart effective design," in *Web systems design and online consumer behavior*, IGI Global, 2005, pp. 232–251.

[45] M. S. B. Mimoun, M. Garnier, R. Ladwein, and C. Benavent, "Determinants of e-consumer productivity in product retrieval on a commercial website: An experimental approach," *Information & management*, vol. 51, no. 4, pp. 375–390, 2014.

[46] H. K. Farsani and M. Nematbakhsh, "A semantic recommendation procedure for electronic product catalog," *International Journal of Applied Mathematics and Computer Sciences*, vol. 3, no. 2, pp. 86–91, 2006.

[47] D. Fisher, L. Xu, and N. Zard, "Ordering effects in clustering," in *Machine Learning Proceedings 1992*, Elsevier, 1992, pp. 163–168.

[48] J. Roure and L. Talavera, "Robust incremental clustering with bad instance orderings: A new strategy," in *Ibero-American Conference on Artificial Intelligence*, Springer, 1998, pp. 136–147.

[49] L. E. B. da Silva and D. C. Wunsch, "A study on exploiting vat to mitigate ordering effects in fuzzy art," in *2018 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2018, pp. 1–8.

[50] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[51] K. C. Chu, "Applications of artificial intelligence to chemistry. use of pattern recognition and cluster analysis to determine the pharmacological activity of some organic compounds," *Analytical chemistry*, vol. 46, no. 9, pp. 1181–1187, 1974.

[52] M. F. Janowitz, "An order theoretic model for cluster analysis," *SIAM Journal on Applied Mathematics*, vol. 34, no. 1, pp. 55–72, 1978.

[53] D. H. Fisher, "Knowledge acquisition via incremental conceptual clustering," *Machine learning*, vol. 2, no. 2, pp. 139–172, 1987.

[54] S. M. Van Dongen, "Graph clustering by flow simulation," Ph.D. dissertation, Utrecht University, 2000.

[55] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," *ACM Sigmod record*, vol. 28, no. 2, pp. 49–60, 1999.

[56] U. Shaham, K. Stanton, H. Li, B. Nadler, R. Basri, and Y. Kluger, "Spectralnet: Spectral clustering using deep neural networks," *arXiv:1801.01587*, 2018.

[57] K. Turowski, J. K. Sreedharan, and W. Szpankowski, "Temporal ordered clustering in dynamic networks," in *2020 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2020, pp. 1349–1354.

[58] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *science*, vol. 315, no. 5814, pp. 972–976, 2007.

[59] J. Vlasblom and S. J. Wodak, "Markov clustering versus affinity propagation for the partitioning of protein interaction graphs," *BMC bioinformatics*, vol. 10, no. 1, pp. 1–14, 2009.

[60] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[61] J. Fernandes, M. Simsek, B. Kantarci, and S. Khan, "Tabledet: An end-to-end deep learning approach for table detection and table image classification in data sheet images," *Neurocomputing*, vol. 468, pp. 317–334, 2022.

[62] S. Abadal, A. Jain, R. Guirado, J. López-Alonso, and E. Alarcón, "Computing graph neural networks: A survey from algorithms to accelerators," *ACM Computing Surveys (CSUR)*, vol. 54, no. 9, pp. 1–38, 2021.

[63] W. Jiang and J. Luo, "Graph neural network for traffic forecasting: A survey," *Expert Systems with Applications*, p. 117 921, 2022.

[64] R. Winter, F. Noé, and D.-A. Clevert, "Permutation-invariant variational autoencoder for graph-level representation learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 9559–9573, 2021.

[65] K. Schütt, P.-J. Kindermans, H. E. Sauceda Felix, S. Chmiela, A. Tkatchenko, and K.-R. Müller, "Schnet: A continuous-filter convolutional neural network for modeling quantum interactions," *Advances in neural information processing systems*, vol. 30, 2017.

[66] S. Cavallari, V. W. Zheng, H. Cai, K. C.-C. Chang, and E. Cambria, "Learning community embedding with community detection and node embedding on graphs," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 377–386.

[67] A. Tsitsulin, J. Palowitch, B. Perozzi, and E. Müller, "Graph clustering with graph neural networks," *arXiv:2006.16904*, 2020.

[68] D. Jin, B. Li, P. Jiao, D. He, and H. Shan, "Community detection via joint graph convolutional network embedding in attribute network," in *International Conference on Artificial Neural Networks*, Springer, 2019, pp. 594–606.

[69] F.-Y. Sun, M. Qu, J. Hoffmann, C.-W. Huang, and J. Tang, "Vgraph: A generative model for joint community detection and node representation learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[70] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," *arXiv:1803.03324*, 2018.

[71] A. Grover, A. Zweig, and S. Ermon, "Graphite: Iterative generative modeling of graphs," in *International conference on machine learning*, PMLR, 2019, pp. 2434–2444.

[72] R. Liao *et al.*, "Efficient graph generation with graph recurrent attention networks," *Advances in neural information processing systems*, vol. 32, 2019.

[73] S. A. Shah and V. Koltun, "Auto-decoding graphs," *arXiv:2006.02879*, 2020.

[74] W. Jin, R. Barzilay, and T. Jaakkola, "Hierarchical generation of molecular graphs using structural motifs," in *International conference on machine learning*, PMLR, 2020, pp. 4839–4848.

[75] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[76] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[77] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv:1611.09940*, 2016.

[78] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[79] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[80] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1179–1191, 2020.

[81] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[82] A. Karpathy, "Deep reinforcement learning: Pong from pixels," *http://karpathy.github. io/2016/05/31/rl*, 2016.

[83] J. Zhang, J. Kim, B. O'Donoghue, and S. Boyd, "Sample efficient reinforcement learning with reinforce," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 10 887–10 895.

[84] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans, "Bridging the gap between value and policy based reinforcement learning," *Advances in neural information processing systems*, vol. 30, 2017.

[85] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[86] A. X. Lee, A. Nagabandi, P. Abbeel, and S. Levine, "Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model," *Advances in Neural Information Processing Systems*, vol. 33, pp. 741–752, 2020.

[87] L. Zheng, T. Fiez, Z. Alumbaugh, B. Chasnov, and L. J. Ratliff, "Stackelberg actor-critic: Game-theoretic reinforcement learning algorithms," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, 2022, pp. 9217–9224.

[88] O. Vinyals, I. Babuschkin, W. M. Czarnecki, and Mathieu, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019, ISSN: 14764687. DOI: 10.1038/s41586-019-1724-z. [Online]. Available: http://dx.doi.org/10.1038/s41586-019-1724-z.

[89] J. Edmonds, "First mathematical theory of efficient combinatorial algorithms: Jack edmonds share," 1965.

[90] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer Science & Business Media, 2012.

[91] M. M. Flood, "The traveling-salesman problem," *Operations research*, vol. 4, no. 1, pp. 61–75, 1956.

[92] M. R. Garey and D. S. Johnson, "A guide to the theory of np-completeness, a series of books in the mathematical sciences," *Computers and Intractability, WH Freeman and Co, San Francisco, CA*, 1979.

[93] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" In *International Conference on Learning Representations*, 2018.

[94] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[95] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[96] X. Chen and Y. Tian, "Learning to perform local rewriting for combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[97] Q. Ma, S. Ge, D. He, D. Thaker, and I. Drori, "Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning," *arXiv e-prints*, arXiv–1911, 2019.

[98] K. Li, T. Zhang, and R. Wang, "Deep reinforcement learning for multiobjective optimization," *IEEE transactions on cybernetics*, vol. 51, no. 6, pp. 3103–3114, 2020.

[99] Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim, "Learning improvement heuristics for solving routing problems..," *IEEE transactions on neural networks and learning systems*, 2021.

[100] W. Kool, H. van Hoof, J. Gromicho, and M. Welling, "Deep policy dynamic programming for vehicle routing problems," in *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, Springer, 2022, pp. 190–213.

[101] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management science*, vol. 6, no. 1, pp. 80–91, 1959.

[102] M. Veres and M. Moussa, "Deep learning for intelligent transportation systems: A survey of emerging trends," *IEEE Transactions on Intelligent transportation systems*, vol. 21, no. 8, pp. 3152–3168, 2019.

[103] X. Zang, H. Yao, G. Zheng, N. Xu, K. Xu, and Z. Li, "Metalight: Value-based meta-reinforcement learning for traffic signal control," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 1153–1160.

[104] L. Brunke *et al.*, "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, pp. 411–444, 2022.

[105] L.-C. Yang and A. Lerch, "On the evaluation of generative models in music," *Neural Computing and Applications*, vol. 32, no. 9, pp. 4773–4784, 2020.

[106] P. R. Halmos, *Naive set theory*. Courier Dover Publications, 2017.

[107] Z.-Q. Zhao, S.-T. Xu, D. Liu, W.-D. Tian, and Z.-D. Jiang, "A review of image set classification," *Neurocomputing*, vol. 335, pp. 251–260, 2019.

[108] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[109] H. Oliveira, C. Silva, G. L. Machado, K. Nogueira, and J. A. dos Santos, "Fully convolutional open set segmentation," *Machine Learning*, pp. 1–52, 2021.

[110] S. Thrun and L. Pratt, *Learning to learn*. Springer Science & Business Media, 2012.

[111] J. Schmidhuber, "Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-... hook," Ph.D. dissertation, Technische Universität München, 1987.

[112] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*, 2001, pp. 285–295.

[113] J. Wang *et al.*, "Learning fine-grained image similarity with deep ranking," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1386–1393.

[114] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian, "Scalable person re-identification : A benchmark university of texas at san antonio," in *ICCV 2015*, 2015, pp. 1116–1124, ISBN: 978-1-4673-8391-2. DOI: 10.1109/ICCV.2015.133.

[115] G. Bebis and M. Georgiopoulos, "Feed-forward neural networks," *IEEE Potentials*, vol. 13, no. 4, pp. 27–31, 1994.

[116] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

[117] E. Wagstaff, F. Fuchs, M. Engelcke, I. Posner, and M. A. Osborne, "On the limitations of representing functions on sets," in *International Conference on Machine Learning*, PMLR, 2019, pp. 6487–6494.

[118] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[119] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv:1607.06450*, 2016.

[120] R. Girgis *et al.*, "Latent variable sequential set transformers for joint multi-agent motion prediction," in *International Conference on Learning Representations*, 2021.

[121] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, "Point transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 16 259–16 268.

[122] F. Fuchs, D. Worrall, V. Fischer, and M. Welling, "Se (3)-transformers: 3d roto-translation equivariant attention networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1970–1981, 2020.

[123] N. Janakarajan, J. Born, and M. Manica, "A fully differentiable set autoencoder," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 3061–3071.

[124] S. Zare and H. Van Nguyen, "Picaso: Permutation-invariant cascaded attentional set operator," *arXiv:2107.08305*, 2021.

[125] M. J. Hutchinson, C. Le Lan, S. Zaidi, E. Dupont, Y. W. Teh, and H. Kim, "Lietransformer: Equivariant self-attention for lie groups," in *International Conference on Machine Learning*, PMLR, 2021, pp. 4533–4543.

[126] N. Naderializadeh, S. Kolouri, J. F. Comer, R. W. Andrews, and H. Hoffmann, "Set representation learning with generalized sliced-wasserstein embeddings," *arXiv:2103.03892*, 2021.

[127] J. J. Hopfield and D. W. Tank, ""neural" computation of decisions in optimization problems," *Biological cybernetics*, vol. 52, no. 3, pp. 141–152, 1985.

[128] D. Krotov and J. J. Hopfield, "Dense associative memory for pattern recognition," *Advances in neural information processing systems*, vol. 29, 2016.

[129] M. Demircigil, J. Heusel, M. Löwe, S. Upgang, and F. Vermet, "On a model of associative memory with huge storage capacity," *Journal of Statistical Physics*, vol. 168, no. 2, pp. 288–299, 2017.

[130] D. Krotov and J. Hopfield, "Dense associative memory is robust to adversarial inputs," *Neural computation*, vol. 30, no. 12, pp. 3151–3167, 2018.

[131] M. Widrich *et al.*, "Modern hopfield networks and attention for immune repertoire classification," *Advances in Neural Information Processing Systems*, vol. 33, pp. 18 832–18 845, 2020.

[132] V. Srinivasan, "Linear programming computational procedures for ordinal regression," *Journal of the ACM (JACM)*, vol. 23, no. 3, pp. 475–487, 1976.

[133] P. McCullagh, "Regression models for ordinal data," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 42, no. 2, pp. 109–127, 1980.

[134] J. Cheng, Z. Wang, and G. Pollastri, "A neural network approach to ordinal regression," in *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, IEEE, 2008, pp. 1279–1284.

[135] N. Mostafazadeh *et al.*, "A corpus and cloze evaluation for deeper understanding of commonsense stories," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 839–849.

[136] Y. Xiao, B. Liu, and Z. Hao, "Multiple-instance ordinal regression," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 9, pp. 4398–4413, 2018.

[137] H. Zhu *et al.*, "Convolutional ordinal regression forest for image ordinal estimation," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[138] P. A. Gutiérrez, M. Perez-Ortiz, J. Sanchez-Monedero, F. Fernandez-Navarro, and C. Hervas-Martinez, "Ordinal regression methods: Survey and experimental study," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 1, pp. 127–146, 2015.

[139] J. L. Stuart and J. R. Weaver, "Matrices that commute with a permutation matrix," *Linear Algebra and Its Applications*, vol. 150, pp. 255–265, 1991.

[140] A. Grover, E. Wang, A. Zweig, and S. Ermon, "Stochastic optimization of sorting networks via continuous relaxations," *Preprint arXiv:1903.08850*, vol. ArXiv, 2019.

[141] G. Mena, D. Belanger, S. Linderman, and J. Snoek, "Learning latent permutations with gumbel-sinkhorn networks," in *International Conference on Learning Representations 2018*, Feb. 2018.

[142] S. Linderman, G. Mena, H. Cooper, L. Paninski, and J. Cunningham, "Reparameterizing the birkhoff polytope for variational permutation inference," in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2018, pp. 1618–1627.

[143] P. Emami and S. Ranka, "Learning permutations with sinkhorn policy gradient," *ArXiv*, vol. abs/1805.07010, 2018.

[144] N. Nishida and H. Nakayama, "Word ordering as unsupervised learning towards syntactically plausible word representations," in *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2017, pp. 70–79.

[145] Y. Zhang, J. Hare, and A. Prügel-Bennett, "Learning representations of sets through optimized permutations," *arXiv e-prints*, arXiv–1812, 2018.

[146] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: From pairwise approach to listwise approach," in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 129–136.

[147] B. Mitra *et al.*, "An introduction to neural information retrieval," *Foundations and Trends® in Information Retrieval*, vol. 13, no. 1, pp. 1–126, 2018.

[148] K. Pfannschmidt, P. Gupta, and E. Hüllermeier, "Deep architectures for learning context-dependent ranking functions," *ArXiv*, vol. abs/1803.05796, 2018.

[149] Q. Ai, K. Bi, J. Guo, and W. B. Croft, "Learning a deep listwise context model for ranking refinement," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2018, pp. 135–144.

[150] C. Wang, H. Zhu, C. Zhu, C. Qin, and H. Xiong, "Setrank: A setwise bayesian approach for collaborative ranking from implicit feedback," in *Proceedings of the aaai conference on artificial intelligence*, vol. 34, 2020, pp. 6127–6136.

[151] L. Pang, J. Xu, Q. Ai, Y. Lan, X. Cheng, and J. Wen, "Setrank: Learning a permutation-invariant ranking model for information retrieval," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020, pp. 499–508.

[152] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv:1409.0473*, 2014.

[153] O. Vinyals, Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, "Grammar as a foreign language," *Advances in neural information processing systems*, vol. 28, 2015.

[154] J. Donahue *et al.*, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 2625–2634.

[155] W. Zaremba and I. Sutskever, "Learning to execute," *arXiv:1410.4615*, 2014.

[156] D.-T. Lee and B. J. Schachter, "Two algorithms for constructing a delaunay triangulation," *International Journal of Computer & Information Sciences*, vol. 9, no. 3, pp. 219–242, 1980.

[157] B. Cui, Y. Li, M. Chen, and Z. Zhang, "Deep attentive sentence ordering network," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 4340–4349.

[158] L. Logeswaran, H. Lee, and D. Radev, "Sentence ordering and coherence modeling using recurrent neural networks," in *Thirty-second aaai conference on artificial intelligence*, 2018.

[159] Z. Sun, J. Tang, P. Du, Z.-H. Deng, and J.-Y. Nie, "Divgraphpointer: A graph pointer network for extracting diverse keyphrases," in *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2019, pp. 755–764.

[160] Y. Yin *et al.*, "An external knowledge enhanced graph-based neural network for sentence ordering," *Journal of Artificial Intelligence Research*, vol. 70, pp. 545–566, 2021.

[161] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *International conference on machine learning*, PMLR, 2016, pp. 478–487.

[162] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, "Towards k-means-friendly spaces: Simultaneous deep learning and clustering," in *international conference on machine learning*, PMLR, 2017, pp. 3861–3870.

[163] F. Li, H. Qiao, and B. Zhang, "Discriminatively boosted image clustering with fully convolutional auto-encoders," *Pattern Recognition*, vol. 83, pp. 161–173, 2018.

[164] S.-H. Lee and C.-S. Kim, "Deep repulsive clustering of ordered data based on order-identity decomposition," in *International Conference on Learning Representations*, 2020.

[165] Y. Li, P. Hu, Z. Liu, D. Peng, J. T. Zhou, and X. Peng, "Contrastive clustering," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 8547–8555.

[166] E. Aljalbout, V. Golkov, Y. Siddiqui, M. Strobel, and D. Cremers, "Clustering with deep learning: Taxonomy and new methods," *arXiv:1801.07648*, 2018.

[167] Y.-C. Hsu, Z. Lv, J. Schlosser, P. Odom, and Z. Kira, "Multi-class classification without multi-class labels," in *International Conference on Learning Representations*, 2018.

[168] Y.-C. Hsu, Z. Lv, and Z. Kira, "Learning to cluster in order to transfer across domains and tasks," *arXiv:1711.10125*, 2017.

[169] S. Coward, E. Visse-Martindale, and C. Ramesh, "Attention-based clustering: Learning a kernel from context," *arXiv:2010.01040*, 2020.

[170] G. Huang, H. Larochelle, and S. Lacoste-Julien, "Centroid networks for few-shot clustering and unsupervised few-shot classification," *arXiv:1902.08605*, vol. 3, no. 7, 2019.

[171] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[172] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," *Advances in neural information processing systems*, vol. 28, 2015.

[173] J. Leskovec and A. Krevl, *Snap datasets: Stanford large network dataset collection*, 2014.

[174] H. Liu, T. Zhou, J. Wang, L. Jing, and M. K. Ng, "Deep amortized relational model with group-wise hierarchical generative process," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2022.

[175] S. Fogel, H. Averbuch-Elor, D. Cohen-Or, and J. Goldberger, "Clustering-driven deep embedding with pairwise constraints," *IEEE computer graphics and applications*, vol. 39, no. 4, pp. 16–27, 2019.

[176] H. Liu, Y. Jia, J. Hou, and Q. Zhang, "Imbalance-aware pairwise constraint propagation," in *Proceedings of the 27th ACM international conference on multimedia*, 2019, pp. 1605–1613.

[177] B. Boecking and A. Dubrawski, "Pairwise feedback for data programming," *arXiv:1912.07685*, 2019.

[178] A. Shukla, G. S. Cheema, and S. Anand, "Semi-supervised clustering with neural networks," in *2020 IEEE Sixth International Conference on Multimedia Big Data (BigMM)*, IEEE, 2020, pp. 152–161.

[179] B. Boecking, V. Jeanselme, and A. Dubrawski, "Constrained clustering and multiple kernel learning without pairwise constraint relaxation," *arXiv:2203.12546*, 2022.

[180] H. Zhang, T. Zhan, S. Basu, and I. Davidson, "A framework for deep constrained clustering," *Data Mining and Knowledge Discovery*, vol. 35, no. 2, pp. 593–620, 2021.

[181] D. D. Lewis, Y. Yang, T. Russell-Rose, and F. Li, "Rcv1: A new benchmark collection for text categorization research," *Journal of machine learning research*, vol. 5, no. Apr, pp. 361–397, 2004.

[182] I. Davidson, K. L. Wagstaff, and S. Basu, "Measuring constraint-set utility for partitional clustering algorithms," in *European conference on principles of data mining and knowledge discovery*, Springer, 2006, pp. 115–126.

[183] R. Van Der Linden, R. Lopes, and R. Bidarra, "Procedural generation of dungeons," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 1, pp. 78–89, 2013.

[184] N. Parmar *et al.*, "Image transformer," in *International conference on machine learning*, PMLR, 2018, pp. 4055–4064.

[185] K. Han, A. Xiao, E. Wu, J. Guo, C. Xu, and Y. Wang, "Transformer in transformer," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15 908–15 919, 2021.

[186] N. Engel, V. Belagiannis, and K. Dietmayer, "Point transformer," *IEEE Access*, vol. 9, pp. 134 826–134 840, 2021.

[187] C. He, R. Li, S. Li, and L. Zhang, "Voxel set transformer: A set-to-set approach to 3d object detection from point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 8417–8427.

[188] A. Santoro *et al.*, "A simple neural network module for relational reasoning," *Advances in neural information processing systems*, vol. 30, 2017.

[189] E. Wagstaff, F. B. Fuchs, M. Engelcke, M. A. Osborne, and I. Posner, "Universal approximation of functions on sets," *Journal of Machine Learning Research*, vol. 23, no. 151, pp. 1–56, 2022.

[190] R. L. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro, "Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs," in *International Conference on Learning Representations*, 2018.

[191] W. Kay *et al.*, "The kinetics human action video dataset," *arXiv:1705.06950*, 2017.

[192] C.-Y. Ma, A. Kadav, I. Melvin, Z. Kira, G. AlRegib, and H. P. Graf, "Attend and interact: Higher-order object interactions for video understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6790–6800.

[206] A. Jaegle, F. Gimeno, A. Brock, O. Vinyals, A. Zisserman, and J. Carreira, "Perceiver: General perception with iterative attention," in *International conference on machine learning*, PMLR, 2021, pp. 4651–4664.

[207] G. Mialon, D. Chen, A. d'Aspremont, and J. Mairal, "A trainable optimal transport embedding for feature aggregation and its relationship to attention," in *ICLR 2021-The Ninth International Conference on Learning Representations*, 2021.

[208] M. Kim, "Differentiable expectation-maximization for set representation learning," in *International Conference on Learning Representations*, 2021.

[209] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms," *arXiv:1708.07747*, 2017.

[210] T. U. Haque, N. N. Saber, and F. M. Shah, "Sentiment analysis on large scale amazon product reviews," in *2018 IEEE international conference on innovative research and development (ICIRD)*, IEEE, 2018, pp. 1–6.

[211] Y. Huang, A. Conkey, and T. Hermans, "Planning for multi-object manipulation with graph neural network relational classifiers," *arXiv:2209.11943*, 2022.

[193] B. Gavish and S. C. Graves, "The travelling salesman problem and related problems," 1978.

[194] G. Jäger and J. Rogers, "Formal language theory: Refining the chomsky hierarchy," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 367, no. 1598, pp. 1956–1970, 2012.

[195] K. Sadeghi and S. Lauritzen, "Markov properties for mixed graphs," *Bernoulli*, vol. 20, no. 2, pp. 676–696, 2014.

[196] C. Bilodeau, W. Jin, T. Jaakkola, R. Barzilay, and K. F. Jensen, "Generative models for molecular discovery: Recent advances and challenges," *Wiley Interdisciplinary Reviews: Computational Molecular Science*, e1608, 2022.

[197] T. Barrett, W. Clements, J. Foerster, and A. Lvovsky, "Exploratory combinatorial optimization with reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 3243–3250.

[198] N. Funk, G. Chalvatzaki, B. Belousov, and J. Peters, "Learn2assemble with structured representations and search for robotic architectural construction," in *Conference on Robot Learning*, PMLR, 2022, pp. 1401–1411.

[199] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ international conference on intelligent robots and systems*, IEEE, 2012, pp. 5026–5033.

[200] C. Meister, R. Cotterell, and T. Vieira, "If beam search is the answer, what was the question?" In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 2173–2185.

[201] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, "Federated learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 13, no. 3, pp. 1–207, 2019.

[202] R. Ball, L. Duhadway, K. Feuz, J. Jensen, B. Rague, and D. Weidman, "Applying machine learning to improve curriculum design," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2019, pp. 787–793.

[203] M. E. Webb *et al.*, "Machine learning for human learners: Opportunities, issues, tensions and threats," *Educational Technology Research and Development*, vol. 69, no. 4, pp. 2109–2130, 2021.

[204] A. Buchard *et al.*, "Learning medical triage from clinicians using deep q-learning," *arXiv e-prints*, arXiv–2003, 2020.