Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Input-Output Representations for Supervised Clustering Methods

Jakub Monhart

Supervisor: Ing. Jan Drchal, Ph.D., Artificial Intelligence Center FEE
Field of study: Cybernetics and Robotics
May 2021

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Monhart Jakub** Personal ID number: **483438**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Input-Output Representations for Supervised Clustering Methods

Bachelor's thesis title in Czech:

Kódování vstupů a výstupů pro metody supervizovaného shlukování

Guidelines:

The task is to compare various approaches to encode input and output data for neural network-based supervised clustering methods.
1) Explore the state-of-the-art methods for supervised-clustering. Focus on algorithms based on the neural network paradigm.
2) Either implement or use existing reference implementations of selected methods.
3) Consider modifications of these methods to allow comparison of different input-output data representation approaches.
3) Compare the methods using both synthetic and real-world data (supplied by the supervisor). The experiments should mainly focus on 1) different possibilities to encode variable-sized inputs and outputs, 2) the ability to deal with intra-cluster data dependencies, and 3) input size scalability.

Bibliography / sources:

[1] Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., & Smola, A. J. (2017). Deep sets. In Advances in neural information processing systems (pp. 3391-3401).
[2] Ari Pakman, Yueqi Wang, Catalin Mitelut, Jinhyung Lee, Liam Paninski Proceedings of the 37th International Conference on Machine Learning, PMLR 119:7455-7465, 2020.
[3] Pakman, Ari, et al. "Attentive Clustering Processes." arXiv preprint arXiv:2010.15727 (2020).
[4] Lee, Juho, et al. "Set transformer: A framework for attention-based permutation-invariant neural networks." International Conference on Machine Learning. PMLR, 2019.
[5] Lee, Juho, Yoonho Lee, and Yee Whye Teh. "Deep amortized clustering." arXiv preprint arXiv:1909.13433 (2019).
[6] Hsu, Yen-Chang, and Zsolt Kira. "Neural network-based clustering using pairwise constraints." arXiv preprint arXiv:1511.06321 (2015).
[7] Coward, Samuel, Erik Visse-Martindale, and Chithrupa Ramesh. "Attention-Based Clustering: Learning a Kernel from Context." arXiv preprint arXiv:2010.01040 (2020)

Name and workplace of bachelor's thesis supervisor:

Ing. Jan Drchal, Ph.D., Department of Theoretical Computer Science, FIT

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **28.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until:
by the end of summer semester 2021/2022

Ing. Jan Drchal, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student's signature

Acknowledgements

I would like to thank my supervisor, Jan Drchal, for his useful guidance, insightful comments, and support during my work on this thesis.

I would also like to give special thanks to my family and close friends, who support me greatly throughout my studies.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 21, 2021

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 21. května 2021

Abstract

In this thesis, we explore solutions to the supervised clustering problem, focusing on neural network-based methods. We are motivated by problems such as semantic text summarization, topic extraction, and fast annotation of data.

The goal of a supervised clustering model is to partition a *set* of a variable number of *elements* into clusters. This is done by first training the model using labeled data.

In the first part, we explore machine learning approaches to processing set-structured data and the current state of the art methods for solving the supervised clustering problem. The theory necessary to define set-processing methods is reviewed, and two state of the art models for solving the supervised clustering problem are described in detail. We further propose two new methods, each using a different representation of the input and output.

Using experiments on one real-world and two synthetic datasets, we compare the two state of the art methods with the proposed methods. We explore the ability to deal with intra-cluster data dependencies and the scalability of the examined models to the size of a set of elements to be clustered.

Keywords: supervised clustering, machine learning, neural networks, set-structured data

Supervisor: Ing. Jan Drchal, Ph.D., Artificial Intelligence Center FEE

Abstrakt

V rámci této práce zkoumáme možná řešení pro problém supervizovaného shlukování se zaměřením na metody založené na neuronových sítích. Naší motivací jsou problémy jako je shrnutí textu podle sémantiky, extrakce tématu z textu a rychlá anotace dat.

Cílem modelu řešícího supervizované shlukování je zpracovat *množinu* obsahující proměnný počet *prvků* a rozdělit tyto prvky do shluků. Za tímto účelem je model nejdříve trénován pomocí dat u kterých známe správné rozdělení prvků do shluků.

V první části zkoumáme přístupy strojového učení ke zpracování dat strukturovaných jako proměnný počet prvků v množině a současné state of the art metody pro řešení supervizovaného shlukování. Uvádíme teorii nezbytnou k definování metod zpracovávajících množiny prvků a podrobně popisujeme dva state of the art modely pro řešení supervizovaného shlukování. Dále jsme navrhli dvě nové metody, z nichž každá používá jiné kódování vstupu a výstupu.

Pomocí experimentů na jednom reálném a dvou syntetických datasetech porovnáváme popsané state of the art metody s metodami které jsme navrhli. Zaměřujeme se na schopnost zpracovat data se závislostmi mezi prvky uvnitř shluků a na škálovatelnost zkoumaných modelů při rostoucí velikosti vstupu.

Klíčová slova: supervizované shlukování, strojové učení, neuronové sítě, data strukturovaná do množin

Překlad názvu: Kódování vstupů a výstupů pro metody supervizovaného shlukování

Contents

Figures

Tables

Chapter 1

Introduction

Clustering is traditionally an unsupervised learning problem used frequently for statistical and exploratory data analysis. Its task is to partition a set of elements (samples) into multiple clusters, where elements from the same cluster are supposed to be more similar than elements from different clusters. With some data, however, it might not always be clear what the similarity of elements means. Most unsupervised clustering methods utilize a distance of the elements' representations.

Instead of solving the clustering problem as an unsupervised one, we build on recent advancements in supervised learning. These advancements are mainly due to deep learning networks that are able to learn a useful representation of the input data. Using supervised clustering, we want to omit the need to specify similarity between elements manually. Instead, we use ground truth clustering information to learn a deep learning model to infer the clusters directly or indirectly (by learning the similarity metric).

There are two problems with the supervised approach to clustering. First, supervised deep learning methods primarily work with samples with fixed-sized representation, whereas a set of elements we want to cluster can be of variable size. Second, when using deep learning for classification or regression, the number of outputs is generally known before the training. This is not the case with clustering, where the number of clusters is arbitrary. As such, a method solving a supervised clustering problem needs to be able to predict the number of clusters in given data. This essentially means that the supervised clustering problem consists of working with a variable number of inputs and outputs.

Our work on supervised clustering started with the Newspaper dataset described in Section 5.3. In short, we are given textboxes from a newspaper page. Each textbox is described by a set of features (such as position or font size) and our task is to cluster the textboxes into articles. Example of such page is shown in Figure 1.1. To solve this problem, we proposed MIL model (see Section 4.1). At that time, state of the art models described in Chapter 3 were published, and we decided to compare them with our approach. We proposed another model (Permutation Equivariant Model — PEM) inspired by the state of the art model Deep Amortized Clustering (Section 3.1) to test if architecture using simpler building blocks would work.

Figure 1.1: Examples of pages from Newspaper dataset. Text-boxes bounded with rectangle of same colour are from same cluster (article).

The approach of MIL and ABC (Attention Based Clustering — one of the state of the art methods) can be to some extent described as learning a similarity metric between two elements using the context of all other elements in the input. On the other hand, DAC and PEM process the whole variable-sized input directly and produce a part of the clustering at once.

We are also interested in datasets with intra-cluster dependencies. In such clusters, samples are not independently distributed. To provide an example, consider the difference between samples drawn from a mixture of Gaussians distribution versus samples representing points on multiple intersecting circles. Discussed intra-cluster dependency can also be found in the positioning of articles in the Newspaper dataset (see Figure 1.1). The headline is typically center- or left-aligned on top relative to other textboxes, and textboxes containing the body of the article are mostly structured into columns.

1.1 Motivation

Apart from the Newspaper dataset, we provide some possible uses of supervised clustering.

Consider, for example, a large number of documents that a user wants to separate into several groups based on semantics. Defining a rule-based system to do such clustering might be tedious and unfeasible. Instead, the user could divide a small portion of the documents into desirable groups and use the

model to learn to cluster the rest of the texts. This approach removes the need to formulate the definition of clusters or to define a similarity between two texts manually.

Another possible use is cheaper and faster data labeling that could work by first visualizing the data using some low-dimensional embedding such as PCA or t-SNE. Using this visualization, an annotator could quickly pick elements that should be assigned the same class (assuming such elements would be embedded close to each other in the low-dimensional embedding) according to sought-out labeling. This crude labeling would then be used as ground truth information to train a supervised clustering model. Afterward, the rest of the dataset could be quickly divided by the trained model into groups that satisfy the annotator's demands (specified by the initial crude labeling of the part of the dataset).

1.2 Problem specification

We are given a dataset consisting of sets (instead of fixed-sized samples). Each set contains a variable number of elements, where each element is represented by a vector $\boldsymbol{x}_i \in \mathbb{R}^d$ with a fixed number of features d. A whole set can be denoted as $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n_X}\}$, where n_X is the variable number of elements in set. We trade the mathematical correctness of our notation for clarity by using the same letter X for representing the set as matrix $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n_X}]^T \in \mathbb{R}^{n_X \times d}$, with elements stored as rows. A ground truth clustering for each set X is given by vector $\boldsymbol{y} \in [1, \ldots, k_X]^{n_X}$, where k_X is a true number of clusters in X. Clusters in set X are denoted with numbers $1, \ldots, k_X$ and for element $\boldsymbol{x}_i$, the true cluster number is given by i-th element y_i of vector $\boldsymbol{y}$.

The goal of a supervised clustering model is to process elements of a set X and predict the true cluster number y_i (denoting to which cluster the i-th element belongs) for each element x_i ($i = 1, \ldots, n_X$) of the set. This is done by first optimizing the model using a dataset of sets for which we have the ground truth clustering informationx. Both the number of elements and the number of clusters are variable. That is, the number of elements and the number of clusters can differ from set to set. Supervised clustering models must further satisfy an important property: the cluster assignment they predict must be invariant to permuting the order of elements in the input set. This means that upon shuffling the elements $\boldsymbol{x}_i$ of X, the same elements must be assigned to the same clusters as before.

Chapter 2

Background

Before defining individual models that solve the supervised clustering problem specified in Section 1.2, we describe fundamental ideas and methods behind them. These ideas and methods are based on the neural network paradigm. Methods discussed in this chapter do not address the supervised clustering problem directly, but rather the task of processing set-structured data generally, since supervised clustering is a set-processing task. The task of processing sets includes a range of problems — for example, 3D shape recognition, detecting set outliers, or multiple instance learning.

We can divide methods in this chapter into two groups. The first group of methods maps elements of a set from one representation to another. Such method's output is a set with the same number of elements as in the input set. A simple one-layer feedforward neural network could do this by mapping each element individually, but that would omit information, a context, from the rest of the set. A better approach is to embed the context into the output representation of each element. Such representation can then be pooled (using, for example, max or mean function) into a single vector or processed by a feedforward neural network element-wise. Equivariant Layer (Section 2.3) and Self-Attention Block from Set Transformer (Section 2.5) discussed in this chapter belong into this category of methods. Because a set is, by definition, an unordered group of elements, methods from this group must be *permutation equivariant*, as discussed in [12]. A funtion $\boldsymbol{f}$ operating on sets is permutation equivariant, if for any permutation π it holds:

$$\boldsymbol{f}[\boldsymbol{x}_{\pi(1)}, \ldots \boldsymbol{x}_{\pi(n_X)}] = [f(\boldsymbol{x}_{\pi(1)}), \ldots, f(\boldsymbol{x}_{\pi(n_X)})], \tag{2.1}$$

that is, the order of elements in the output set permutes correspondingly upon permuting elements in the input set.

The second group of methods maps sets with a variable number of elements into fixed-sized representation. This is a case of Multiple Instance Learning (Section 2.2) and Pooling by Multi-head Attention block (Section 2.5). Having a fixed-sized representation of the variable-sized set then, for example, allows us to use well-researched machine learning methods for fixed-length representation. We require a method from this group to be *permutation invariant* (discussed again in [12]): the output does not change upon permutation of elements in the input set (shuffling rows of its matrix representation).

Since in this thesis we work exclusively with data structured as a variable number of elements in a set, where the elements have a fixed number of features, we continue to use the notation proposed in Section 1.2.

We denote a set of elements as:

$$X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n_X}\}, \qquad \boldsymbol{x}_i \in \mathbb{R}^d \tag{2.2}$$

and use the same letter X for representing the set as matrix $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n_X}]^T \in \mathbb{R}^{n_X \times d}$, where the elements are stored as rows.

2.1 Neural Networks

All methods used in this work fall within the field of machine learning, specifically neural networks. This means we can consider all described models (methods) to be functions with a large number of modifiable parameters. For each model, we also define a proper loss function. Since all mathematical operations in our models are differentiable, we can evaluate the loss function on training data (set X and corresponding ground-truth cluster numbers $\boldsymbol{y}$), differentiate the loss function with respect to all parameters (back-propagation), and optimize the parameters using a gradient-based method like Adam [5]. This process of optimizing the model's parameters using loss function and training data is generally called *training* in machine learning.

A frequently used component in the models we discuss is a standard feedforward neural network layer (with bias). We apply it row-wise to input matrix and denote it rFF. Considering a set $X \in \mathbb{R}^{n_X \times d}$, rFF is defined as:

$$\mathrm{rFF}(X) = XW + \mathbf{b}^T, \tag{2.3}$$

where $W \in \mathbb{R}^{d \times d_M}$ is a weight matrix and $\mathbf{b} \in \mathbb{R}^{d_M}$ is a bias. Shape of the output is given by parameter d_M: $\mathrm{rFF}(X) \in \mathbb{R}^{n_X \times d_M}$.

2.2 Multiple Instance Learning

Multiple Instance Learning (MIL) formalism assumes samples to be structured as multisets (considering the mathematical definition of set and multiset, a multiset, unlike a set, can contain an element repeatedly). The multisets are called bags in the MIL literature. We denote these bags (multisets) as *sets* to be concise with the rest of the thesis but do not require the elements in the set to be unique.

Each set contains a variable number of fixed-sized elements $\boldsymbol{x}$. The ground truth information y (class label or continuous variable) is only available for the set as a whole compared to having one label for each element. The goal is to find a function $f(X)$ that predicts label y of this set. Such function must be invariant to permuting the order of elements in the input set X (Permutation invariance).

This formalism is motivated by difficulty to describe real-world objects with fixed-sized numerical vectors. Using a set of vectors is usually more

natural. To give examples of such problems, we state two datasets used to test methods solving MIL problems in [9]: **Musk** dataset, where one set represents a molecule, and each element of that set represents one conformation of this molecule. The goal is to predict if the molecule is active, which happens if at least one of the conformations is active. And **Protein** dataset, where each set represents one protein and an element contains molecular and chemical properties of some part of that protein's sequence. The task is to predict if a protein represented by a set belongs to a particular family of proteins.

Authors of [9] proposed Neural Network formalism to solve MIL problems. This formalism is based on embedding the set X into a fixed-sized vector using mapping:

$$\phi(X) \in \mathbb{R}^m. \tag{2.4}$$

Having set X represented as a fixed-sized vector, we can apply any differentiable function to obtain the estimate of ground-truth label y.

Mapping ϕ consists of embedding each element $\boldsymbol{x} \in \mathbb{R}^d$ individually using neural network k_a"

$$k_a(\boldsymbol{x}, \theta_a) \in \mathbb{R}^m \tag{2.5}$$

with parameters θ_a. Embedded elements are then aggregated feature-wise by a pooling function g into a single vector of the same size m. If we use a pooling function for which we can compute derivative during back-propagation (e.g., mean or max), parameters θ_a can be optimized using a gradient-based method.

The whole architecture we will use can be denoted as:

$$y \approx k_b(g(\{k_a(\boldsymbol{x}_1, \theta_a), \ldots, k_a(\boldsymbol{x}_{n_X}, \theta_a)\}), \theta_b), \tag{2.6}$$

where k_b is neural network with parameters θ_b. We specify the exact form of neural networks k_a and k_b later.

A general function called *Invariant model* working with sets and labels on set-level is also described in [12]. The resulting architecture is of the same form as the one described above. Moreover, such architecture was also proven by authors of [12] to be a universal approximator for any set function.

2.3 Equivariant Layer

Article [12] focuses on functions working with set-structured data generally, motivated by problems such as estimation of population statistics or processing of data in cosmology. It introduces the Invariant model mentioned in the previous section, and *Equivariant layer* described below.

The Equivariant Layer again works with sets of a variable number of elements with a fixed size. For a set X on the input, it produces an output set with the same number of elements of size d_M (defined as a parameter of the layer). In contrast to rFF (2.3), elements of the set (rows of matrix X) are not

processed independently. During the processing of set X, information from the input representation of one element is used to produce the representation of each element of the output set. This enables the model to encode interactions between the elements and process the elements in context to each other.

We consider the set X to be represented as matrix: $X \in \mathbb{R}^{n_X \times d}$ in the following definitions. Equivariant layer is then defined as

$$\text{EquivLayer}(X) = \sigma(X\Lambda - \mathbf{1}\mathbf{1}^T X\Gamma), \tag{2.7}$$

where $\Lambda, \Gamma \in \mathbb{R}^{d \times d_M}$ are trainable parameters. The part $(\mathbf{1}\mathbf{1}^T X) \in \mathbb{R}^{n_X \times d}$ of the equation above results into a matrix with identical rows, where j-th element of the row equals to the sum of j-th column of X (sum of j-th feature over all elements in the set X). Output of EquivLayer is a set of elements again. Number of elements is preserved: $Y \in \mathbb{R}^{n_X \times d_M}$. Max and mean version of the *Equivariant Layer* can be defined as:

$$\text{EquivLayer}_{\text{max}}(X) = \sigma(X\Lambda - \mathbf{1}\text{MAX}(X)\Gamma), \tag{2.8}$$

$$\text{EquivLayer}_{\text{mean}}(X) = \sigma(X\Lambda - \mathbf{1}\text{MEAN}(X)\Gamma). \tag{2.9}$$

Max and mean in the equations above are computed for each column of X individually: $\mathbf{1}\text{max}(X)$, $\mathbf{1}\text{mean}(X) \in \mathbb{R}^{n_X \times d}$. We use EquivLayer_{MEAN} instead of the default Equivariant Layer (2.7), since difference between $\mathbf{1}\text{mean}(X)$ and $\mathbf{1}\mathbf{1}^T X$ is only in scaling by n_X.

A single equivariant layer is permutation equivariant. Stacking multiple equivariant layers results in a model that is also permutation equivariant.

2.4 Self-Attention

We deviate shortly into sequence processing to explain the motivation behind the self-attention mechanism but return to set-structured data to define it precisely.

Recurrent neural networks (RNN) are today a standard model architecture used to process sequences of data in tasks such as machine translation. Part of solution for such problems is mapping variable-length sequence of symbols represented as $(x_1, \ldots, x_n)$ to another representation of same length $(y_1, \ldots, y_n)$. In RNN, a sequence is processed sequentially and the hidden state of the RNN after processing x_i is needed before x_{i+1} can be processed. This inhibits the use of parallelization and, in practice, restricts the modeling of dependencies between symbols with large distances in the sequence. Self-attention allows to model these dependencies between symbols irrespective of their distance and enables parallel computation. These advantages motivated the use of the self-attention mechanism jointly with RNN in sequence modeling tasks.

The Transformer [10] is a model for sequence modeling that took this approach further and is based entirely on self-attention instead of recurrence. The reliance on self-attention is motivated by the problems of recurrence-based models described above. We mention the Transformer architecture

because it inspired the Set Transformer described in the next section and popularized the self-attention mechanism we describe below.

Attention is a function mapping set of n queries and a set of m key-value pairs to n outputs. Queries, keys, values and outputs are all vectors and we represent them as rows of matrices $Q \in \mathbb{R}^{n \times d_q}$, $K \in \mathbb{R}^{m \times d_q}$, $V \in \mathbb{R}^{m \times d_v}$. This is consistent with how we represent sets in the rest of this thesis. Single vector from output of attention function $\text{Attention}(Q, K, V) \in \mathbb{R}^{n \times d_v}$ is weighted sum of values V with weights computed by a compatibility function with the corresponding query and key as input:

$$\text{Attention}(Q, K, V) = \text{softmax}(\text{compat}(Q, K))V, \tag{2.10}$$

where $\text{compat}(Q, K)$ is a compatibility function defined in the following text.

Authors of the Transformer architecture [10] use scaled dot-product compatibility function defined as

$$\text{compat}_{\text{mul}}(Q, K) = \frac{QK^T}{\sqrt{d_q}}, \tag{2.11}$$

which adds $\frac{\cdot}{\sqrt{d_q}}$ operation to simple dot-product (multiplicative) compatibility function to adress possibly large values of dot-product QK^T. Such values can subsequently push softmax function to regions with a very small gradient.

We also use additive compatibility function from [1], as it is originally used in the Attention Based Clustering model defined in the next chapter:

$$(\text{compat}_{\text{add}}(Q, K))_{i,j} = \tanh(\boldsymbol{q}_i + \boldsymbol{k}_j)^T \boldsymbol{w}, \tag{2.12}$$

where $\boldsymbol{w}$ is vector of trainable parameters and $\boldsymbol{q}_i$, $\boldsymbol{k}_j$ are i-th and j-th row of Q and K. Differences between these two compatibility functions are discussed in [10] with the conclusion that both perform similary, but the scale-dot product compatibility function can be implemented more space and time efficiently.

Instead of performing single attention function, [10] proposed to first create h different d_q^M, d_q^M, d_v^M- dimensional representations as linear projections of Q, K, V (each row is projected independently) and apply attention function to each of these h projections. This enables the model to attend to different positions of the input set at once and was shown to achieve better results in various tasks. Such Multi-head attention function is defined as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O, \tag{2.13}$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V), \tag{2.14}$$

where $W_i^Q \in \mathbb{R}^{d_q \times d_q^M}$, $W_i^K \in \mathbb{R}^{d_q \times d_q^M}$, $W_i^V \in \mathbb{R}^{d_v \times d_v^M}$, $W^O \in \mathbb{R}^{d_v \times d_M}$ are learnable parameters. We use $d_q^M = \frac{d_q}{h}$ and $d_v^M = \frac{d_v}{h}$, d_M is a model parameter.

The self-attention (or intra-attention) is then attention with queries, keys and values being the same set of vectors. This allows us to relate elements

(vectors) of the input set between each other and compute representation containing information about interactions between the elements of the input set.

2.5 Set Transformer

The Set Transformer builds on the Transformer architecture [10] (mainly attention and self-attention) with a focus on processing set-structured data, motivated by the same type of problems as [9] and [12], discussed in previous sections. It is inspired by the general set-processing function from [12] (identical to architecture (2.6)). The Set Transformer is composed of blocks that use self-attention to process sets. This allows the blocks to encode pairwise and higher-order interactions between elements in the set. We are only interested in the individual blocks but show an example of the final model architecture at the end of this section.

In the following definitions, we consider two sets of n_X and n_Y elements $[\boldsymbol{x_1}, \ldots, \boldsymbol{x_{n_X}}]^T$, $[\boldsymbol{y_1}, \ldots, \boldsymbol{y_{n_Y}}]^T$ stored as rows of matrices $X \in \mathbb{R}^{n_X \times d_x}, Y \in \mathbb{R}^{n_Y \times d_y}$.

Main building part is Multi-head Attention Block (MAB) defined as:

$$\mathrm{MAB}(X, Y) = H + \mathrm{rFF}(H), \quad H = X + \mathrm{rFF}(\mathrm{MultiheadAtt}(X, Y)), \tag{2.15}$$

where rFF is row-wise feedforward layer. The output of $\mathrm{MAB}(X, Y)$ contains the same number of elements (rows of output matrix) as the input set X, dimension of the output elements d_M is given by weight matrix used for projection in (2.13).

Using MAB, we define Self-Attention Block (SAB):

$$\mathrm{SAB}(X) = \mathrm{MAB}(X, X), \tag{2.16}$$

capable of encoding relationships between elements of set X into the output set. By stacking variable number of SABs, we can model more complicated interactions between the elements of input set X. Such stack of L SABs is denoted SAB_L.

To encode a set with an arbitrary number of elements into k elements, we use Pooling by Multi-head Attention (PMA):

$$\mathrm{PMA}_k(X) = \mathrm{MAB}(S, X), \tag{2.17}$$

where $S = [\boldsymbol{i_1}, \ldots, \boldsymbol{i_k}]^T$ are trainable parameters and since S is the first input of the MAB, size of the output is $\mathrm{MAB}(S, X) \in \mathbb{R}^{k \times d_M}$.

To solve the $O(n^2)$ time complexity of SAB, authors of [7] used Induced Self-Attention Block instead:

$$\mathrm{ISAB(X)} = \mathrm{MAB}(X, \mathrm{PMA}_k(X)), \tag{2.18}$$

Rather than comparing elements of X between each other directly (which is the source of the quadratic complexity), ISAB compares them through k

inducing points S (learnable parameters of the PMA block). This modification reduces the time complexity of ISAB to $O(kn)$. We can again stack L ISABs, which we denote ISAB_L.

The overall Set Transformer architecture is structured as an encoder and decoder (similarly to 2.6). Encoder can be composed of L SAB or ISAB stacked blocks:

$$\text{Encoder}(X) = \text{SAB}_L(X) \qquad \text{Encoder}(X) = \text{ISAB}_L(X), \tag{2.19}$$

which transforms set $X \in \mathbb{R}^{n_X \times d}$ into a set representation $Z \in \mathbb{R}^{n_X \times d_M}$, where d_M is parameter of Encoder. Decoder can be, for example, defined as:

$$\text{Decoder}(X) = \text{rFF}(\text{SAB}(\text{PMA}_k(Z))) \in \mathbb{R}^{k \times d'_M}. \tag{2.20}$$

It maps the features Z into set with fixed number of elements k of defined length d'_M. If $k = 1$ (as it would be to obtain single label per set, for example to solve MIL problem), the SAB block in the Decoder can be omitted.

2.6 Spectral Clustering

Most of the models described in the two following chapters do not output cluster assignment directly, but their output can be transformed into a similarity matrix of the set to be clustered. A similarity matrix of set X is defined as $S \in \mathbb{R}^{n_X \times n_X}$, where $s_{i,j} \in [0, 1]$ describes the similarity between i-th and j-th element of X.

We decided to use *spectral clustering* to obtain cluster assignment from the similarity matrix, as it is already used with the ABC [3] model described in Section 3.2. Furthermore, there exists a method for estimating the number of clusters from the similarity matrix, which is particularly designed for spectral clustering (the number of clusters in a set is generally not known in the supervised clustering problem).

The set X can be viewed as a weighted graph $G = (V, E, W)$, where $V = \{v_1, \ldots, v_{n_X}\}$ are vertices representing elements $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n_X}\}$ of this set. The weight matrix is set equal to the similarity matrix $W = S$. Since S and consequently W are symmetric, graph G is undirected. Several variants of spectral clustering exist, and they differ in what type of graph Laplacian they use. We used spectral clustering implemented in Python library `Scikit-learn`, which uses normalised symmetric Laplacian [8]. The normalized symmetric Laplacian is defined as follows:

$$L_{\text{sym}} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}, \tag{2.21}$$

where:

$$L = D - W \tag{2.22}$$

is unnormalised Laplacian and D is a diagonal matrix with degrees of vertices v_i:

$$d_i = \sum_{j=1}^{n_X} w_{i,j} \tag{2.23}$$

on the diagonal.

The algorithm defined in [8] then follows by finding first k_X eigenvectors $\boldsymbol{u_1}, \ldots, \boldsymbol{u_{k_X}}$ of L_{sym} corresponding to k_X lowest eigenvalues. These eigenvectors form matrix $U \in \mathbb{R}^{n_X \times k_X}$ with $\boldsymbol{u_1}, \ldots, \boldsymbol{u_{k_X}}$ as columns. From U, matrix N is created by normalizing rows to norm 1. Rows $(\boldsymbol{n}_i)_{i=1,\ldots,n_X} \in \mathbb{R}^{k_X}$ of N are then fed as n_X points into the k-means algorithm to be clustered into k_X clusters. Since vectors $(\boldsymbol{n}_i)_{i=1,\ldots,n_X}$ are what's called a spectral embedding of elements $(\boldsymbol{x}_i)_{i=1,\ldots,n_X}$ of the original set X, the cluster number of $\boldsymbol{n}_i$ corresponds to cluster number of element $\boldsymbol{x}_i$.

Other types of Spectral clustering (described for example in [11]) also use eigenvectors of corresponding graph Laplacian and k-means but differ in some details.

We use the number of clusters k_X throughout the algorithm, but that is not known in the type of problem described in Section 1.2. In [11], authors describe the eigengap heuristic to solve this problem. This heuristic says to choose the number of clusters k_X so as that all eigenvalues $\lambda_1, \ldots, \lambda_{k_X}$ are very small and λ_{k_X+1} is relatively large. Using this heuristic, authors of [3] estimate the number of clusters as:

$$\text{NumClusters}(X) = \text{argmax}_{i \in \{1,\ldots,n\}}\{\lambda_i - \lambda_{i+1}\}, \tag{2.24}$$

where λ_i is the i-th largest eigenvalue of L_{sym}.

Various reasons for why Spectral clustering and the eigengap heuristic work are stated in [11].

The most computationally demanding part of this clustering process is eigenvalue decomposition, whose time complexity is $\mathcal{O}(n^3)$, where in our case n is the dimension of the similarity matrix S, which equals to the number of elements n_X in the set X.

Chapter 3
State Of The Art

In this chapter, we show two selected state of the art methods for solving supervised clustering problems. In Chapter 7, we evaluate these methods on artificial and real-world datasets (described in Chapter 5) and compare them to methods proposed in Chapter 4. Both methods are built as neural networks (functions with a large number of optimizable parameters and an appropriate loss function) and use the attention and self-attention mechanism defined in Section 2.4.

Deep Amortized Clustering model is the only model presented in this thesis that outputs cluster assignment directly (although iteratively), while Attention Based Clustering only produces a similarity matrix of elements in the input set. A kernel-based unsupervised clustering method must be used to obtain cluster assignment from such similarity matrix.

See Table 4.1 for a brief summarization of properties of the models described in this and the following chapters.

In the following definitions, we continue to follow the notation of set X and ground truth cluster numbers $\boldsymbol{y}$ from Section 1.2.

3.1 Deep Amortized Clustering

Part of the motivation behind Deep Amortized Clustering (DAC) [6] is to be able to efficiently learn a model that infers cluster assignment directly (in comparison to producing similarity matrix of elements in the input set), irrespective of the number of clusters in the set. To do this, DAC takes a set X with a variable number of elements of fixed size and infers members of clusters one cluster per iteration. Elements whose membership to some cluster is inferred at the current iteration are held out, and the resulting smaller set is fed into the model in the next iteration. This process is repeated until all elements are assigned to some cluster, or the maximum number of iterations is reached. Authors of DAC call this process of iterative clustering *Filtering*. Because DAC uses blocks from Set Transformer, it is capable of working with a variable number of elements in each set (variable number of inputs).

Blocks of Set Transformer that DAC model is composed of are defined in [7] and described in Section 2.5. Only scale-dot compatibility (2.11) in the

attention mechanism is used in this model. There are two types of the model that differ in the strategy of choosing the next cluster to infer: *Anchored Filtering* and *Minimum Loss Filtering*.

Anchored Filtering randomly samples one of the elements at each iteration. This sampled element $\boldsymbol{x}_a$ is called *anchor*, where a is row index of $\boldsymbol{x}_a$ in matrix X (input set). The model then proceeds to infer all elements of X that belong to the same cluster as the *anchor*. Elements inferred to belong to the same cluster as the anchor are held out, and the rest of the set is fed as input to the model again. Once all elements are assigned to some cluster (or the defined maximum number of iterations is reached), this iterative process ends.

Model's architecture can be described as:

$$H_X = \mathrm{ISAB}_{\mathrm{L}}(X), \qquad H_{X|a} = \mathrm{MAB}(H_X, \boldsymbol{h_a}), \tag{3.1}$$

$$H_m = \mathrm{ISAB}_{\mathrm{L}'}(H_{X|a}), \qquad \boldsymbol{m} = \mathrm{sigmoid}(\mathrm{rFF}(H_m)), \tag{3.2}$$

where $\boldsymbol{h_a}$ is a-th row of H_X and $\boldsymbol{m} = [m_1, \ldots, m_n]^T$ is a vector of probabilities of the elements of set X belonging to the same cluster as sampled anchor $\boldsymbol{x_a}$. The *holding out* of elements whose cluster is already inferred is done by setting the corresponding value of compatibility function in (2.10) to negative value with large magnitude. This forces the output of softmax to be zero, effectively discarding the hold out elements from the weighted sum of V.

The loss function is defined as:

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{m}, a, n_X) = \frac{1}{n_X} \sum_{i=1}^{n} \mathrm{BCE}(m_i, \mathbb{1}_{\{y_i = y_a\}}), \tag{3.3}$$

where BCE is Binary cross entropy function and $\mathbb{1}_{\{y_i=y_a\}}$ is 1 if the i-th and a-th elements are in the same cluster and 0 otherwise.

Minimal Loss Filtering uses following loss function:

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{m}, n_X) = min_{j \in \{1,\ldots,k_X\}} \left(\frac{1}{n_X} \sum_{i=1}^{n} \mathrm{BCE}(m_i, \mathbb{1}_{\{y_i = j\}}) \right), \tag{3.4}$$

and cluster to be inferred at the current iteration is based on the minimisation.

Choosing the next cluster to infer via *anchor* helps the model learn to cluster more complicated datasets easily. We only use *Anchored Filtering* and therefore do not show the detailed architecture for the *Minimal Loss Filtering*.

During the training process, we sample only one anchor per set in batch, compute $\boldsymbol{m}$ and optimize the weights using the loss function with Adam optimizer. This means complete clustering is not inferred while training. To obtain the full clustering, the iterative process described above is used. In an optimal situation, only k_X iterations are needed to infer the clusters. The time complexity of a feedforward of one set through the DAC model grows linearly with the number of elements n_X in the set X.

3.2 Attention Based Clustering

Authors of the Attention Based Clustering model (ABC) [3] were strongly motivated by processing data in context, providing an example of clustering characters according to which language they belong to. Consider some characters from Greek and Latin that resemble each other (for example, χ and X or capital of τ and T). It should be easier to cluster such characters correctly if given in the context of other characters from the same language (for example, in a word of some text). This focus on context corresponds with our interest in methods capable of inferring clusters with intra-cluster dependencies (for example, the Circles dataset described in Section 5.2).

The ABC model processes a set X with a variable number of elements of fixed size and outputs symmetric $n_X \times n_X$ matrix A, where $A_{i,j}$ describes the pairwise similarity between the i-th and j-th element of the set X. Clusters of set X are then inferred using spectral clustering (any unsupervised kernel-based clustering method can be used). For the use of such unsupervised clustering method, the number of clusters needs to be predicted first. To do this, authors of ABC use the eigengap method that was described along with the spectral clustering in Section 2.6.

For the following architecture definition, we consider the same matrix representation X of a set of elements and vector $\boldsymbol{y}$ with true indices of clusters as in the previous sections. Since the output of the ABC model is a prediction of the similarity matrix, we define $Y \in \mathbb{R}^{n_X \times n_X}$ to be ground-truth pairwise similarity matrix computed from $\boldsymbol{y}$. Element $y_{i,j} \in \{0, 1\}$ of Y describes if the i-th and j-th elements of the input set are similar ($y_{i,j} = 1$), or disimilar ($y_{i,j} = 0$).

The ABC model's architecture can be divided into embedding layer:

$$\mathcal{T}(X) = \mathrm{SAB}_L(\mathrm{rFF}_N(X)), \tag{3.5}$$

where rFF_N is N stacked row-wise feedforward layers with tanh activation function in between. Stacked SAB ensures information from each input element of set X is encoded into all elements of the output set $\mathcal{T}(X)$. The output of the embedding layer is of shape $\mathcal{T}(X) \in \mathbb{R}^{n_X \times d_M}$, where d_M is the parameter of rFF_N (2.3) and SAB (both rFF and SAB have parameter denoted d_M, that is set to a same value in $\mathcal{T}$). Symmetric similarity matrix is computed from embedding $Z = \mathcal{T}(X)$ with function κ:

$$\kappa(Z) = \frac{1}{2}[\mathrm{sigmoid}(\mathrm{compat}(Q, K)) + \mathrm{sigmoid}(\mathrm{compat}(Q, K))^T], \tag{3.6}$$

where $Q = \mathrm{rFF}_\mathrm{Q}(Z) \in \mathbb{R}^{n_X \times d_M}$, $K = \mathrm{rFF}_\mathrm{K}(Z) \in \mathbb{R}^{n_X \times d_M}$ and rFF_K, rFF_Q are single row-wise feedforward layers.

With $\mathcal{T}$ and κ defined, output of the model $A = \mathrm{ABC}(X) \in \mathbb{R}^{n_X \times n_X}$ is computed simply as:

$$\mathrm{ABC}(X) = \kappa(\mathcal{T}(X)). \tag{3.7}$$

We experiment with both types of compatibility functions for the attention inside SAB and in (3.6). These models are denoted as $\text{ABC}_{\text{MULTI}}$ and ABC_{ADD}.

The loss function is defined as:

$$\mathcal{L}(\text{ABC}(X), Y) = \frac{1}{n_X^2} \sum_{i,j} \text{BCE}(\text{ABC}(X)_{i,j}, Y_{i,j}). \tag{3.8}$$

Since the loss function is optimized using only Y as ground-truth information (true pairwise similarity), the ABC model can be trained without ground truth cluster assignment.

During training, we only compute the similarity matrix (output of ABC) without inferring clusters and use Adam optimizer to optimize the model. Because stacked SAB is used in the ABC model, its time complexity is $\mathcal{O}(n_X^2)$ during training. To infer clusters, we use spectral clustering with the eigengap method whose time complexity is $\mathcal{O}(n_X^3)$.

Chapter 4

Proposed methods

In this chapter, we propose new methods to solve the supervised learning problem. Those models are based on definitions in Chapter 2 and in contrast to SOTA methods described in the previous chapter, they do not (apart from MIL_{PMA}) use the attention mechanism.

In Table 4.1, we briefly summarise properties of the models described in this and previous chapters.

4.1 Multiple Instance Learning with Pair Representation

We propose a method transferring a supervised clustering problem into a multiple instance learning one. This is done simply by estimating the pairwise similarity of elements in input set X, while representing a pair of elements from X as another set.

Assume we have set X of elements that we want to cluster. We sample random pair of elements $\boldsymbol{x}_i$, $\boldsymbol{x}_j$ of set the X and create a new set $X_{\text{pair}_{i,j}}$ using *Pair Representation*. Using model solving multiple instance learning problems, we then process $X_{\text{pair}_{i,j}}$ and predict the probability of pair $\boldsymbol{x}_i$, $\boldsymbol{x}_j$ being in the same cluster. If we create all possible pairs of elements from X, we can construct a pairwise similarity matrix and proceed to infer the clusters using *eigengap method* for the number of clusters estimation and *Spectral clustering* in the same fashion as with the ABC model in Section 3.2.

Given set $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n_X}\}$ and a pair of it's elements $\boldsymbol{x}_i, \boldsymbol{x}_j$, we construct the *Pair Representation* as:

$$X_{\text{pair}_{i,j}} = \{\boldsymbol{x}'_1, \ldots, \boldsymbol{x}'_{n_X}\}, \tag{4.1}$$

where

$$\boldsymbol{x}'_k = [\boldsymbol{x}_k^T, \boldsymbol{x}_i^T, \boldsymbol{x}_j^T]^T \in \mathbb{R}^{3d}, \tag{4.2}$$

that is, we concatenate features of the original element with index k and features of the elements from the sampled pair. Motivation for this representation follows the discussion in Chapter 1 about our focus on datasets with intra-cluster dependencies. In such case, pairwise similarity depends

not only on the sampled pair alone but rather on the pair in the context of the whole set X. This approach to embed the context into the pairwise similarity prediction is simpler, although more naive than the one using the self-attention mechanism in the ABC model.

We represent the pair set as matrix $X_{\text{pair}_{i,j}} \in \mathbb{R}^{n_X \times 3d}$ and define the MIL model used in this thesis as:

$$\text{MIL}(X_{\text{pair}_{i,j}}) = \text{rFF}_{N'}(g(\text{rFF}_N(X_{\text{pair}_{i,j}}))), \tag{4.3}$$

where rFF_N denotes N stacked row-wise feedforward layers (with ReLU activation functions in between) and $g(\cdot)$ is a pooling function. We experiment with two types of $g(\cdot)$, first is a simple feature-wise mean$(\cdot)$ and we call such model MIL_{MEAN}.

Next, we use the Pooling by Multi-head Attention with $S = [\boldsymbol{i_1}]$:

$$g(\cdot) = \text{PMA}_1(\cdot), \tag{4.4}$$

which aggregates n_X fixed sized elements into 1 element of the same size using the attention mechanism. In some experiments, MIL_{PMA} using attention in the pooling operation achieves greater results than MIL_{MEAN}, and we attribute that to the comparison of elements in set with each other that takes place in the attention inside PMA.

The loss function is:

$$\mathcal{L}(\text{MIL}(X_{\text{pair}_{i,j}}), y_{i,j}) = \text{BCE}(\text{MIL}(X_{\text{pair}_{i,j}}), y_{i,j}), \tag{4.5}$$

where $y_{i,j}$ is element of the true similarity matrix Y (see Section 3.2), that is, it equals 1 if the i-th and j-th elements are in the same cluster and 0 otherwise.

To cluster a set X, we sample all possible unordered pairs with replacement and construct pair sets $X_{\text{pair}_{i,j}}$. Since we can create $\frac{(n_X+1)n_X}{2}$ pairs from a set of n_X elements and each set $X_{\text{pair}_{i,j}}$ representing one pair contains n_X elements of size $3d$ (d is the size of element from the input set), this method is very time and memory consuming. All pair sets are then fed to MIL model and we construct the prediction of similarity matrix A from the results. From this matrix, we can infer clusters in the same way as with the ABC model's output.

To make the training process more tractable, we only sample 50 pairs from each set X on the input.

During training, we do not compute the similarity matrix and do not infer the clusters, time complexity then depends on whether we sample all pairs: $\mathcal{O}(n_X^2)$, or if we sample defined amount of pairs from each set (as we do in this thesis): $\mathcal{O}(n_X)$. To cluster the dataset, we must evaluate the MIL model on all possible pairs ($\mathcal{O}(n_X^2)$) to obtain the similarity matrix and then use spectral clustering with the eigengap method to obtain the clusters ($\mathcal{O}(n_X^3)$).

4.2 Permutation Equivariant Model with Same Cluster Representation

The last architecture, Permutation Equivariant Model (PEM), is inspired by the DAC model's process of inferring clusters. Instead of self-attention, in this model we use *Equivariant layer* defined in Section 2.3 to obtain useful representation of the input set.

We sample random element $\boldsymbol{x_a}$ of the input set X and call it *anchor*. The whole set X with $\boldsymbol{x_a}$ is then fed into the PEM model, which returns a probability of each element of the set being in the same cluster as the *anchor* (*Same Cluster Representation* of the output). To infer cluster membership of all elements in the set X, we choose every element, one by one, of the set as the anchor and pass it to the model with the whole set. From the outputs, we construct a pairwise similarity matrix and use the same process as with ABC and MIL (spectral clustering with eigengap method) to predict the clusters.

Considering the set is represented as matrix $X \in \mathbb{R}^{n_X \times d}$, we first encode X using:

$$H_e = [\overbrace{(\text{EquivLayer} \circ \text{ELU}) \circ \cdots \circ (\text{EquivLayer} \circ \text{ELU})}^{N_e}](X), \tag{4.6}$$

with ELU (Exponential Linear Unit) as an activation function. This activation function was used in [12] when building models with the Equivariant Layer. N_e is the number of (EquivLayer $\circ$ ELU) blocks. Anchor is encoded using:

$$\boldsymbol{h}_a = [\overbrace{(\text{rFF} \circ \text{ReLU}) \circ \cdots \circ (\text{rFF} \circ \text{ReLU})}^{N_a}](H_e), \tag{4.7}$$

where N_a is the number of (rFF $\circ$ ReLU) blocks. Each row of H_e is then replaced with a concatenation of itself and $\boldsymbol{h}_{\boldsymbol{a}}^T$, creating $H_{ea} \in \mathbb{R}^{n_X \times (d_e + d_a)}$, where d_e is the size of the encoded elements in H_e and d_a is the size of $\boldsymbol{h}_a$.

The i-th row of the matrix H_{ea} contains a concatenation of features of the i-th row of the matrix H_e and features of the encoded anchor $\boldsymbol{h}_a$. This tells the model we want to infer elements from the cluster to which the anchor belongs. Finally, we compute vector of probabilities m as:

$$\boldsymbol{m} = [\overbrace{(\text{EquivLayer} \circ \text{ELU}) \circ \cdots \circ (\text{EquivLayer} \circ \text{ELU})}^{N_{ea}} \circ \text{rFF}](H_{ea}), \tag{4.8}$$

where m_i is the probability of i-th element of X belonging to the same cluster as the anchor. N_{ea} is the number of (EquivLayer $\circ$ ELU) blocks.

The loss function is defined as:

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{m}, a, n_X) = \frac{1}{n_X} \sum_{i=1}^{n_X} BCE(m_i, \mathbb{1}_{\{y_i = y_a\}}), \tag{4.9}$$

where $\mathbb{1}_{\{y_i = y_a\}}$ is 1 if the i-th and a-th elements of X are in the same cluster and 0 otherwise.

To compute the similarity matrix A, we sample each element of X as the anchor and feed it to the model with X. From outputs $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{n_X}$ corresponding to $a = 1, \ldots, n_X$, we construct matrix $M \in \mathbb{R}^{n_X \times n_X}$ with vectors $\boldsymbol{m}_i$ stored as rows. To ensure A is symmetric, we compute it as:

$$A = \frac{1}{2}(M + M^T). \tag{4.10}$$

The output of PEM (Same cluster representation) is of the same form as the output of DAC. Both models output vector $\boldsymbol{m} \in \mathbb{R}^{n_X}$, where m_i is the probability that i-th element of the input set belongs to the same cluster as the anchor (used in both PEM and DAC models' forward passes). This lends itself to infer clusters using PEM with the same iterative process as with the DAC model. We experimented with this process to infer clusters with PEM but achieved worse results than using the method described above (creating similarity matrix and inferring clusters using spectral clustering).

Throughout the training process, we sample only one anchor per set in batch. Since PEM consists of rFFs and Equivariant layers, the time complexity during training is $\mathcal{O}(n_X)$. To infer the clusters in set X, we must pass X into the model n_X-times (with each element of X as the anchor each time) to create the similarity matrix. This leads to time complexity $\mathcal{O}(n_X^2)$ to create the similarity matrix and $\mathcal{O}(n_X^3)$ to infer the clusters using spectral clustering.

Model	Output	Attention	Train complexity	Clustering complexity
DAC	Assignes clusters directly.	Uses only scale-dot compatibility.	$\mathcal{O}(n_X)$	$\mathcal{O}(k_X n_X)$
ABC	Outputs similarity matrix.	Uses both compatibility functions.	$\mathcal{O}(n_X^2)$	$\mathcal{O}(n_X^3)$
MIL	Outputs pairwise similarity, from which a similarity matrix is constructed.	Uses attention only for attention pooling.	$\mathcal{O}(n_X)$	$\mathcal{O}(n_X^3)$
PEM	Same cluster representation of output. Similarity matrix is constructed.	Does not use attention.	$\mathcal{O}(n_X)$	$\mathcal{O}(n_X^3)$

Table 4.1: A short summary of examined models. Training (time) complexity accounts for training process described in each section for corresponding model. Clustering process differs from training (evaluating all pairs with MIL, iterating through all elements as anchors with PEM) and hence the different clustering (time) complexity.

Chapter 5

Datasets

We use three different datasets to test the models described in Chapters 3 and 4. The first two, Mixture of Gaussians (MoG) and Circles, are artificial. This allows us to test the models on sets with different parameters, for example, lower or higher number n_X of elements and fixed or variable number of clusters in a set. We use MoG to test scalability to the number of elements in a set of examined models and Circles to experiment with the model's ability to cluster sets with intra-cluster dependencies.

The third dataset, Newspaper, consists of pages of Czech newspaper `Právo`. We are given individual textboxes on each page, and the goal is to divide those textboxes into articles. This primarily serves to show the use of the supervised clustering model on real-world data.

5.1 Mixture of Gaussians

This is a well-known dataset with a simple cluster structure (two-dimensional Gaussian distribution) and can be usually seen as a toy-problem case to show various unsupervised clustering methods. To generate this dataset, we slightly modify a process described in appendix B of [6]. We show this process with our modifications in the following text. We use this dataset to compare scalability to the size of sets of models discussed in this thesis.

A set of the MoG (Mixture of Gaussians) dataset consists of n_X elements (points on a two-dimensional plane) structured into several clusters. Number of elements n_X is defined as parameter. Each cluster is defined by Gaussian distribution with two-dimensional mean and variance. The goal of learned model is to predict the true cluster y_i $(i = 1, \ldots, n_X)$ for each element in the set. A sample of few such sets is shown in Figure 5.1.

Each set is generated using the following process. First, a random number k_X of clusters is chosen from specified bounds $[k_{\min}, k_{\max}]$:

$$k_X \sim \mathrm{Unif}(k_{\min}, k_{\max}). \tag{5.1}$$

A true cluster number y_i is generated for each element using categorical

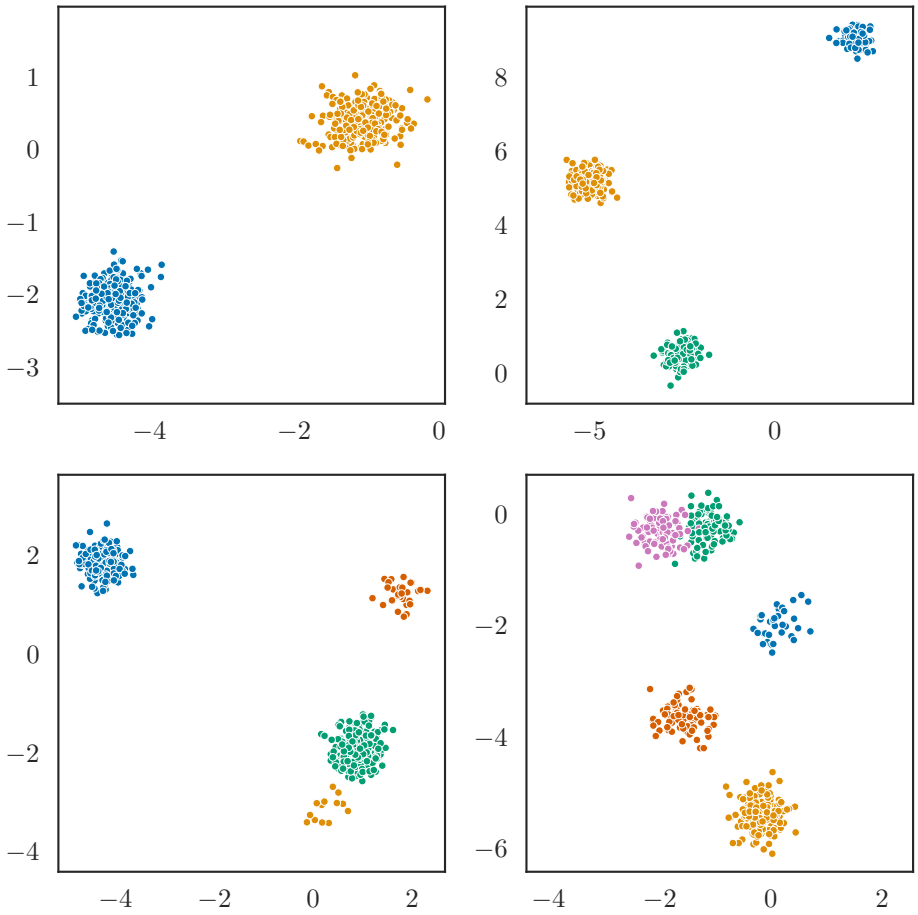

Figure 5.1: Examples of sets from the MoG dataset. Displayed are sets with 2, 3, 4 and 5 clusters respectively.

distribution with custom probabilities p sampled using Dirichlet distribution:

$$p \sim \mathrm{Dir}(\overbrace{[1, \ldots, 1]}^{k_X}), \tag{5.2}$$

$$(y_i)_{i=1}^{n_X} \sim \mathrm{Cat}(p). \tag{5.3}$$

For each cluster, two-dimensional mean $\boldsymbol{\mu}_j$ and variance $\boldsymbol{\sigma}_j$ are sampled from Normal and logNormal distribution:

$$(\boldsymbol{\mu}_j)_{j=1}^{k_X} \sim \mathrm{Normal}([0,0]^T, 9I), \tag{5.4}$$

$$(\boldsymbol{\sigma}_j)_{j=1}^{k_X} \sim \mathrm{logNormal}(\log(0.25)[1,1]^T, 0.01I). \tag{5.5}$$

Finally, each element is generated as:

$$(\boldsymbol{x}_i)_{i=1}^{n_X} \sim \mathrm{Normal}(\boldsymbol{\mu}_{y_i}, \mathrm{diag}(\boldsymbol{\sigma}_{y_i}^2)). \tag{5.6}$$

As is apparent from the generation process above, we generate two features for each element $\boldsymbol{x}_i \in \mathbb{R}^2$ (x and y coordinate in a two-dimensional plane) and a true cluster number y_i.

5.2 Circles

This dataset was used in [3] (where the ABC model was introduced) to simulate real-world datasets with intra-cluster dependencies and cluster intersections, which require the use of context during the clustering process. Such problems were discussed in Section 3.2 in more detail. We test all models examined in this thesis on this dataset to compare them on sets with a structure more complicated than simpler MoG.

Each set again consists of n_X points (elements) lying on k_X likely to be overlapping circles. Number n_X of points in a set is defined as parameter during training and evaluation. Model's task is to cluster those points according to the circle they lie on. A sample of sets from the Circles dataset with various number of clusters is displayed in Figure 5.2 .

To generate one set of the Circles dataset, we use the following process (used in [3]). A random number k_X of clusters is sampled from given bounds $[k_{\min}, k_{\max}]$:

$$k_X \sim \mathrm{Unif}(k_{\min}, k_{\max}). \tag{5.7}$$

For each cluster, center $\boldsymbol{O}_j$ of the circle constituting the j-th cluster is generated as:

$$(r_{cj})_{j=1}^{k_X} \sim \sqrt{\mathrm{Unif}(0,1)} \qquad (\theta_{cj})_{j=1}^{k_X} \sim \mathrm{Unif}(0, 2\pi) \tag{5.8}$$

$$\boldsymbol{O}_j = \begin{bmatrix} r_{cj}\cos(\theta_{cj}) \\ r_{cj}\sin(\theta_{cj}) \end{bmatrix}. \tag{5.9}$$

Radius of each circle (cluster) is randomly chosen from Normal distribution and clamped to the range $[0.2, 0.4]$:

$$(r'_j)_{j=1}^{k_X} \sim \mathrm{Normal}(0.3, 0.1) \qquad r_j = \min[\max(r'_j, 0.2), 0.4]. \tag{5.10}$$

Before generating true cluster assignments randomly, we ensure each cluster contains at least one element (point):

$$(y_i)_{i=1}^{k_X} = i \tag{5.11}$$

The rest of the true labels is then generated using categorical distribution with uniform probabilities p:

$$p = [\overbrace{1/k_X, \ldots, 1/k_X}^{k_X}] \qquad (y_i)_{i=k_X+1}^{n_X} \sim \mathrm{Cat}(p). \tag{5.12}$$

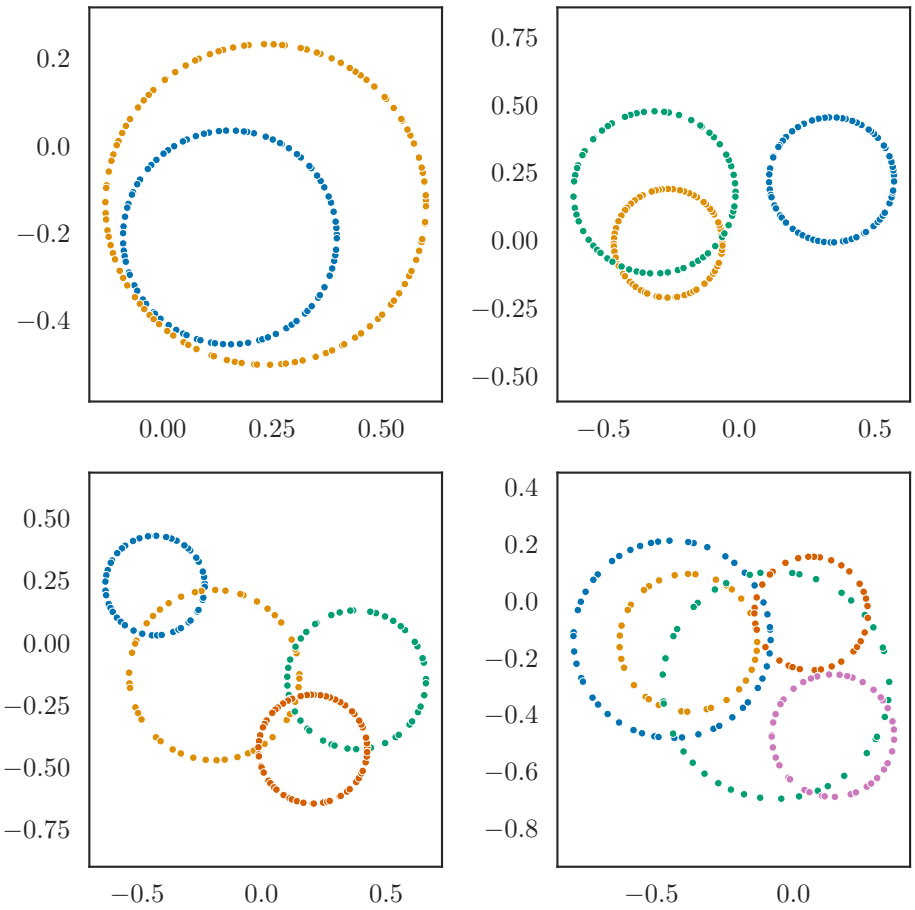

Figure 5.2: Examples of sets from the Circles dataset. Displayed are sets with 2, 3, 4 and 5 clusters respectively.

Afterwards, each element of set is generated as:

$$(\theta_i)_{i=1}^{n_X} \sim \text{Uniform}(0, 2\pi) \tag{5.13}$$

$$\boldsymbol{x}_i = \boldsymbol{O}_{y_i} + \begin{bmatrix} r_{y_i} \cos(\theta_i) \\ r_{y_i} \sin(\theta_i) \end{bmatrix}. \tag{5.14}$$

As in the MoG dataset, each element $\boldsymbol{x}_i \in \mathbb{R}^2$ consists of two features (x and y coordinate in a two-dimensional plane) and true cluster number y_i.

5.3 Newspaper

To showcase the examined models on real-world data, we use pages of Czech daily newspaper `Právo`, kindly provided by `Newton Media, a.s.`[1] for re-

[1] `https://www.newtonmedia.cz`

search use only. The dataset is not public and cannot be provided as a part of this work.

Each page is represented as one set. One page contains a variable number of textboxes (called elements in the vocabulary used throughout this thesis). Each textbox (denoted as $\boldsymbol{x}_i$) belongs to one article given by a true cluster number y_i. Examples of two pages from the Newspaper dataset are shown in Figure 5.3. Textboxes contain not only bodies of text, but also headlines,

Figure 5.3: Examples of pages from the Newspaper dataset. Textboxes bounded with rectangle of same colour are from a same cluster (article).

image captions, author names, citations and in some cases clusters with only one textbox occur (such as `PROFESE` textbox in the bottom part of the page on the right in Figure 5.3).

Textboxes on each page were divided into articles by Ing. Radek Mařík, CSc.[2] using expert rules to create a dataset we could use to train our models. My supervisor Ing. Jan Drchal, Ph.D. discarded pages, where clustering done via expert rules was not optimal to create a cleaner dataset. For tabloid-style newspapers with a more complicated structure (position and shape) of articles, expert rules are not a feasible option to create the training dataset automatically, and manual annotation would need to be used.

For each textbox, we have six features $\boldsymbol{x}_i \in \mathbb{R}^6$, and those are x and y coordinates describing the position of lower-left and upper-right corner of the textbox, the font size used in the given textbox and the length of the

[2] `https://comtel.fel.cvut.cz/en/users/marikr`

text as character count. All features were standardized to be in the range $[0, 1]$ before any further division into train and test parts. Since we also have the full-text content of each textbox, we could use it to construct features conveying the semantics of the text. In this work, we were able to achieve quite favorable results without such features, but they might help to learn a better model if some newspaper with a more complicated article structure (usually tabloid type) was used.

In total, the Newspaper dataset contains 349 pages (sets) and 2295 articles. This is a relatively small dataset (in comparison to the generated Circles and MoG datasets we use, or most datasets used for modern machine learning tasks).

The distribution of the number of clusters (articles) per page in the Newspaper dataset is shown in the top histogram of Figure 5.4. Apart from one outlier (page with 24 articles), the number of articles is distributed around a mean value 6.5. The bottom histogram shows the distribution of the number of elements (textboxes) per page.

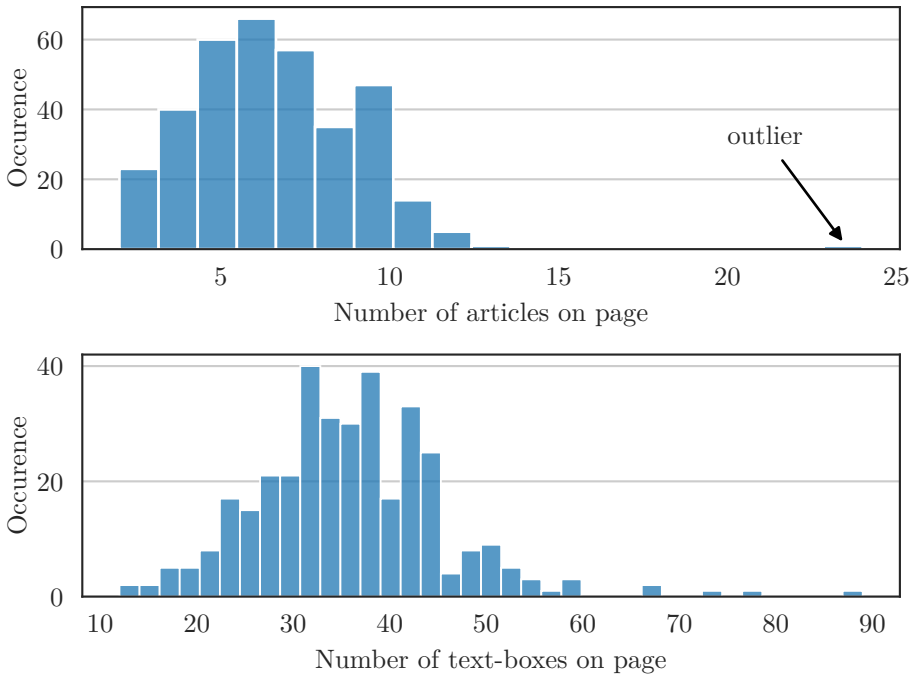

Figure 5.4: Histogram showing distribution of number of articles and textboxes per page in Newspaper dataset.

We also explore the distribution of features describing textboxes throughout all pages of the Newspaper dataset. The left histogram in Figure 5.5 shows one particular font size is used in the majority of all textboxes, which is expected as most textboxes contain body text of the article. There are some outliers, mainly in higher values corresponding to titles and quotes. The

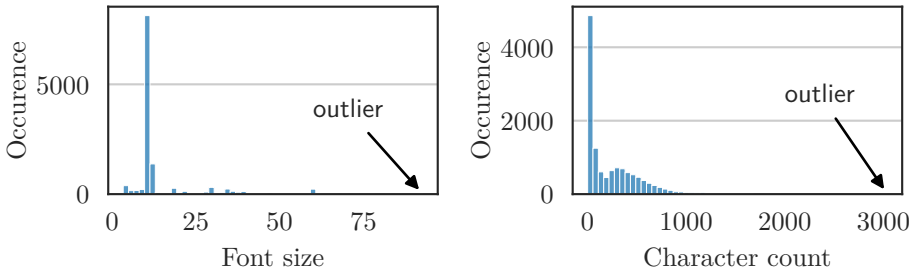

Figure 5.5: Histogram showing distribution of font sizes and character counts throughout all textboxes in the Newspaper dataset.

distribution of the number of characters in textboxes throughout the whole Newspaper dataset is displayed in the histogram on the right half of Figure 5.5. Apart from a large number of outliers in low values and few outliers in very high values, the distribution looks as expected. Low-value outliers correspond mostly to textboxes with only 1 or 2 characters. This is caused usually by some artefacts such as blue boxes next to subtitles or abbreviations in infographics (both shown in Figure 5.6).

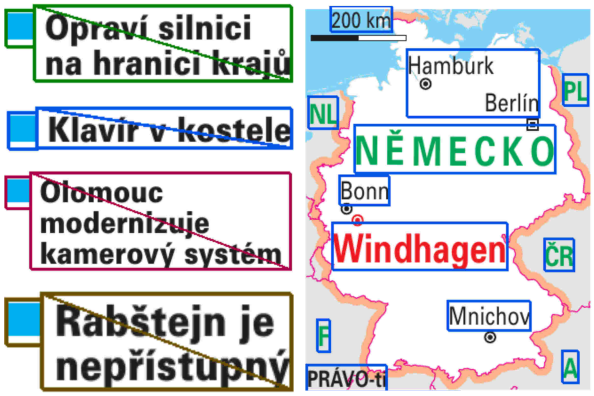

Figure 5.6: Textboxes with low number of characters. Blue boxes on the right are counted as one character. Abbreviations such as the ones on the map usually occur in an infographic.

Features describing position of textbox are displayed in Figure 5.7. Horizontal position (left histograms) shows regular positioning of textboxes into columns. Right histograms showing vertical position tell no interesting information.

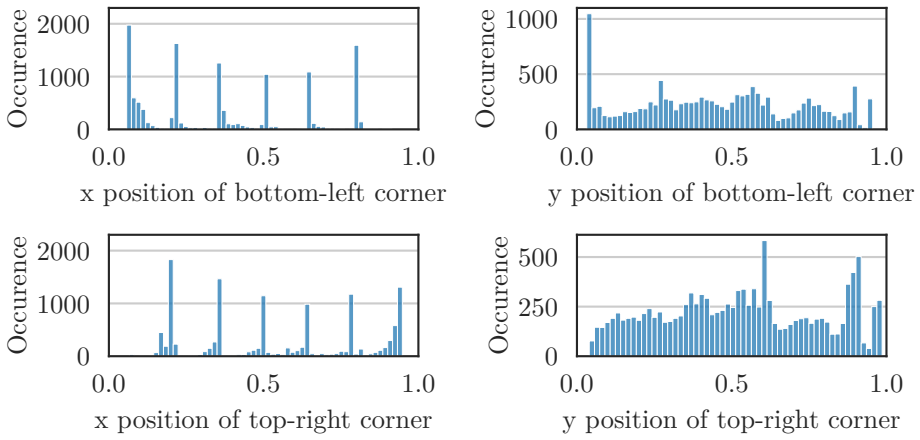

Figure 5.7: Histograms showing distribution of features describing position of textboxes in Newspaper dataset. Horizontal position is given by x, vertical by y. Values are standardised to be in the range [0,1].

Chapter 6

Implementation

All models are implemented using `Python`[1] mainly with `PyTorch`[2] and `NumPy`[3] libraries. The DAC[4] and ABC[5] models were implement by their authors in `PyTorch` and we used their implementation in our code with modifications to facilitate for all used datasets. We further implemented the additive compatibility function to work with DAC's attention implementation and used it in ABC instead of their attention implementation because DAC's implementation was faster. We also used Equivariant layer[6] implemented by its author. For spectral clustering, we used implementation from scikit-learn[7] Python library. For the proposed models, we used our implementation. The MIL model's implementation is also available on GitHub[8] with examples and documentation.

To generate the MoG dataset, we used the implementation of DAC's authors in PyTorch, which we modified according to the process described in Section 5.1. Generation of Circles dataset was also implemented in PyTorch by ABC's authors but had to be modified to allow generating sets with a variable number of clusters.

The same training and evaluation pipeline is adapted to be used with all datasets from Chapter 5. This makes it easy to add new datasets to train and test the models on.

The training process is done on batches of data with size defined by batch size B. This means each batch contains B sets, where each set generally contains a different number of elements n_X. The model processes one batch, computes the value of the corresponding loss function, and updates its parameters using an optimizer (back-propagation). All models in this thesis use Adam as the optimizer.

PyTorch represents data as multidimensional tensors and their shape in

[1] `https://www.python.org`
[2] `https://pytorch.org`
[3] `https://numpy.org`
[4] `https://github.com/ICLR2020anonymous/dac`
[5] `https://github.com/DramaCow/ABC`
[6] `https://github.com/manzilzaheer/DeepSets`
[7] `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html`
[8] `https://github.com/jakubmonhart/mil_pytorch`

our implementation corresponds to shapes of matrices in this thesis. We intentionally set the number of elements in set n_X to a constant value when using the MoG and Circles datasets to allow for faster processing of batches of data. In such case, a tensor with a batch of data is of shape $[B, n_X, d]$, where d is a fixed number of features of an element.

In the case of the Newspaper dataset, where each set (page) contains a different number of elements (textboxes), the whole batch cannot be represented as one tensor because shapes of all dimensions of tensor must be fixed. Because of this, we save each set into a separate tensor of shape $[1, n_X, d]$. The model processes B such tensors and aggregates the gradients. The weights of the model are then optimized using the aggregated gradients.

As for hardware, we used Research Center for Informatics[9] High Performance Computing cluster to train and evaluate the models. This enabled us to train more models faster using their GPUs on a server.

To organize all experiments (at the time of writing this chapter, we have more than 900 individual runs logged - including hyperparameter searches and training and testing runs), we used Weights and Biases [2] tracking tools.

The implementation of all models, along with code for the MoG and Circles dataset generation can be found in the `src/` folder of the attachement `code.zip`. Scripts for running individual models on the synthetic datasets are to be found in the folder `scripts/`. The implementation is also available as a GitHub repository[10].

6.1 Bug in DAC's implementation

We came across a bug in the implementation of DAC by its authors. When creating stacked SAB or ISAB block, in file `dac/neural/attention.py`, lines 50-52 and 81-83 create the required number of blocks, but they are all copies of each other, sharing the same set of weights (parameters). This effectively causes the model to iterate the data through the same SAB (or ISAB) layer L times (L is the number of required stacked SABs or ISABs) instead of having L blocks with parameters updated individually. This might have been caused by a different behavior of the original implementation based on the used version of PyTorch. We implemented this part of DAC correctly in our code.

[9] `http://rci.cvut.cz`

[10] `https://github.com/jakubmonhart/supervised-clustering-methods`

Chapter 7

Experiments

Models described in Chapters 3 and 4 were tested on datasets from Chapter 5 in various experiments. We start with experiments on the Circles dataset to test the model's ability to infer overlapping clusters with intra-cluster structure. To explore the input size scalability of the models (we are interested in a growing number of elements in the input set), we test the models on the Mixture of Gaussians dataset. Finally, we evaluate the models on the real-world Newspaper dataset.

Before showing the results, we specify a training procedure we used to optimize the models and an evaluation procedure used to obtain the metrics from the predicted clustering. For each model on each dataset, we first carried out a hyperparameter search to find the optimal setting. This was done by experimenting with various hyperparameter combinations and choosing the setting with which the model achieved lowest loss on the validation dataset. For the artificial datasets (Circles and MoG), we generated a training dataset consisting of 100k sets, each with 100 elements (number n_X of elements is later specified for various experiments, but the hyperparameter search was conducted with $n_X = 100$). The number of sets in the training dataset was reduced to 10k for both MIL models, as they are significantly more time-consuming than the rest of the models but trained well on this smaller dataset. From this training dataset, 5% was held out from optimization (back-propagation) and used for early stopping. The trained model was then evaluated on generated validation dataset with 1k sets. This procedure was repeated only three times with both MIL and ABC_{ADD} models due to long training times, but five times with all other models. Best hyperparameters were chosen based on the model's results on the validation dataset. Since Circles and MoG are synthetic datasets, the same repeated process was used to test the model and obtain the reported results.

The Newspaper dataset is relatively small (349 sets with approximately 30-40 elements per set). We left 10% of the data (testing part), which was not used before the final evaluation. The rest of the data (training part) was divided into five parts and individual parameter settings were evaluated using 5-fold cross-validation during hyperparameter optimization. During this cross-validation, one fold of the training part is used as a validation dataset at a time, and the rest of the folds is used as a training dataset.

5% of the training dataset is used for early stopping. Best hyperparameters were chosen based on the model's results on validation datasets during the cross-validation. After best hyperparameters are found, models are trained on the whole train part and tested on the test part (10% of the original data). During this evaluation, 5% of the training part is left out for early stopping. This process for final evaluation is repeated five times to obtain the reported results.

For optimization, we used Adam optimizer for each model on each dataset.

To compare true and predicted clustering, we need a suitable metric. This metric must output the same value when numbers (indices) of clusters are changed in either true or predicted clustering. We mainly use the Adjusted Rand Index (ARI) because of its clear interpretation and Normalized Mutual Information (NMI) to confirm the ARI results. Both metrics are symmetric and return a similarity of the two clusterings in the range $[0, 1]$.

Given true and predicted clusterings $\boldsymbol{y}_{\text{true}}, \boldsymbol{y}_{\text{pred}} \in [1, \ldots, k_X]^{n_X}$ of set $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n_X}\}$, simple Rand Index considers all possible pairs of elements and computes number of pairs that belong to same cluster according to both clusterings $\boldsymbol{y}_{\text{true}}, \boldsymbol{y}_{\text{pred}}$ and number of pairs that belong to different clusters according to both of the clusterings. Value of Rand Index is then computed as a sum of those two numbers divided by the total number of pairs. From this description, Rand Index can be considered as an analogy to accuracy. Adjusted Rand Index is Rand Index corrected for chance (see [4] for precise definition). This means random labeling has an ARI score near 0 and perfect labeling has an ARI score equal to 1.

As another measure, we use normalization of Mutual Information, which computes *agreement* of true and predicted clustering, normalized into the range [0,1]. Value 0 corresponds to no mutual information (as defined in information theory) between true and predicted clustering, and value 1 corresponds to perfect correlation (perfect predicted clustering). In contrast to ARI, NMI is not adjusted for chance.

In most cases, the order of models according to ARI values in an experiment corresponds to the order of models according to NMI values.

We use box plots to compare results of an experiment graphically. Each box corresponds to one experiment, which consists of 3 or 5 (depending on the model) measurements of given metric. We show sample box plot with description in Figure 7.1 for illustration.

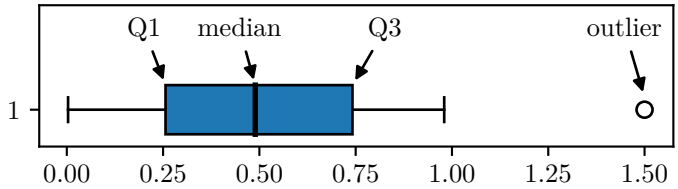

Figure 7.1: Illustrative box plot

Consider the measurements sorted in an array, divided into halves. Q1 denotes a first quartile, which is a median of the lower half of the array, Q3 is a third quartile, which is a median of the upper half of the array. Space between first and third quartile is called interquartile range (IQR = Q3 − Q1). Left whisker displays either Q1 − 1.5IQR or the lowest sample from the array, depending on which is higher. Analogously, the right whisker displays either Q3 + 1.5IQR or the highest sample from the array, depending on which is lower. Any data points from the array that are not in between the two whiskers are plotted as an outlier (small circle).

In the following tables, we show optimal hyperparameter settings for individual models that were used during the experiments.

Parameter	**Circles**	**MoG**	**Newspaper**
Learning rate	1e−4	3e−4	1e−4
Batch size	64	64	8
d_M — dimension of Multi-head attention output	256	128	512
$L = L'$ — number of ISAB blocks	8	4	4
h — number of heads in Multi-head attention	8	4	8
k — number of inducing points in ISAB	32	32	32
early stopping — number of epochs after achieving the lowest validation loss before the learning is stopped	10	10	40
number of learnable parameters (thousands)	8621	1107	17215

Table 7.1: DAC hyperparameters

7.1 Circles

Using the Circles dataset described in Section 5.2, we explore the model's ability to cluster sets with intra-cluster dependency — a circle structure in the case of this dataset. As a default dataset setting, we consider the number of elements in set to be fixed to $n_X = 100$ and number of circles (clusters) k_X drawn randomly (5.7) from range [2,6] for each set.

To see if the number of elements in a set affects the model's performance

Parameter	**Circles, MoG**		**Newspaper**	
ABC type	ADD	MULTI	ADD	MULTI
Learning rate	1e−4	1e−4	1e−4	3e−4
Batch size	32	32	8	8
Hidden size — defines size of rFF and parameter d_M of SAB in (3.5)	256	256	512	512
N — number of rFF layers in (3.5)	2	2	2	2
L — number of SAB blocks in (3.5)	8	8	2	2
h — number of heads in Multi-head attention	8	8	4	4
early stopping — number of epochs after achieving the lowest validation loss before the learning is stopped	10	10	10	10
number of learnable parameters (thousands)	2304	2303	2891	2891

Table 7.2: ABC hyperparameters

while keeping the other dataset settings fixed, we run experiments in Subsection 7.1.2.

Similarly, we test the effect of fixing the number of clusters in all sets in Subsection 7.1.3.

Due to a very long computation time for training MIL_{PMA} model and large traffic on the RCI cluster used for the experiments, we were not able to complete the necessary computations in time to present the results of this model on the Circles dataset.

7.1.1 Dataset with default setting

Before any modifications of the dataset, we report comparison of the model on the default settings in Table 7.5. Results from this experiment are further displayed using box plot in Figure 7.2.

From the box plot, we see both versions of the ABC model are (apart from being best performing) almost deterministic compared to the rest of the models. This is probably caused by the lack of stochasticity in ABC's training process, where all elements of the input set are processed and used in back-

Parameter	**Circles**		**MoG**		**Newspaper**	
MIL type	MEAN	PMA	MEAN	PMA	MEAN	PMA
Learning rate	1e−3	1e−3	1e−3	1e−3	1e−3	1e−3
Batch size	32	32	32	32	16	16
Hidden size — defines size of both rFFs in (2.6)	256	256	256	256	256	512
$N = N'$ — number of rFF layers in (2.6)	3	3	3	3	3	3
h — number of heads in Multi-head attention (applies only to MIL_{PMA})	-	4	-	4	-	4
early stopping — number of epochs after achieving the lowest validation loss before the learning is stopped	20	10	5	10	40	40
number of learnable parameters (thousands)	265	529	265	529	269	2113

Table 7.3: MIL hyperparameters

propagation. In contrast to this, when training DAC and PEM models, only one anchor is randomly sampled per input set, and when training MIL models, only 50 pairs are randomly sampled from the input set. This random sampling of anchors and pairs introduces stochasticity into the training process, which propagates into stochastic results of the experiment.

We further note that all state of the art models perform better than models proposed in this thesis.

The mean version of Permutation Equivarinat model PEM_{MEAN} performs significantly better than the max version PEM_{MAX}, which we try to explain by how max and mean operations propagate inputs to output differently. Max function in the Equivariant layer propagates information from only one element in each feature, which seems counter-productive in tasks where the context (all other elements in the set) is important.

Parameter	**Circles**	**MoG**	**Newspaper**	
PEM type	Both	Both	MAX	MEAN
Learning rate	1e−4	1e−4	3e−4	3e−4
Batch size	64	128	16	16
$d_e = d_a$ — hidden size of Equivariant layers in (4.6) and rFFs in (4.7)	256	256	256	512
$N_e = N_a = N_{eq}$ — number of Equivariant layers in (4.6) and (4.8) and rFFs in (4.7)	6	6	6	6
early stopping — number of epochs after achieving the lowest validation loss before the learning is stopped	20	20	20	20
number of learnable parameters (thousands)	4134	4134	4140	16538

Table 7.4: PEM hyperparameters

7.1.2 Effect of number of elements in input set

We anticipate that decreasing the number of elements in a set, and therefore reducing the context from which the model can draw information, will at some point lead to a decrease in the model's performance. With a lower number of elements, it becomes increasingly harder to infer the underlying structure of the cluster. In the case of a circle, at least 3 points are necessary to infer its position and radius, but more is needed to see the shape clearly and, more importantly, to distinguish various circles from each other.

We use the default dataset setting but decrease the number of elements in a set and test the models on a dataset of sets with $n_X = 20, 50, 100$ elements to confirm our hypothesis above. Achieved metrics are reported in Table 7.6 and compared using box plot in Figure 7.3. Three box plots are displayed for each model corresponding to 20, 50, and 100 elements (from left to right).

Performance of each model is significantly decreased by the drop from 50 to 20 elements in the set, less so by the drop from 100 to 50 elements. This

Model	ARI	NMI
DAC	0.9345 ± 0.0204	0.9372 ± 0.0192
$\text{ABC}_{\text{multi}}$	0.9791 ± 0.0018	0.9794 ± 0.0017
ABC_{add}	$\mathbf{0.9908 \pm 0.0011}$	$\mathbf{0.9909 \pm 0.0010}$
MIL_{MEAN}	0.8561 ± 0.0296	0.8580 ± 0.0283
MIL_{PMA}	-	-
PEM_{MAX}	0.7960 ± 0.0208	0.8186 ± 0.0184
PEM_{MEAN}	0.8817 ± 0.0089	0.8860 ± 0.0079

Table 7.5: Results on Circles dataset with constant $n_X = 100$. Number of clusters sampled randomly from range [2, 6] for each set (see Section 5.2). For MIL models and ABC_{ADD}, we report ARI and NMI averaged on 3 runs. For all other models, results were averaged on 5 runs. Results are further compared graphically in Figure 7.2.

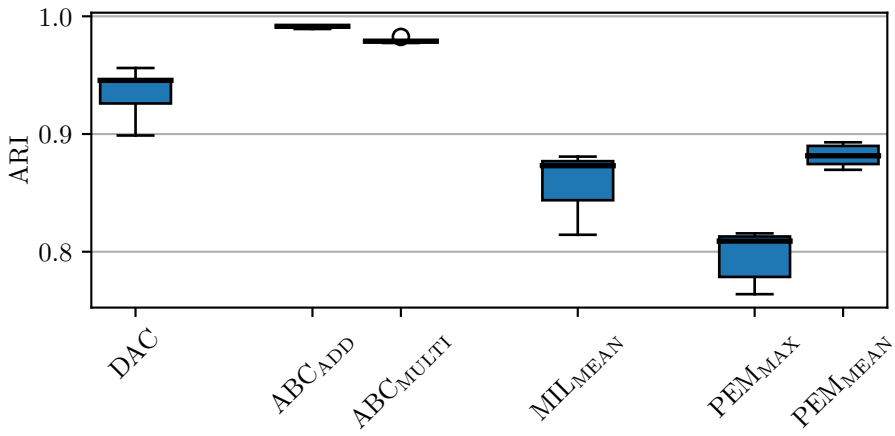

Figure 7.2: Box plot representation of results from experiment reported in Table 7.5.

is consistent with our hypothesis. In comparison to other models, DAC and $\text{ABC}_{\text{MULTI}}$ are capable of learning and inferring the cluster structure (circles) relatively well even from the dataset with only 20 elements.

7.1.3 Effect of number of clusters in set

Supervised clustering problem with a same number of clusters k_X in each set naturally seems to be easier to solve. We test this on dataset with the default setting, but with number of clusters set constant $k_X = 4$. Results are reported in Table 7.7 and compared using box plots in Figure 7.4.

All models but DAC achieve greater results on a dataset with a fixed number of clusters. The DAC model actually benefits from the variable number of clusters. This is caused by the iterative process it uses to infer the clusters of the input set. During training, one element is randomly sampled

Model	20 elements		50 elements		100 elements	
	ARI	NMI	ARI	NMI	ARI	NMI
DAC	**0.6397**	**0.7688**	0.8942	0.9085	0.9345	0.9372
ABC_{multi}	0.6333	0.6963	0.9291	0.9412	0.9791	0.9794
ABC_{add}	0.1378	0.1826	**0.9568**	**0.9646**	**0.9908**	**0.9909**
MIL_{MEAN}	0.2968	0.3333	0.6397	0.6611	0.8561	0.8580
MIL_{PMA}	-	-	-	-	-	-
PEM_{MAX}	0.4361	0.4869	0.7374	0.7799	0.7960	0.8186
PEM_{MEAN}	0.2656	0.2962	0.7983	0.8246	0.8817	0.8860

Table 7.6: Results on Circles dataset with number of clusters sampled randomly from the range [2, 6] for each set. We report ARI and NMI averaged on 3 runs for both MIL and ABC_{ADD} models and on 5 runs for the rest of the models. Results are further compared using box-plots in Figure 7.3.

Model	4 clusters		2-6 clusters	
	ARI	NMI	ARI	NMI
DAC	0.8460	0.8460	0.9345	0.9372
ABC_{multi}	0.9893	0.9875	0.9791	0.9794
ABC_{add}	**0.9952**	**0.9944**	**0.9908**	**0.9909**
MIL_{MEAN}	0.9098	0.9068	0.8561	0.8580
MIL_{PMA}	-	-	-	-
PEM_{MAX}	0.8301	0.8407	0.7960	0.8186
PEM_{MEAN}	0.9206	0.9155	0.8817	0.8860

Table 7.7: Results on Circles dataset with 100 elements. We report ARI and NMI averaged on 5 runs (results for MIL are averaged on 3 runs due to time consumption). Box-plot comparison is in Figure 7.4.

as an anchor and processed with the input set by the model. For this input, the DAC model predicts which elements of the input set belong to the same cluster as the anchor. After this, the model is presented with a new set. This means that during training on a dataset with constant $k_X = 4$, the DAC model only encounters sets with 4 clusters. But during the iterative cluster infer process, elements of the set are iteratively held out if assigned to some cluster, and the rest of the set is processed by the model again. This creates a situation where the DAC model must process a set with less than 4 clusters, even though it was never trained on such sets during the training phase. By presenting it with sets with a variable number of clusters, we also train the model on sets it will encounter during the cluster inferring. This fact is used for the DAC model trained on the Newspaper dataset as data augmentation. If a set of Newspaper dataset is presented to the DAC model during training, we discard a random number of clusters to create a set with a lower number of clusters, solving the problem that occurred when training DAC on the Circles dataset with $k_X = 4$.

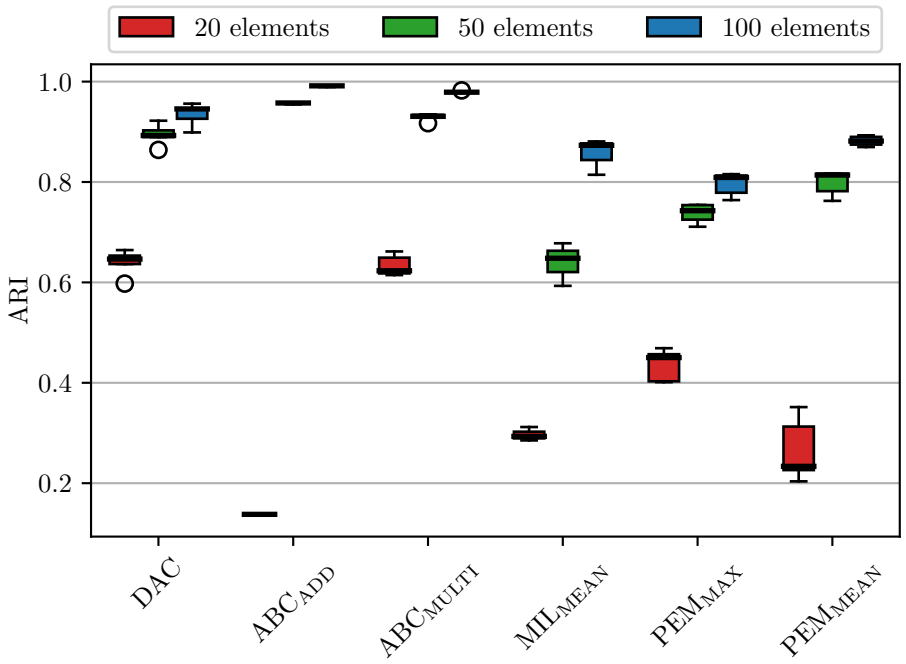

Figure 7.3: Comparison of results from experiment reported in Table 7.6.

7.2 Mixture Of Gaussians

MoG dataset is arguably an easier supervised clustering problem than the Circles dataset, and for this reason, we chose it to test the scalability of the examined models. In the following experiments with MoG, we again randomly sample the number of clusters k_X from the range [2, 6]. For each experiment, we fix the number of elements n_X in the input set to some value and measure the time to train the model and the time to cluster the test dataset, which consists of 1k sets of n_X elements.

In Figure 7.5, we compare achieved ARI on test dataset plotted against time required to train the model. Models are divided with colour. The same plot, additionally with measurements of the MIL_{PMA} model, is displayed in Figure A.1 in Appendix A. Train time of MIL_{PMA} is very large and when plotted, it obfuscates other results in the plot. Results from training the model on various datasets with different n_X are displayed. The plus sign marks the position of mean of both measured values (time and ARI). The number next to the two-dimensional mean denotes the number of elements n_X used for that particular experiment. In Figure A.2 in Appendix A, we also display these results with median values and two-dimensional version of box plots. We moved the two-dimensional box plot Figure A.2 into the appendix because the results are dispersed and hard to follow. Cleaner results might be achieved by running each experiment for several more times but that was

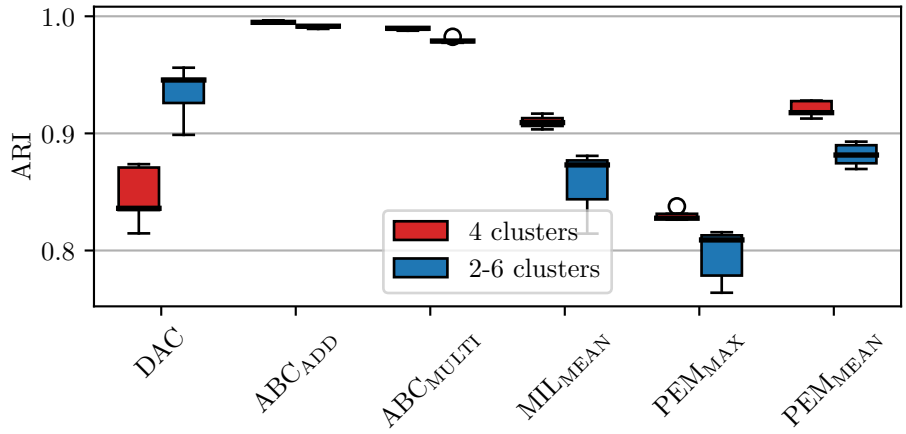

Figure 7.4: Box plot comparison of results from experiment reported in Table 7.7.

not possible due to limited computational resources. The two-dimensional box plot representation is described in detail in the appendix.

We see that overall, the DAC model performs best based on both the time and the clustering results. Performance of $\text{ABC}_{\text{MULTI}}$ model given by ARI is also high compared to other models; on the dataset with 100 elements, it even surpasses DAC. Since the box plots (Figure A.2) of DAC and $\text{ABC}_{\text{MULTI}}$ overlap, their relative precedence in performance could be questioned.

Because of reasons described in Section 4.1, the MIL model's time performance is significantly worse compared to all other models. Training time of MIL_{PMA} (see Figure A.1) is even higher than of MIL_{MEAN} due to the attention mechanism used in pooling.

On some occasions, the model's performance based on ARI decreases when the number of elements was raised. This contradicts with our finding from Subsection 7.1.2. Clusters in the MoG dataset, however, do not have an intra-cluster structure, and our argument that a larger number of elements helps the model to detect such structure fails here. It might also be caused by the low number of measurements (as seen for example on the results of DAC model in Figure A.2).

Looking at the scale of the ARI axis, the performance of the PEM_{MEAN} modle is only slightly worse than of $\text{ABC}_{\text{MULTI}}$, while its training time is significantly lower for the same set size.

Time required to infer the clusters is compared to the achieved ARI in Figure 7.6. The approximate order of models based on measurements in Figure 7.5 did not change in Figure 7.6, but the difference between the DAC and other models became much more significant. This is caused by the low time complexity ($\mathcal{O}(k_X n_X)$) of DAC's clustering process. All other models use spectral clustering, whose time complexity is $\mathcal{O}(n_X^3)$, which inhibits the use of these models for problems with a very large number of elements.

A two-dimensional box plot version of Figure 7.6 is displayed in Appendix

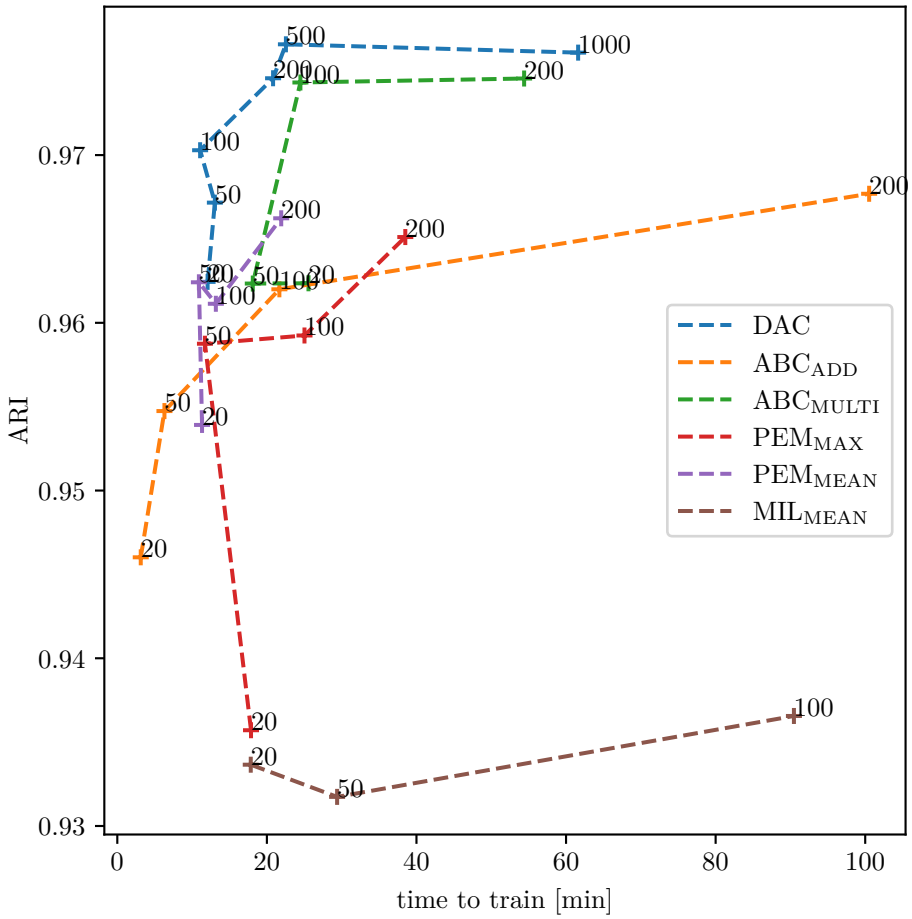

Figure 7.5: ARI achieved on test dataset vs time to train the model on the MoG dataset, with the number of clusters sampled randomly from the range [2, 6] for each set. Numbers of elements n_X in each set for corresponding measurements are displayed next to the plus signs. Mean values are displayed.

A as Figure A.3.

7.3 Newspaper

We present performance of examined models on the real-world Newspaper dataset in Table 7.8, box plot comparison of these results can be seen in Figure 7.7. Contrary to the Circles and MoG artificial datasets, where DAC and ABC models dominated in performance, the best model to cluster the Newspaper dataset turned out to be MIL (both versions). A possible reason for this is that MIL (as a model with simpler structure — see Table 7.3 for number of trainable parameters) is able to generalize from small Newspaper dataset, whereas other used models fail to do so well. When experimenting

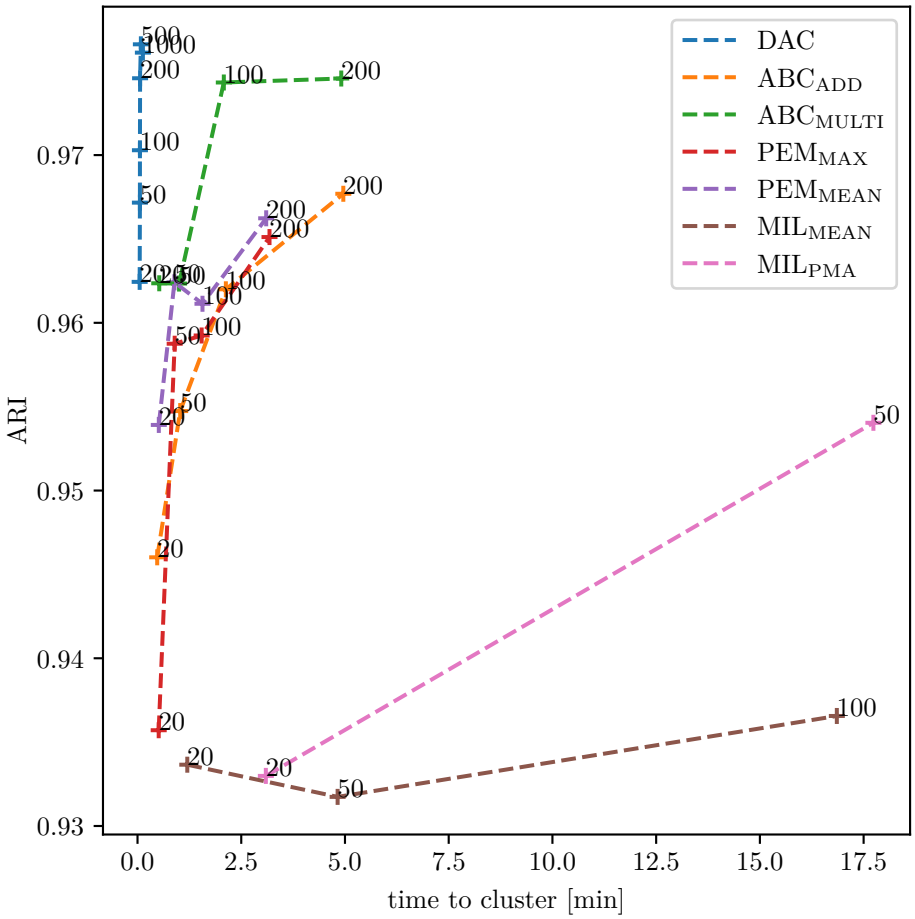

Figure 7.6: ARI achieved on test dataset vs time to cluster 1000 sets of the MoG dataset. Numbers of elements n_X in each set for corresponding measurements are displayed next to the plus signs. Mean values are displayed.

with artificial data, much larger datasets were generated, and because of that, the problem with generalization from small dataset would not occur.

The MIL$_{\text{PMA}}$ model using attention for pooling scored slightly better than MIL$_{\text{MEAN}}$. The difference is, however, not significant enough (the quartiles in box plot overlap) to rule out the MIL$_{\text{PMA}}$ as a better model. We will need to investigate further how attention versus mean pooling affects the performance of the MIL model.

7.4 Discussion of results

Throughout the experiments on synthetic data, the state of the art models showed greater performance than proposed methods. This shows that a more complex model architecture using self-attention is propably needed to solve such problems. On the real-world Newspaper dataset, we achieved best results

Model	ARI	NMI
DAC	0.7989 ± 0.0554	0.8607 ± 0.0371
ABC_{ADD}	0.8581 ± 0.0076	0.9047 ± 0.0116
$\text{ABC}_{\text{MULTI}}$	0.7779 ± 0.1289	0.8435 ± 0.1061
MIL_{MEAN}	0.8692 ± 0.0531	0.9152 ± 0.0398
MIL_{PMA}	$\mathbf{0.9094 \pm 0.0125}$	$\mathbf{0.9420 \pm 0.0076}$
PEM_{MAX}	0.5908 ± 0.0497	0.6937 ± 0.0389
PEM_{MEAN}	0.5821 ± 0.0406	0.6798 ± 0.0331

Table 7.8: Models comparison on Newspaper dataset. Box-plot representation of these results can be seen in Figure 7.7. We report ARI and NMI averaged on 5 runs.

using our proposed model MIL_{PMA}.

We showed that the DAC model is the only feasible model for problems with a large number of elements in set due to time consumption of training and clustering, but on datasets with smaller sets, MIL (on the Newspaper dataset) and ABC (on the synthetic datasets) models showed better performance.

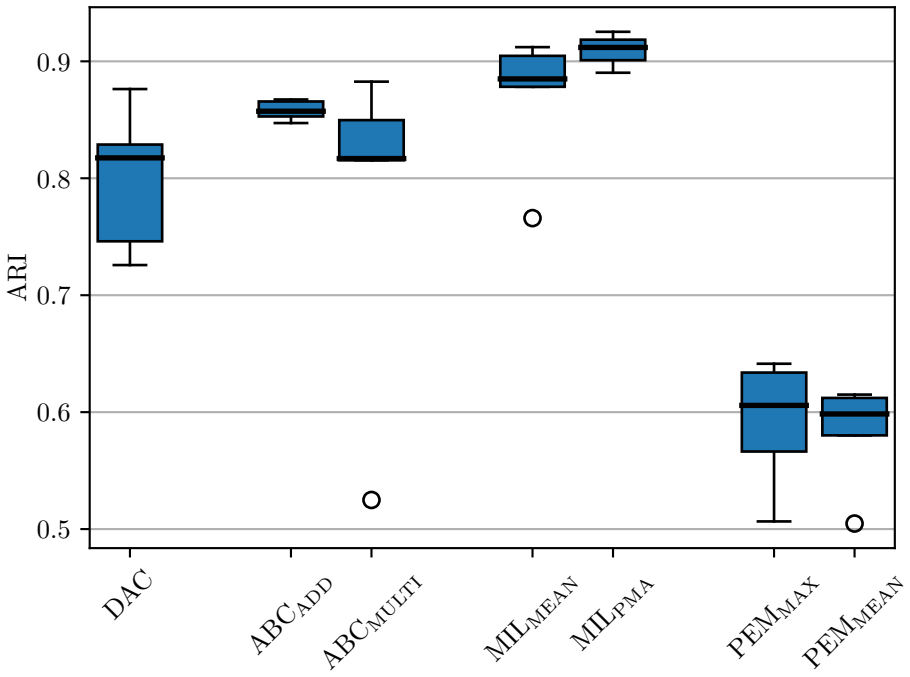

Figure 7.7: Graphical comparison of results from Table 7.8.

Chapter 8
Conclusion

In this thesis, we focused on the supervised clustering problem and comparison of models that use different approaches to represent the input and output to solve the problem using neural networks. We explored current literature and chose two state of the art architectures, DAC and ABC, both relying on the self-attention mechanism as a fundamental building block. Another two models, MIL and PEM, were proposed to explore the clustering ability of simpler architectures. Basic ideas and methods behind the examined models were organized and described in Chapter 2, followed by detailed description of the models in Chapters 3 and 4.

The implementation of DAC and ABC was modified to be compatible with all used datasets. Some further modifications were needed to solve bugs and to optimize the computation of attention in the ABC model. Together with our implementation of the proposed methods, all models form a unified software project that enables us to run experiments easily, store the results and quickly integrate new datasets to test the models on.

In the experimentation phase, we compared the models on one real-world and two synthetic datasets. This included searching for optimal parameters for each model on each dataset and running a large number of experiments. First, we focused on the intra-cluster dependencies of the Circles dataset. Next, the scalability of models to growing input was explored using the MoG dataset. As a real-world example, the Newspaper dataset was used.

8.1 Future work

All models but DAC use spectral clustering (another unsupervised clustering method could be used), which also forces us to use some method for estimating the number of clusters. In future work, we want to focus more on methods that are able to infer the clusters directly and are trained end-to-end such as DAC. As shown in experiments on the MoG dataset, the independence of a model on an unsupervised clustering method greatly increases the scalability to growing input size.

We also plan to use supervised clustering models to attempt to solve some of the problems discussed in Section 1.1, such as fast and cheap data annotation, text summarization and topic detection.

Bibliography

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[2] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/.

[3] Samuel Coward, Erik Visse-Martindale, and Chithrupa Ramesh. "Attention-Based Clustering: Learning a Kernel from Context". In: *arXiv preprint arXiv:2010.01040* (2020).

[4] Lawrence Hubert and Phipps Arabie. "Comparing partitions". In: *Journal of Classification* 2.1 (Dec. 1985), pp. 193–218. DOI: 10.1007/bf01908075. URL: https://doi.org/10.1007/bf01908075.

[5] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[6] Juho Lee, Yoonho Lee, and Yee Whye Teh. "Deep Amortized Clustering". In: *CoRR* abs/1909.13433 (2019). arXiv: 1909.13433. URL: http://arxiv.org/abs/1909.13433.

[7] Juho Lee et al. "Set transformer: A framework for attention-based permutation-invariant neural networks". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 3744–3753.

[8] Andrew Ng, Michael Jordan, and Yair Weiss. "On spectral clustering: Analysis and an algorithm". In: *Advances in neural information processing systems* 14 (2001), pp. 849–856.

[9] Tomáš Pevný and Petr Somol. "Using neural network formalism to solve multiple-instance problems". In: *International Symposium on Neural Networks*. Springer. 2017, pp. 135–142.

[10] Ashish Vaswani et al. "Attention is all you need". In: *arXiv preprint arXiv:1706.03762* (2017).

[11] Ulrike Von Luxburg. "A tutorial on spectral clustering". In: *Statistics and computing* 17.4 (2007), pp. 395–416.

[12] Manzil Zaheer et al. "Deep Sets". In: *CoRR* abs/1703.06114 (2017). arXiv: 1703.06114. URL: http://arxiv.org/abs/1703.06114.

Appendix A

Mixture of Gaussian experiments - additional plots

Additional figures for Section 7.2 are displayed here.

In Figure A.2 and A.3, results are displayed using a "two-dimension box plot". The plus sign marks position of median of both measured values (time and ARI) during the particular experiment. Rectangle around median displays the Q1 and Q3 (described in the introduction of Chapter 7) positions for values displayed on y axis (ARI) and x axis (time) analogously to normal box plot (see Figure 7.1). Outliers are not displayed. Number next to the plus sign showing the median value denotes number of elements n_X used for that particular experiment. Models are divided with colour.

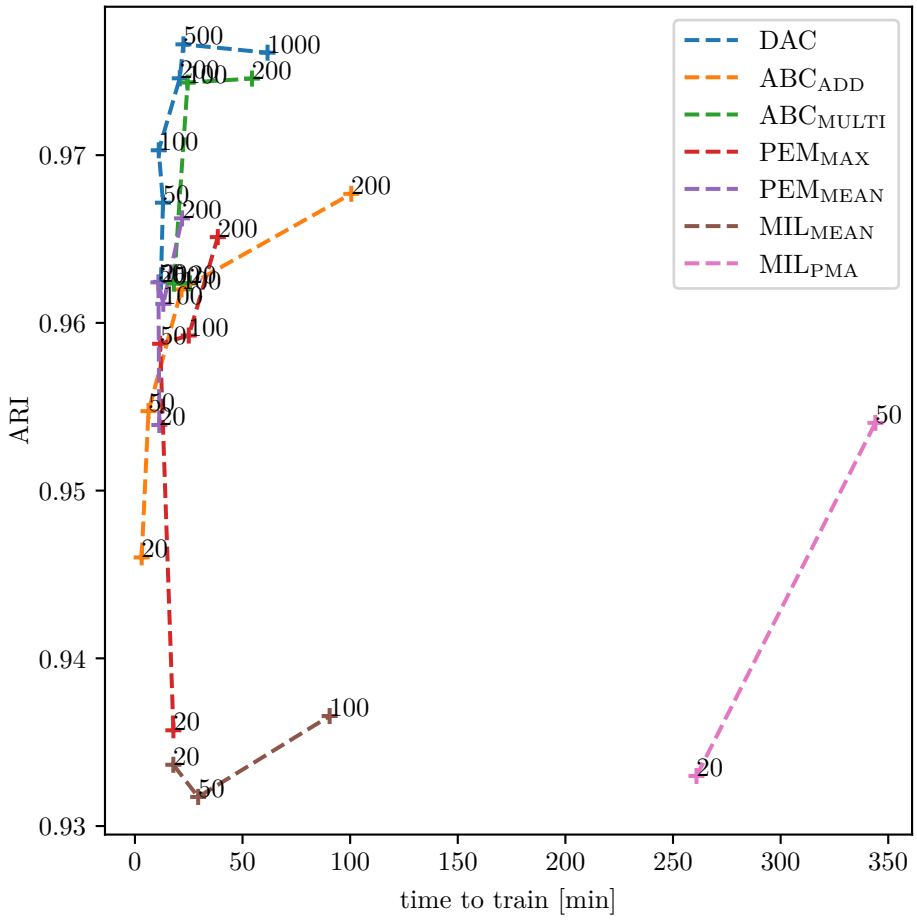

Figure A.1: ARI achieved on test dataset vs time to train the model on the MoG dataset, with the number of clusters sampled randomly from the range [2, 6] for each set. Numbers of elements n_X in each set for corresponding measurements are displayed next to the plus signs. Mean values are displayed.

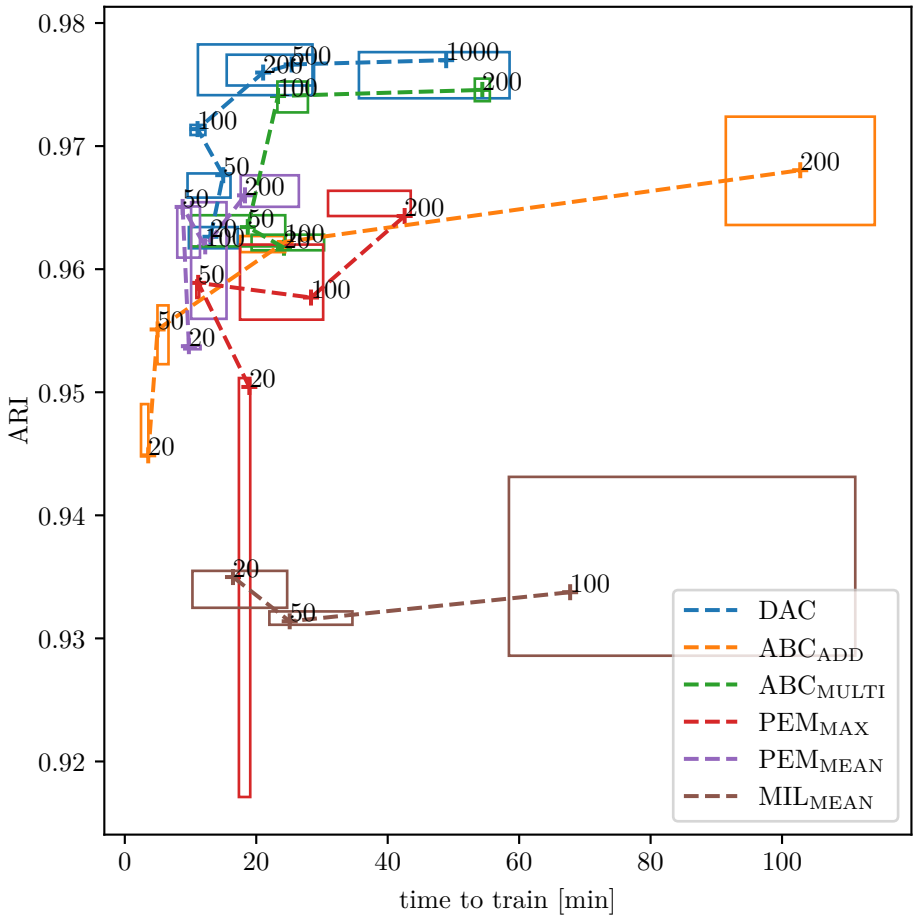

Figure A.2: ARI achieved on test dataset vs time to train the model on the MoG dataset, with the number of clusters sampled randomly from the range [2, 6] for each set. Numbers of elements n_X in each set for corresponding measurements are displayed next to the plus signs. Median values with two-dimensional box plots are displayed.

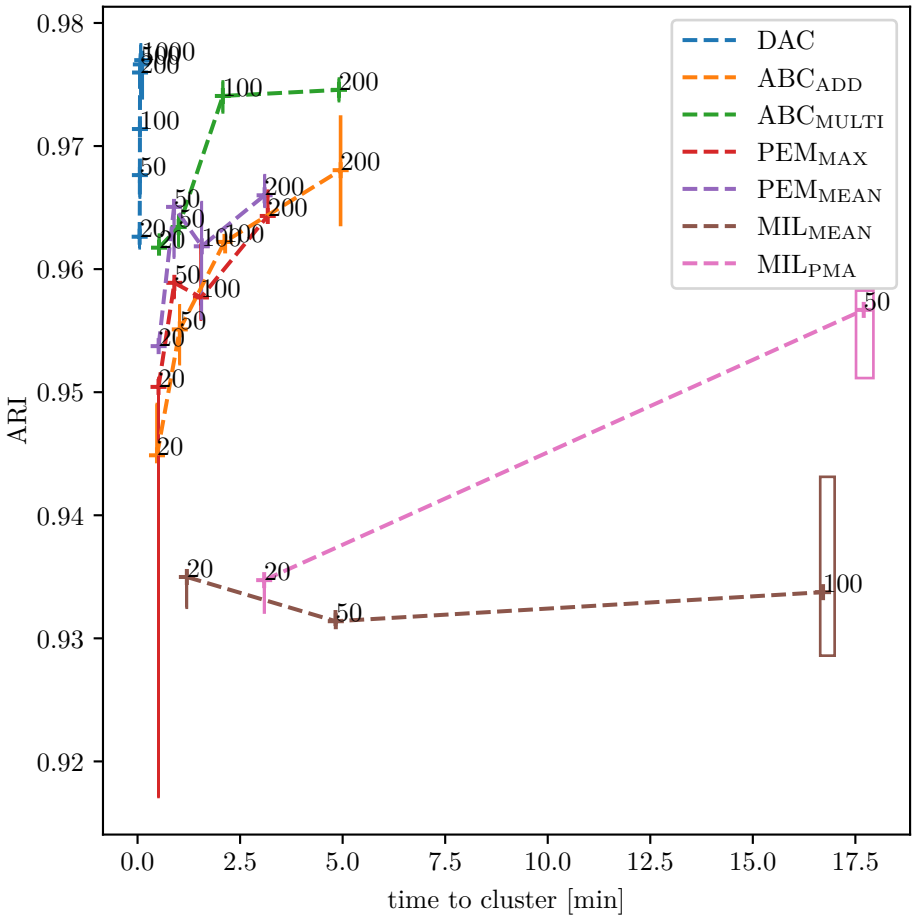

Figure A.3: ARI achieved on test dataset vs time to cluster 1000 sets of the MoG dataset. Numbers of elements n_X in each set for corresponding measurements are displayed next to the plus signs. Median values with two-dimensional box plots are displayed.