UNIVERSITY OF
CAMBRIDGE

Neural Diagrammatic Reasoning

Duo Wang

Churchill College

This dissertation is submitted on August, 2020 for the degree of Doctor of Philosophy

Declaration

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. It is not substantially the same as any that I have submitted, or am concurrently submitting, for a degree or diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. I further state that no substantial part of my dissertation has already been submitted, or is being concurrently submitted, for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. This dissertation does not exceed the prescribed limit of 60 000 words.

Duo Wang
August, 2020

Neural Diagrammatic Reasoning

Duo Wang

Abstract

Diagrams have been shown to be effective tools for humans to represent and reason about complex concepts. They have been widely used to represent concepts in science teaching, to communicate workflow in industries and to measure human fluid intelligence. Mechanised reasoning systems typically encode diagrams into symbolic representations that can be easily processed with rule-based expert systems. This relies on human experts to define the framework of diagram-to-symbol mapping and the set of rules to reason with the symbols. This means the reasoning systems cannot be easily adapted to other diagrams without a new set of human-defined representation mapping and reasoning rules. Moreover such systems are not able to cope with diagram inputs as raw and possibly noisy images. The need for human input and the lack of robustness to noise significantly limit the applications of mechanised diagrammatic reasoning systems.

A key research question then arises: *can we develop human-like reasoning systems that learn to reason robustly without predefined reasoning rules?* To answer this question, I propose **Neural Diagrammatic Reasoning**, a new family of diagrammatic reasoning systems which does not have the drawbacks of mechanised reasoning systems. The new systems are based on deep neural networks, a recently popular machine learning method that achieved human-level performance on a range of perception tasks such as object detection, speech recognition and natural language processing. The proposed systems are able to learn both diagram to symbol mapping and implicit reasoning rules only from data, with no prior human input about symbols and rules in the reasoning tasks. Specifically I developed EulerNet, a novel neural network model that solves Euler diagram syllogism tasks with 99.5% accuracy. Experiments show that EulerNet learns useful representations of the diagrams and tasks, and is robust to noise and deformation in the input data. I also developed MXGNet, a novel multiplex graph neural architecture that solves Raven Progressive Matrices (RPM) tasks. MXGNet achieves state-of-the-art accuracies on two popular RPM datasets. In addition, I developed Discrete-AIR, an unsupervised learning architecture that learns semi-symbolic representations of diagrams without any labels. Lastly I designed a novel inductive bias module that can be readily used in today's deep neural networks to improve their generalisation capability on relational reasoning tasks.

Acknowledgements

This dissertation is never possible with the support from many people that I would like to acknowledge and thank. First of all I would like to acknowledge my two supervisors, **Pietro Lio** and **Mateja Jamnik**. They have provided me with ideas and inspirations for my research, and gave me the mental support much needed to get through the PhD journey. Next I would like to thank **Cecilia Mascolo** and **Sean Holden** for advising me for the first and second year of PhD. I would also like to thank many collaborators who provide me with inspirations and directions. They are (not ordered) **Zhongzhao Teng, Yiren Zhao, Petar Velickovic, Jin Zhu, Filippo Spiga, Rui Zhang, Junwei Yang, Rolleen Colleens, Robert Mullins, John Suckling**. I would also like to acknowledge **Christopher Summerfield, Zoe Kourtzi, Guy Williams** and **everyone from the AI group** in the Department of Computer Science and Technology, University of Cambridge, for providing suggestions and ideas on my research. Finally I would like my friends and family who provided me the support much needed for this journey. I would especially like to thank my parents **Xiaohong Wang** and **Hongwei Gao**, for raising me to who I am today, and **Tao Dou**, for the endless care and love.

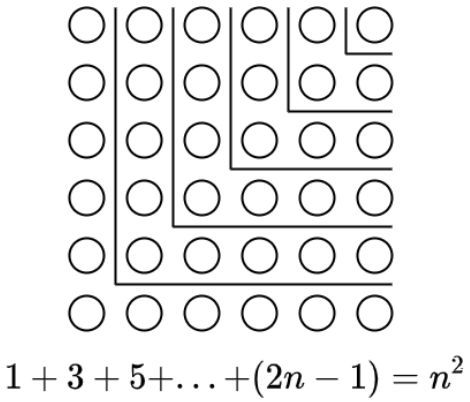
Contents

Chapter 1

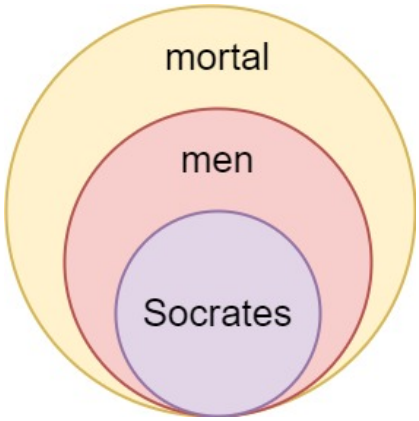
Introduction

Diagrams have been used since ancient times to represent, convey, and reason with abstract concepts, ranging from proving odd natural sums (Figure 1.1a), representing ontological relationships (Figure 1.1b) and illustrating pulley system designs (Figure 1.1c). A well-known proverb says that "a diagram is worth ten thousand words". Indeed, the effectiveness of diagrams in representing and conveying abstract concepts has been theoretically argued [63] and experimentally shown [87]. The human brain is naturally designed to efficiently capture information presented in visual forms. In fact, a type of diagrammatic reasoning, "Raven Progressive Matrices" or RPM [79], is one of the most popular approaches to measuring human fluid intelligence. Solving diagrammatic reasoning tasks like RPM requires the same type of intelligence needed for numerous daily and industrial scenarios, such as assembling IKEA furniture from pieces; determining potential actions of vehicles in front of yours based on traffic lights and road signs and turning signals; and planning a software project with workflow diagrams.

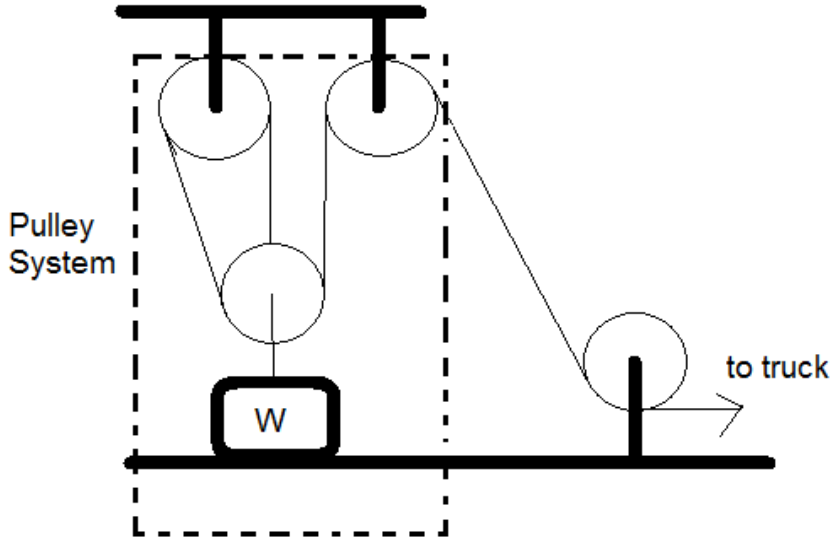
For an AI system to achieve Artificial General Intelligence [28], it must possess the same form of intelligence required to solve a range of diagrammatic reasoning tasks such as RPM. Reasoning is defined as the process of applying logic to infer conclusions from premises (deductive), infer rules from observations (inductive), and infer hidden variables given rules and conclusions (abductive). Many approaches have been proposed for different types of diagrams. For example, Jamnik et al. [51] developed DIAMOND for automating diagrammatic proofs of arithmetic theorems. Barwise et al. [6] used blocks-world to teach and reason in first order logic with Hyperproof. Kortenkamp et al. [61] developed Cinderella for geometric theorem proving. Stapleton et al. [93] developed Edith for automated Euler diagram theorem proving. Urbas et al. [99] extended Edith to spider diagrams and developed Speedith. However, most of the proposed approaches are mechanised reasoning systems. In these systems, mechanising reasoning with diagrams usually relies on methods of encoding diagrams as symbolic representations that can be easily processed with a



(a) Odd natural sum



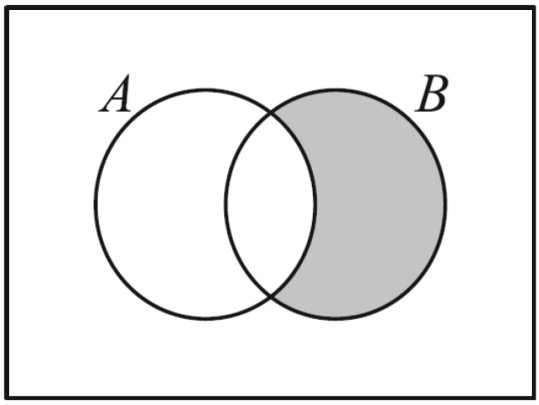
(b) Euler diagram syllogism



(c) Pulley systems

Figure 1.1: Examples of using diagrams to more effectively convey abstract concepts. (a) shows the diagrammatic proof of the odd naturals sum theorem. (b) shows an Euler diagram (a type of diagram that represents sets and their relationships, properly introduced in Section 2.2.1)that represent the syllogism "All men is mortal, Socrates is a man, therefore Socrates is mortal". (c) shows a pulley system diagram typically used to represent a pulley system designed for weight lifting.

rule-based program. Figure 1.2 shows an example of Venn diagrams represented as a set of symbols used in rule-based system processing. Such methods rely on human experts to define the framework of diagram-to-symbol mapping and the set of rules to reason with the symbols. This means these systems can hardly generalise beyond the defined input domain, as rules in the new domain must be manually defined by human experts. Moreover, these systems are not able to cope with noisy input, unless the noise model is already known and integrated into the diagram-to-symbol mapping. The need for human labour when adapting to new diagrams, the inability to generalise beyond defined input domains, and the lack of robustness to noise limit the applications of such systems.



- $L(d) = \{A, B\}$
- $Z(d) = \{(\emptyset, \{A, B\}), (\{A\}, \{B\}), (\{B\}, \{A\}), (\{A, B\}, \emptyset)\}$
- $Z^*(d) = \{(\{B\}, \{A\})\}$

Figure 1.2: Example of encoding a Venn diagram into symbols. For most systems, the user inputs the symbols (left) and the diagrams (right) are generated from the symbols. Here, $L(d)$ is the set of contour labels, $Z(d)$ is the set of zones (S_1, S_2), that are inside of contour S_1 and outside of contour S_2. $Z^*(d)$ is the set of shaded zones.

In this dissertation, I propose **Neural Diagrammatic Reasoning**, a new family of neural diagrammatic reasoning systems that does not have the drawbacks of existing mechanised reasoning systems. Neural Diagrammatic Reasoning systems are based on deep neural networks (DNNs), a recently popular machine learning method that achieves human-level performance on object detection, speech recognition, and natural language processing. DNNs have stacked layers of artificial neurons that can learn an arbitrary function f, which maps data x to target y as $y = f(x)$. A detailed review of DNNs is given in Section 2.3. By learning from data, DNNs require few pre-defined rules from human experts. DNNs generalise well on identically and independently distributed (i.i.d) data [122] and are robust to noise [83]. Given these advantages, a DNN is a good candidate for learning diagrammatic reasoning. However, DNNs have mostly been applied on perception tasks such as object recognition, with limited applications on diagrammatic reasoning tasks (except for several concurrent works on visual reasoning, which will be discussed in Section 1.1). This thesis discusses how DNNs can be used both for learning diagram-to-representation mappings, and reasoning mechanisms with the learned representations. I propose EulerNet, which learns to solve Euler Diagram Syllogisms, and MXGNet, which learns to solve Raven Progressive Matrices Reasoning (details in Section 2.2). To reduce the need for labelled training data, I also develop models that learn to extract symbolic representations from diagrams in an unsupervised manner, meaning no data labelling is required. Lastly, as EulerNet and MXGNet exhibit poor out-of-distribution generalisation performance, I develop models that can generalise beyond their training distributions, allowing reasoning systems to be applied to unseen data domains. Next, I briefly review a few related works that are concurrent with the works in this dissertation. Lastly, I briefly summarise each of these contributions, which are discussed in detail in subsequent

chapters. Figure 1.3 shows a concept map of the four chapters in this dissertation. In this dissertation I aim to keep the discussion focused and concise, and therefore leave some technical details (e.g., hyper-parameters, training hardware and software environment) to the Appendices.

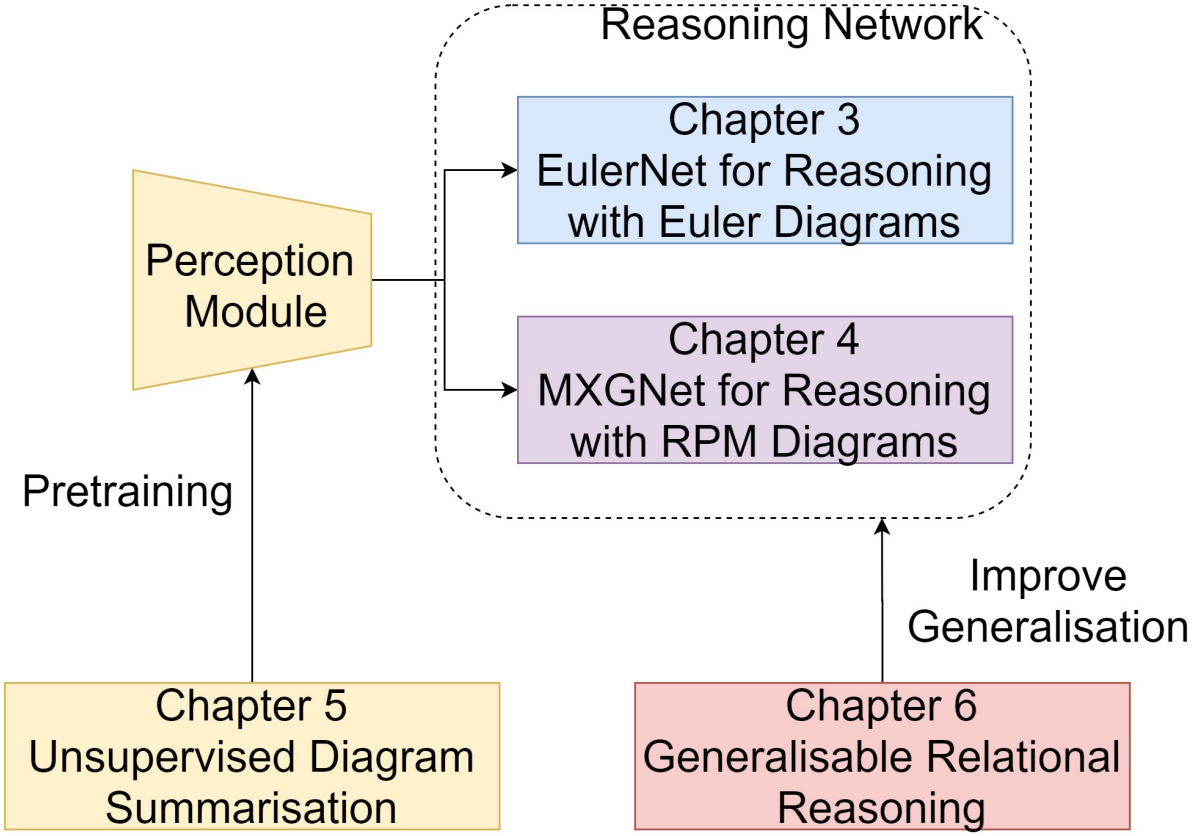


Figure 1.3: A concept map of the chapters in this dissertation. Chapter 3 and Chapter 4 develop new reasoning network architectures tackling different types of diagrammatic reasoning problems. Both architectures have similar front-end perception modules for perceiving the diagrams. Chapter 5 develops an unsupervised learning method for diagrammatic summarisation, which can be used for pre-training the perception modules. Chapter 6 develops a novel inductive bias that improves the generalisation of the reasoning networks.

1.1 Related works

In this section I discuss a few exemplary related lines of works that push the boundary for DNN-based reasoning systems. These works are mostly concurrent with the work discussed in this dissertation.

Visual Question Answering: The most popular visual reasoning task for machine learning is the Visual Question Answering (VQA) task. This task asks a question about an image, such as "what is the colour of the fruit in the person's hand?". While many different

VQA tasks are proposed (e.g., [81, 2, 54, 46]), most of them contain more factual questions such as "what is the colour of . . . ?" and "where is the object . . . ?". Among these tasks, CLEVR [54], a synthetic VQA task that focuses specifically on reasoning, is most relevant to this dissertation. CLEVR seeks to reduce the implicit dataset bias often associated with factual questions by only asking questions that focus on visual-spatial reasoning. Since the invention of CLEVR, many methods have been developed to tackle this task. Among them, Relation Networks [85] explicitly compute pairwise relations between grid feature locations to better suit the relational comparisons involved in the reasoning. Neural Module Networks [44] train a DNN to translate the question asked into a sequence of either pre-defined or trainable modules that process the image to arrive at the answer. This idea is picked up by Symbolic-VQA [114], which adds object-level representations and more symbolic program execution modules to achieve improved performance. While the object representation module in Symbolic-VQA is pre-trained on an object detection task, the Neural Symbolic Concept Learner [72] jointly learns the object representation and symbolic program generation in a curriculum learning setting.

Raven Progressive Matrices: Besides the Euler diagram syllogism task proposed in this thesis, Raven Progressive Matrices tasks, such as PGM [5] and RAVEN [120], are the only diagrammatic reasoning tasks developed for DNN training. Raven Progressive Matrices are introduced in details in Section 2.2.2. Several methods have been proposed to solve RPM tasks. Barrett et al. [5] proposed the Wild Relation Network, which is an adaptation of the Relation Network [85] for RPM tasks. Jahrens et al. [50] further extend Relation Network to Multi-Layer Relation Networks by introducing hierarchical relation learning. Zhang et al. [120] adapt the perceptual contrast theory from psychology to develop CoPI-Net. Van et al. [101] show that encouraging disentangled representations improves model performance on RPM tasks.

Learning Physics Interaction: There is another line of work not explicitly branded as "visual reasoning", but which implicitly learn to reason with relations expressed as physics laws. For example, Kipf et al. [58] develop NRI, a graph autoencoder that learns simple physical forces (such as spring and gravity force) between particles. Kipf et al [59] later adapt this idea to model more general relations between object representations in a given environment. Janner et al [53] developed O2P2, which learns physical interactions between objects in videos.

1.2 Main Contributions

1.2.1 Investigating Euler Diagram Syllogisms with Deep Neural Networks

In Chapter 3 and [106], I introduce EulerNet, a DNN-based diagrammatic reasoning system that solves Euler diagram syllogism questions, such as the example in Figure 1.1b. EulerNet learns to solve syllogisms represented as Euler diagrams. It takes two Euler diagrams representing the premises in a syllogism as input, and outputs either a categorical (subset, intersection, or disjoint) or diagrammatic conclusion (generating an Euler diagram representing the conclusion) to the syllogism. EulerNet can achieve 99.5% accuracy in generating syllogism conclusions. I analyse the learned representations of the diagrams, and show that meaningful information can be extracted from such neural representations. Furthermore, I show that EulerNet is robust to noise and deformation of the Euler diagrams, with just a 0.4% decrease in accuracy. EulerNet is the first Euler diagram reasoning system that learns diagram-to-representation mapping (instead of needing manually defined mappings) and implicit reasoning rules (instead of needing manually-defined reasoning rules) without requiring any human-defined symbols and rules.

1.2.2 Abstract Diagrammatic Reasoning with Multiplex Graph Networks

EulerNet, while effective, cannot scale to an arbitrary number of contours in the diagrams due to the fixed-length vector representations produced. To remedy this, in Chapter 4 and [109], I introduce MXGNet, a Multi-Layer Graph Network (Graph Neural Networks are introduced in detail in Section 2.6) that tackle Raven Progressive Matrices (RPM) reasoning tasks (introduced in Section 2.2.2). MXGNet combines three powerful concepts, namely, object-level representation, graph neural networks and multiplex graphs, to solve visual reasoning tasks. MXGNet first extracts object-level representations for each element in all diagrams in an RPM task, then forms a multi-layer multiplex graph capturing multiple relations between objects across different diagram panels. MXGNet summarises the multiple graphs extracted from the diagrams in the RPM task, and uses the summarised information to pick the most probable answer from the given candidates. MXGNet is tested on two comprehensive datasets for RPM reasoning, namely PGM and RAVEN. On both datasets, MXGNet outperforms the state-of-the-art models by a considerable margin, achieving 89.6% accuracy on the PGM dataset and 83.91% accuracy on the RAVEN dataset (correct at the time of paper submission). MXGNet shows that object-level representation with graph-based relational learning is a more suitable approach for learning and reasoning with diagrams of multiple elements and parallel relations.

1.2.3 Unsupervised Diagram Summarisation with Deep Generative Models

Supervised Training of EulerNet and MXGNet yields good results, but requires a large amount of labelled data. In Chapter 5 and [108, 107], I introduce unsupervised learning methods for extracting symbolic representations from diagrams. I developed Discrete Attend-Infer-Repeat (Discrete-AIR), a Recurrent Auto-Encoder with structured latent distributions containing discrete categorical distributions, continuous attribute distributions, and factorised spatial attention. While inspired by the original Attend-Infer-Repeat model [21] (details in Section 5.1) and retaining AIR model's capability in identifying objects in an image, Discrete-AIR provides direct interpretability of the latent codes. It is shown that for efficient inference in the case of Multi-MNIST [21] and a multiple-objects version of the the dSprites [74] dataset, the Discrete-AIR model, in contrast to other Variational Auto-Encoders with long latent vectors, needs just one categorical latent variable and one attribute variable (for Multi-MNIST only), together with spatial attention variables. I perform an analysis to show that the learned categorical distributions achieve 87.0% and 94.5% category correspondence rates for Multi-MNIST and for Multi-Sprites. The symbolic representations extracted by Discrete-AIR are directly interpretable, and can be fed into traditional mechanised reasoning systems for rule-based reasoning. Thus, it provides a potential method for amalgamating neural perception modules with mechanised reasoning modules.

As Discrete-AIR extracts semi-symbolic representations for categories and attributes of an entity, it can be readily used to build a network of relations between entities. Relations between entities can be encoded as edges of a graph, while entities are encoded as nodes of a graph. The edge encoding can be initialised in an interpretable way based on the symbolically represented nodes. In this way, Discrete-AIR provides a potential way of applying standard graph neural network methods on the extracted graph to solve the reasoning tasks.

1.2.4 Generalisable Neural Network for Relational Reasoning

MXGNet, while achieving state-of-the-art (SOTA) results, has poor out-of-distribution generalisation performances. To remedy this, in Chapter 6 and [110], I discuss the lack of out-of-distribution (**o.o.d**) generalisation capability of the two models described in [106, 109] and concurrent models on visual reasoning, such as [5, 50]. I develop a neuroscience-inspired inductive-biased module that can be readily amalgamated with current neural network architectures to improve **o.o.d** generalisation performance on relational reasoning tasks. This module learns to project high-dimensional object representations to low-

dimensional manifolds for more efficient and generalisable relational comparisons. It is shown that neural networks with this inductive bias achieve considerably better o.o.d generalisation performance for a range of relational reasoning tasks. For the "maximum of a set" task (finding the maximum values in a set of real valued numbers), the proposed model reduces Mean Square Errors of previous SOTA models by approximately 300-fold. For the "visual object comparison" task (comparing two object's attributes such as size and colour), the proposed model achieves 13.96 to 14.53% increased accuracy over baseline models. For the "Raven Progressive Matrices" task, the proposed model achieved 7.0% increased generalisation accuracy over previous SOTA models. Finally, I analyse the proposed inductive bias module to understand the importance of lower dimension projection, and propose an augmentation to the algorithmic alignment theory to better measure algorithmic alignment with generalisation. The proposed low-dimensional comparators are shown to be effective in improving **o.o.d** generalisation for a range of relational reasoning architectures, and can be readily adapted for any other relational reasoning tasks.

1.3 Publications

Here I provide a list of publications resulting from this dissertation. To clarify the roles of different authors, I am the first author of all publications listed and provide the majority of ideas, texts, figures and experiments in these publications. Pietro Lio and Mateja Jamnik had important roles in advising, guiding, suggesting modifications and proof-reading all listed publications. I host most of the codes in my github page: `https://github.com/thematrixduo`.

1. Duo Wang, Mateja Jamnik, and Pietro Liò. Investigating diagrammatic reasoning with deep neural networks. In Peter Chapman, Gem Stapleton, Amirouche Moktefi, Sarah Perez-Kriz, and Francesco Bellucci, editors, *International Conference on Theory and Application of Diagrams*, LNCS 10871, pages 390–398. Springer, 2018. URL `https://link.springer.com/chapter/10.1007/978-3-319-91376-6_36`

2. Duo Wang, Mateja Jamnik, and Pietro Lio. Unsupervised and interpretable scene discovery with discrete-attend-infer-repeat. *International Conference of Machine Learning, Self-Supervised Learning Workshop*, 2019. URL `https://drive.google.com/file/d/0B4M2lUVyJzS4d29IN2pyUDR5ME8zV0RzdlJtZGdzM2xLV2Vv/view`

3. Duo Wang, Mateja Jamnik, and Pietro Lio. Unsupervised extraction of interpretable graph representations from multiple-object scenes. *International Conference of Machine Learning, Learning and Reasoning with Graph-Structured Representations Workshop*, 2019. URL `https://graphreason.github.io/papers/20.pdf`

4. Duo Wang, Mateja Jamnik, and Pietro Lio. Abstract diagrammatic reasoning with multiplex graph networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ByxQB1BKwH

5. Duo Wang, Mateja Jamnik, and Pietro Lio. Generalisable relational reasoning with comparators in low-dimensional manifolds. *Arxiv Preprint*, 2020. URL https://arxiv.org/abs/2006.08698

Chapter 2

Background

The main contributions of this dissertation, as outlined in Section 1.2, are about applying Deep Neural Networks on Diagrammatic Reasoning tasks. In this chapter I give a brief review of diagrammatic reasoning, with focuses on Euler diagram syllogisms and Raven Progressive Matrices. I also review Deep Neural Networks, and relevant variants including Convolutional Neural Networks (CNN), Graph Neural Networks (GNN) and deep generative models. Before that, I will first establish a common notational framework used throughout this dissertation.

2.1 Notational framework

I aim to keep the notation use in this dissertation clear and consistent. Below is a list of notations used that are largely consistent with notations used by [31].

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
A	A tensor
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\boldsymbol{I}$	Identity matrix with dimensionality implied by context
$\boldsymbol{e}^{(i)}$	Standard basis vector $[0, \dots, 0, 1, 0, \dots, 0]$ with a 1 at position i
$\text{diag}(\boldsymbol{a})$	A square, diagonal matrix with diagonal entries given by $\boldsymbol{a}$
$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\{0, 1, \dots, n\}$	The set of all integers between 0 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b

$a \| \| b$	The concatenation of a and b
$\log x$	Natural logarithm of x
$\|\|\boldsymbol{x}\|\|_p$	L^p norm of $\boldsymbol{x}$
$\|\|\boldsymbol{x}\|\|$	L^2 norm of $\boldsymbol{x}$
$\text{Var}(f(x))$	Variance of $f(x)$ under $P(\text{x})$
$D_{\text{KL}}(P\|Q)$	Kullback-Leibler divergence of P and Q
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$
$\mathbb{E}_{\text{x}\sim P}[f(x)]$ or $\mathbb{E}f(x)$	Expectation of $f(x)$ with respect to $P(\text{x})$

2.2 Diagrammatic Reasoning

Diagrammatic reasoning concerns logical reasoning with entities represented in a diagrammatic fashion. Entities in diagrams can be abstract representations of concepts and things such as the class of mortal beings in existential diagrams, a function unit in workflow diagrams, or market shares of a particular company in a Pie Chart. While there are various types of different diagrammatic systems, in this section I will introduce the two diagram types that are used in this dissertation. They are Euler diagrams and Raven Progressive Matrices.

2.2.1 Euler Diagrams and Syllogisms

Euler diagrams [22, 38] are simple, yet effective diagrammatic representations for reasoning about set relationships. We will use a colour-coded modification of the Gergonne's system of Euler diagrams [27] for its simplicity and visual clarity. In this system, minimal regions are assumed to be non-empty (*i.e.*, the Gergonne system of Euler diagrams assumes existential import, which means empty sets cannot be represented by contours), and shading is not used. I assign a distinct colour to each contour instead of alphabet labels to denote classes. Colour coding facilitates the training of neural networks by reducing the need to associate Alphabet labels with circled regions. There can be four different relationships between two sets A and B, which are:

- $A \supset B$
- $A \subset B$
- $A \cap B \neq \emptyset$
- $A \cap B = \emptyset$

While in theory the fifth relationship $A = B$ is also possible, it is not considered in this work because colour-coded contours will completely overlap and thereby diminish

visual clarity. While this limits the expressiveness of the diagrams, this is needed so as to simply the task for hypothesis testing. Future versions of the dataset can extend to the complete Euler diagrams system with symbol annotations. It is possible to fix this by using contour labels, which will be left for future investigation. Figure 2.1 illustrates how these 4 different set relationships can be represented by 4 different categories of colour-coded Euler diagrams (two sets denoted by Red and Green).

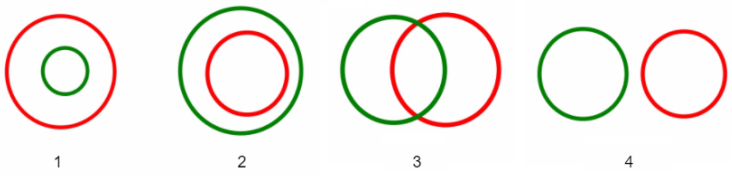


Figure 2.1: Euler diagrams representing 4 possible relationships between non-empty sets G (Green) and R (Red). (1)$G \subset R$. (2) $G \supset R$. (3) $A \cap B \neq \emptyset$. (4) $A \cap B = \emptyset$.

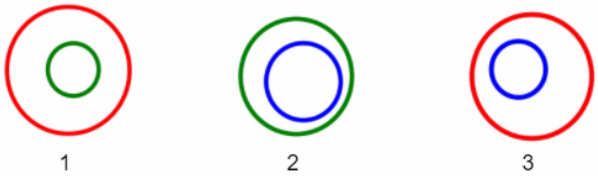
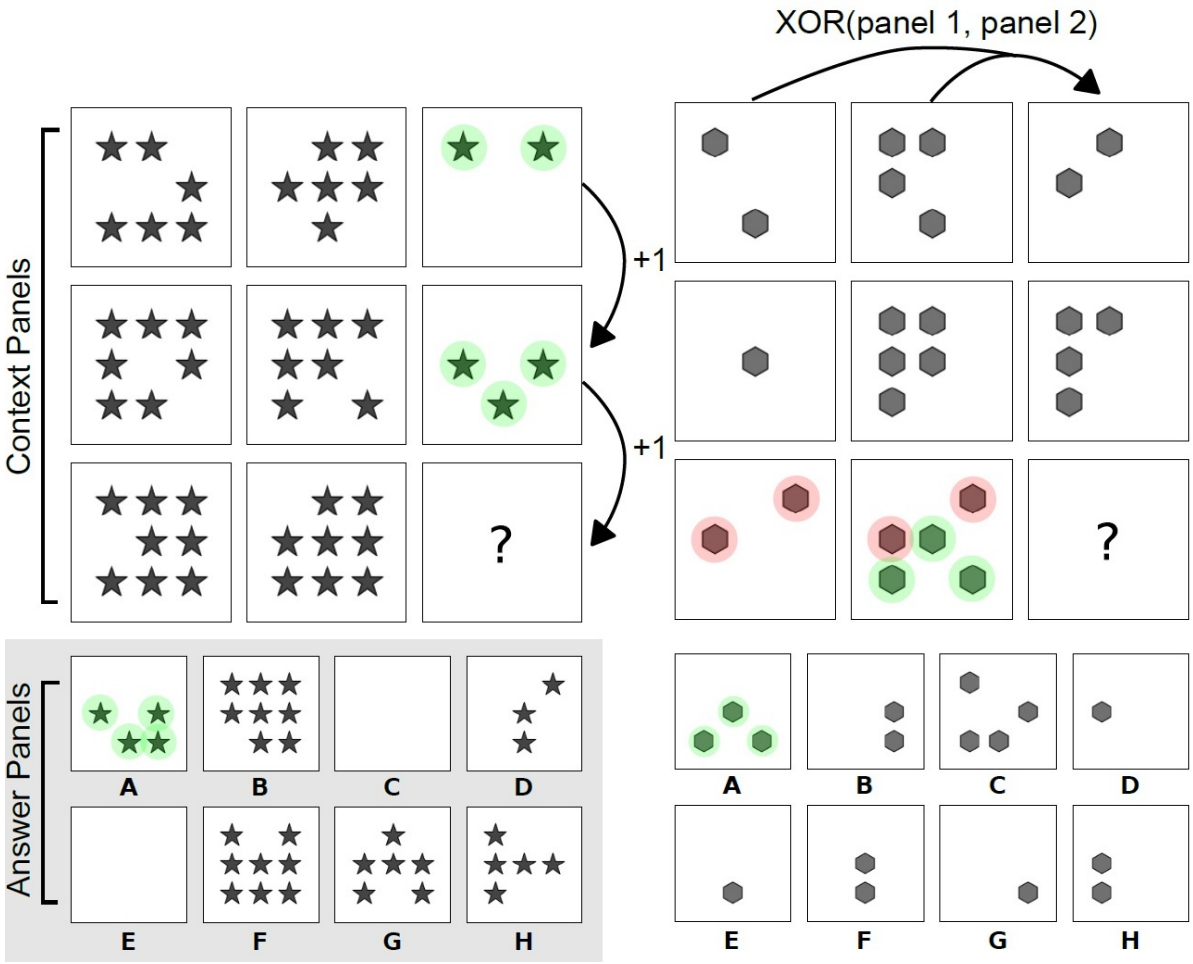


Figure 2.2: Euler diagrams representing the syllogism "All *Green* are *Red*, all *Blue* are *Green*, therefore all *Blue* are *Red*".

Euler diagrams are very effective in representing syllogisms. A syllogism consists of two premises that entail a conclusion. Figure 2.2 illustrates how the colour-coded Euler diagrams represent the syllogism "All *Green* are *Red*, all *Blue* are *Green*, therefore all *Blue* are *Red*". In this dissertation, I do not enforce the fixed-size contour constraint, which means that contours representing the same class can have varying sizes in different diagrams. As this Euler diagram system does not represent partial information, for certain premises there is not a single directly implied conclusion diagram, but several diagrams that are self-consistent [91, 88] with the given premises. An example would be for premises "All B are A, some C are B", consistent conclusions include "some C are A" and "all C are A".

2.2.2 Raven Progressive Matrices

In this section I describe Raven Progressive Matrices (RPM) in the context of the PGM dataset [5] and the RAVEN dataset [120]. RPM is a popular human fluid intelligence test, and has been recently used for measuring a machine's abstract reasoning capability [5]. MXGNet, discussed in Chapter 4, is the SOTA model (correct at the time of publication)

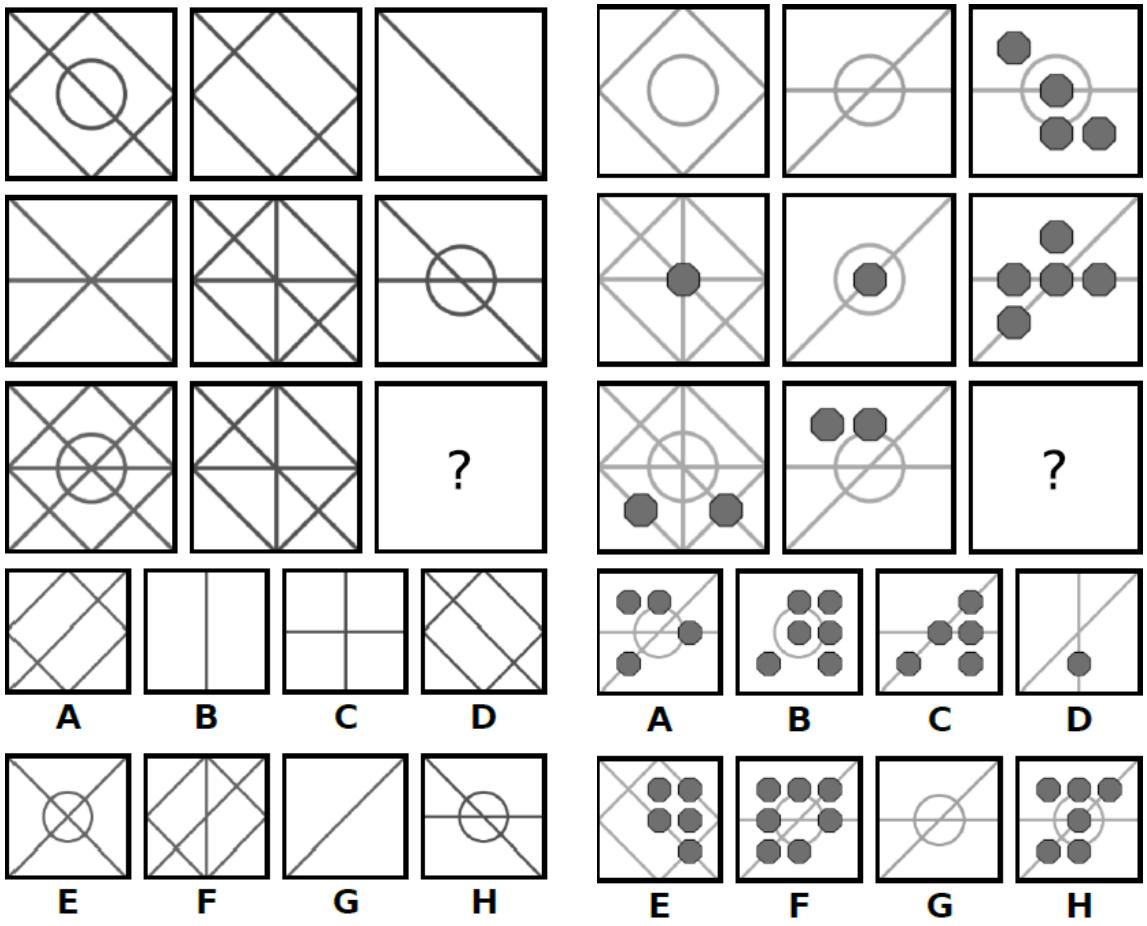


(a) (b)

Figure 2.3: Two examples in PGM dataset. (a) task contains a 'Progression' relation of the number of objects across diagrams in columns while (b) contains an 'XOR' relation of position of objects across diagrams in rows.

on the two RPM datasets. RPM tasks usually have 8 context diagrams and 8 answer candidates. The context diagrams are laid out in a 3×3 matrix $\mathbf{C}$ where $\boldsymbol{c}_{1,1}, \ldots \boldsymbol{c}_{3,2}$ are context diagrams and $\boldsymbol{c}_{3,3}$ is a blank diagram to be filled with 1 of the 8 answer candidates $\mathbf{A} = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_8\}$. One or more relations are present in rows or/and columns of the matrix. Figure 2.3a and 2.3b show two examples from the PGM dataset (Image courtesy [5]). The first example contains a 'Progression' relation of the number of objects across diagrams in columns. The second example contains an 'XOR' relation of the position of objects across diagrams in rows. With the correct answer filled in, the third row and column must satisfy all relations present in the first 2 rows and columns (in the RAVEN dataset, relations are only present in rows). In addition to labels of correct candidate choice, both datasets also provide labels of meta-targets for auxiliary training. The meta-target of a task is a multi-hot vector encoding tuples of (r, o, a) where r is the type of a relation present, o is

the object type and a is the attribute. For example, the meta-target for Figure 2.3b (a) encodes $(XOR, Shape, Position)$. The RAVEN dataset also provides additional structured labels of entity layouts (e.g., the number, types and position of entities) in the diagrams. However, we found that structured labels do not improve results, and therefore did not use them in our experiments.



(a) (b)

Figure 2.4: Two examples in the PGM dataset containing background line objects. (a) contains OR relation of the lines in the columns of the matrix. The correct answer is H. (b) contains both a OR relation of the lines and a *PROGRESSION* relation in the number of octagons in the columns of the matrix. The correct answer is H.

In addition to shape objects, diagrams in the PGM dataset can also contain background line objects that appear at fixed locations. Figure 2.4a and 2.4b show two examples of PGM tasks containing line objects.

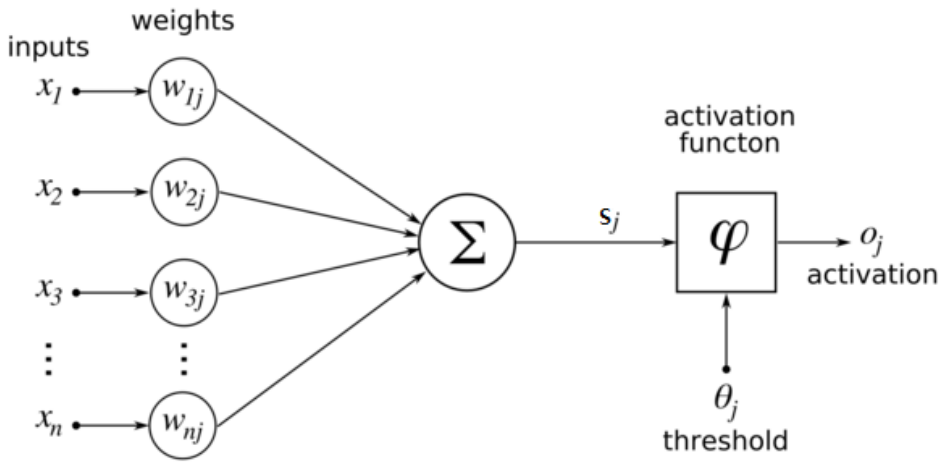


Figure 2.5: Computational model of a Single Artificial Neuron.

2.3 Artificial Neural Networks

Artificial Neural Network (ANN) is an information processing paradigm that is inspired by biological neural systems. In general, an ANN consists of layers of artificial neurons connected in a graph, most often in an acyclic directed form. However, in some types of ANN the connection can be cyclic and undirected. Artificial Neurons are the fundamental computation units in ANN. Figure 2.5 illustrates the computational architecture of an Artificial Neuron. Inputs, denoted as $x_i; i \in 1 \dots n$, are first multiplied with the weight w_{ij} where i corresponds to input index and j corresponds to the j^{th} neuron. The weighted inputs are then summed as:

$$s_j = \sum_{i=1}^{n} w_{ij} x_i \tag{2.1}$$

The sum s_j is then added with a bias θ_j and passed through a non-linear activation function φ to produce the final output of the neuron:

$$o_j = \varphi(s_j + \theta_j) \tag{2.2}$$

φ can be any non-linear function such as Sigmoid function $\frac{1}{1+e^{-x}}$ and Rectified Linear Unit $max(x, 0)$. A single Artificial Neuron has very limited computational capability. However, when many of them (usually at least in thousands) are connected together to form an ANN, very complex functions can be approximately represented by this ANN.

Before the application of back-propagation algorithms, only single layers linear perceptrons can be constructed. A back propagation algorithm, first applied to neural network training by Rumelhart et al [84], applies chain rules of differentiation to back propagate errors from higher layers to lower layers in order to correct weights assigned to lower layer inputs. This algorithm allows researchers to construct multilayer ANN with nonlinear

activation that can represent much more complex functions. The back-propagation algorithm computes a partial derivative (or gradient) of error functions with respect to the weights of each connections. Once the gradient for each weight is obtained, Stochastic Gradient Descent (SGD) can be applied to change weights in the direction that improves the objective function. The update of weights in steps is detailed in Equations 2.3 and 2.4.

$$w_{i,j}^{l} = w_{i,j}^{l} - \alpha \frac{\partial E}{\partial w_{i,j}^{l}} \quad E = error function \tag{2.3}$$

$$\frac{\partial E}{\partial w_{i,j}^{l}} = a_{j}^{l} \delta_{i}^{l+1}, \quad \delta_{i}^{l} = w_{i}^{l} \delta_{i}^{l+1} \varphi'(z_{i}^{l}) \tag{2.4}$$

where $w_{i,j}^{l}$ is weights, α is learning rate, $\frac{\partial E}{\partial w_{i,j}^{l}}$ is the partial derivative of the error function with respect to weights. a_{j}^{l} is input, z_{i}^{l} is activation output, $\varphi'(z_{i}^{l})$ denotes the derivative of the non-linear neuron activation function, and δ^{l} is error at layer l. Errors are back-propagated by computing errors at lower layers with respect to higher layer errors. After a large number of training iterations, the weights could reach local minima or maxima that minimise training errors. ANN can subsequently be used to predict an output label or value for test data.

ANN is used in most of today's DNN architectures for processing feature vectors either presented directly as input or extracted via some feature extraction methods (such as using Convolutional Neural Networks). All proposed architectures in this dissertation use ANN as part of the models, mostly for processing extracted feature vectors.

2.4 Convolutional Neural Network

A Convolutional Neural Network (CNN) is an ANN architecture inspired by studying the animal visual cortex. Hubel et al [45], in their seminal work on cat's primary visual cortex, identified orientation-sensitive simple cells with overlapping local receptive fields and complex cells performing down-sampling-like operations. Fukushima et al [24] first adopted this research and developed the first generation of CNN named "Neocognitron". Neocognitron has 2 types of layers, namely, Simple-layer corresponding to simple cells and Complex-layer corresponding to complex cells. Simple-layer applies a convolution kernel to input images or feature maps to produce 2-dimensional convolution. Figure 2.6 illustrates a convolution operation. This operation can be mathematically described in Equation 2.5.

$$y(i,j) = \sum_{p=-K_1}^{K_1} \sum_{p=-K_2}^{K_2} G(p,q) x(i-p, j-q) \tag{2.5}$$

where $G(p, q)$ is a 2-D convolution kernel, x is the input, y is the output and K_1 and K_2 are widths of kernel $G(p, q)$. For each layer, the same convolution kernel with invariant weights is applied across the input 2-dimensional feature map (output map of the previous layer). The convoluted feature map is then fed into C-layer, which downsamples the feature map by averaging operations. Neocognitron is essentially a network of multiple stacked Simple Cell layers and Complex Cell layers, as illustrated in Figure 2.7.

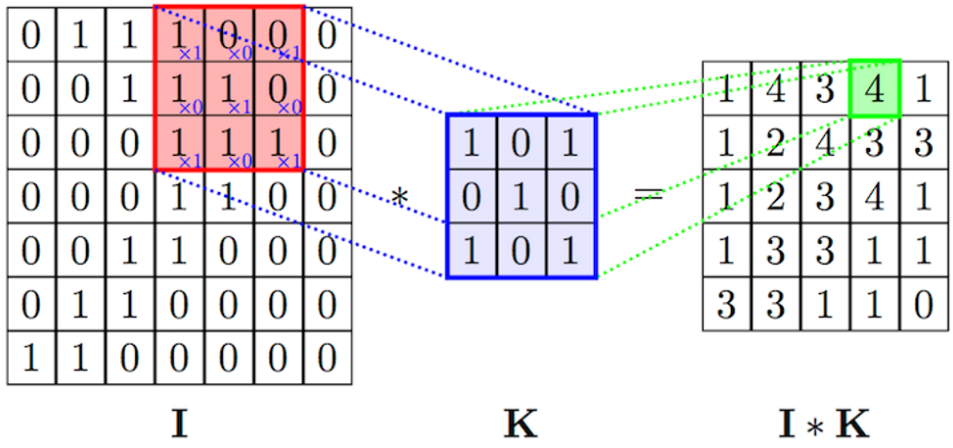


Figure 2.6: Illustration of Convolution Operation. I denotes image and K denotes convolution kernel.

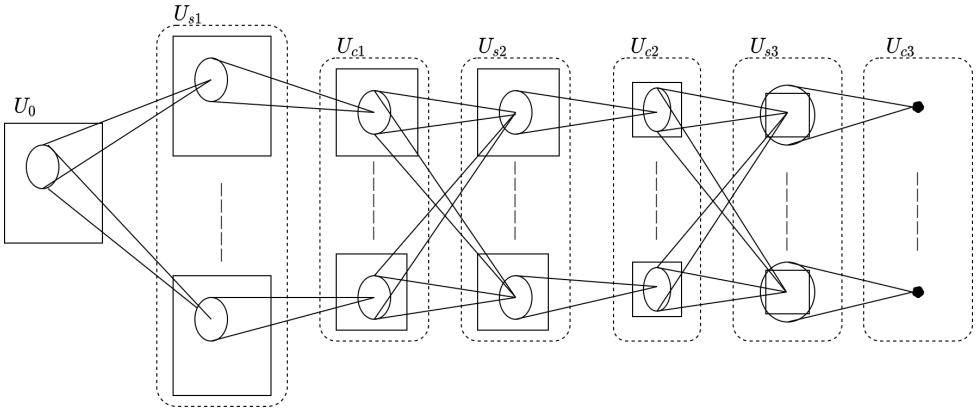


Figure 2.7: Illustration of Network Architecture of Neocognitron [24]

Neocognitron was not widely used due to difficulty in training and sub-optimal design of convolutional filters. Lecun et al [65] build upon this idea by both incorporating a back-propagation training algorithm and designing better structured filters and network architecture. Figure 2.8 shows the LeNet-5 Architecture by Lecun et al [65]. Layers labelled "C" are convolutional layers while layers labelled "S" are sub-sampling layers. At each layer there is a number of parallel feature maps, corresponding to distinct convolution kernels used. In this way, multiple features, such as edge, blob and corners, can be extracted at the same layer. At the top of LeNet-5 there are 2 fully connected layers that are

essentially the general model of an ANN. Lower "C" and "S" layers extract features while higher fully connected layers extract meanings from features. After training with standard back-propagation, LeNet-5 achieved 0.95% error rate on the MNIST dataset.

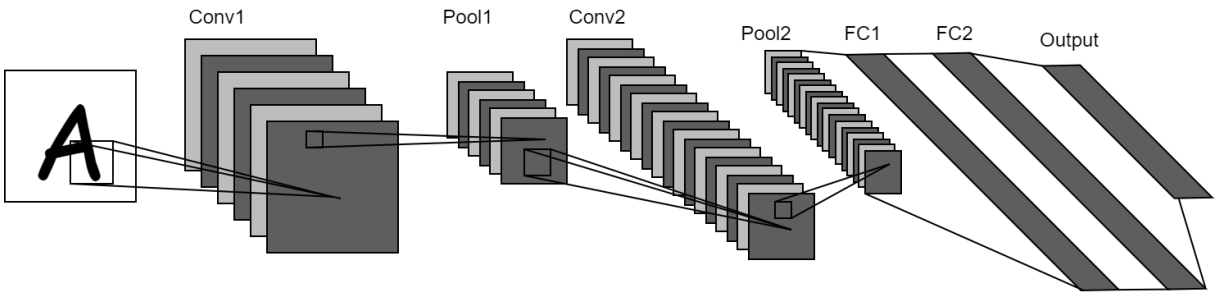


Figure 2.8: LeNet-5 [65] Convolutional Neural Network Architecture used for Handwritten Character Recognition

CNN models have three main advantages over ANN models. Firstly, CNN is spatial-invariant compared to ANN. Features can be anywhere on the input image. By applying the same convolution kernel over all locations in the image, the locations of a feature do not affects its detection. Secondly, CNN is more suitable for vision tasks because CNN takes into account spatial locality. For the vanilla ANN, swapping two pixels will not have any effect on training pixels because their spatial locations are not considered while training. A CNN, by applying kernels, can more accurately capture such spatial locality between neighbour pixels. Thirdly, a CNN has much less training cost per layer because only weights of a few convolution kernels need to be modified. However, for densely-connected ANN, all connection weights need to be updated at each training step.

In this dissertation, CNNs are used as the perception modules that precedes the different reasoning modules introduced in each chapters.

2.5 Selected improvements on training DNNs

While there is a vast amount of improvements in training Deep Neural Networks, I discuss three techniques that prove to be universally useful and are used in this dissertation. They are Batch Normalisation, Residual Networks and Adam SGD optimiser.

2.5.1 Batch Normalisation

Batch Normalisation [48] (BN) was originally proposed to tackle the Covariate Shift problem in DNN training. Covariate Shift means that during DNN training, weights are pushed in a direction that will change the intermediate layer statistics. However, it was recently shown that BN's usefulness in accelerating and stabilising training is not because of its effect on reducing Covariate Shift, but because it smooths the optimisation landscape [86]. Batch

Normalisation is essentially a normalisation layer with learnable affine transformation parameters. Denoting a batch of outputs of a NN layer as $\mathcal{B} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M\}$, a BN layer first computes batch mean $\boldsymbol{\mu}_\mathcal{B}$ and variance $\boldsymbol{\sigma}^2_\mathcal{B}$:

$$\boldsymbol{\mu}_\mathcal{B} = \frac{1}{M}\sum_{i=1}^{M} \boldsymbol{x}_i \qquad \boldsymbol{\sigma}^2_\mathcal{B} = \frac{1}{M}\sum_{i=1}^{M} (\boldsymbol{x}_i - \boldsymbol{\mu}_\mathcal{B})^2 \tag{2.6}$$

Then the BN layer normalises B to give $\{\boldsymbol{y}_i, \ldots, \boldsymbol{y}_M\}$ as:

$$\bar{\boldsymbol{x}}_i = \frac{\boldsymbol{x}_i - \boldsymbol{\mu}_\mathcal{B}}{\sqrt{\boldsymbol{\sigma}^2_\mathcal{B} + \epsilon}} \qquad \boldsymbol{y}_i = \gamma \bar{\boldsymbol{x}}_i + \beta \tag{2.7}$$

Here γ and β are learnable affine transformation parameters. These extra degrees of freedom allow the BN layer to transform the batch distribution as the best fit for the particular tasks. During the test stage, instead of using test batch statistics, it is more common to use the moving average training statistics as they are less spurious.

2.5.2 Residual Networks

While it is intuitive to think that deeper neural networks, with a larger amount of trainable parameters, have more processing power and should achieve improved accuracies, this is not the case in reality [39]. As the neural network gets deeper, both training and testing performance degrades. This is a strange behaviour because if deeper networks are not helping in improving performance, the NN can simply let selected layers become identity layers, so that the NN effectively becomes a shallower network. He et al [39] build on this idea to introduce Residual Networks, which simply allow skip connections that are usually the identity function:

$$f(x) = g(x) + x \tag{2.8}$$

Here $g(x)$ is a NN layer or a block of NN layers. The skipp connections allow information to skip $g(x)$ to reach deeper layers in NN. If $g(x)$ does not help in decreasing loss, it will be forced into small value ranges such that $f(x)$ effectively becomes an identity layer. Residual Networks allow training of much deeper networks without degrading performances. Deeper networks are considered to be more capable due to the larger number of trainable parameters. He et al [39] trained a 152-layer Residual Network which achieved state-of-the-art performance on ImageNet dataset.

Residual Networks are used in Chapter 4 and Chapter 6 as perception module that precedes the subsequent reasoning networks. In Chapter 4 the reasoning network also uses a residual architecture.

2.5.3 Adam optimiser

The simplest Stochastic Gradient Descent Method (SGD), as described in Equation 2.3, suffers from slow convergence and the possibility of getting stuck in saddle points or poor local minima, as the gradient is close to zero and there is no way to escape. Many different techniques have been explored to improve upon SGD. The the most successful one is Adam, which adds adaptive learning rates for weights and momentum. Let $\boldsymbol{g}_t = \frac{\partial E}{\partial \boldsymbol{w}}$ be the gradient with respect to $\boldsymbol{w}$ at time t, Adam maintains moving averages of momentums as:

$$\boldsymbol{m}_{t+1} = \beta_1 \boldsymbol{m}_t + (1-\beta_1)\boldsymbol{g}_t \quad \boldsymbol{v}_{t+1} = \beta_2 \boldsymbol{v}_t + (1-\beta_2)\boldsymbol{g}_t^2 \tag{2.9}$$

where β_1 and β_2 are mixing constants controlling the amount of intake of gradients into the moving average. The default values of the mixing constants are $\beta_1 = 0.9$ and $\beta_2 = 0.999$ (chosen empirically), which heavily biases the averages towards zero early on, so the following bias correction is applied:

$$\bar{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1-\beta_1} \quad \bar{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1-\beta_2} \tag{2.10}$$

Adam then updates weights adaptively based on $\bar{m}_t$ and $\bar{v}_t$ as:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w} - \alpha \frac{\bar{\boldsymbol{m}}_t}{\sqrt{\bar{\boldsymbol{v}}_t} + \epsilon} \tag{2.11}$$

where α is the learning rate, and ϵ is a small constant to prevent division by zero. Dividing by $\sqrt{\bar{\boldsymbol{v}}_t}$ for more variant gradients (larger $\boldsymbol{g}_t^2$ and thus larger accumulated $\boldsymbol{v}_t$) will make the update smaller. Adam optimiser is widely used to train models for all range of tasks, and are shown to accelerate training universally.

Due to the effectiveness of Adam optimiser, it is used for NN training throughout this dissertation.

2.5.4 Variational Auto-Encoders

In this section I briefly describe Variational Auto-Encoders (VAE) [57], a generative model used in Chapter 5 and Chapter 6. The core idea of variation inference is to use an approximate distribution $q_\phi(z|x)$ to approximate the true latent posterior distribution $p(z|x)$, and to optimise a Evidence Lower Bound Objective (ELBO) which is a lower bound to $p(x)$:

$$\log p(x) > \mathcal{L}(x, \theta, \phi) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x, z) - \log q_\phi(z|x)] \tag{2.12}$$

here $p_\theta(x|z)$, the data generating distribution, can be considered as the decoder of an Auto-Encoder while $q_\phi(z|x)$ can be considered as the encoder of an Auto-Encoder. This

equation can be re-arranged into:

$$\mathcal{L}(x, \theta, \phi) = -D_{\mathrm{KL}}(q_\phi(z|x)\|p(z)) + \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] \tag{2.13}$$

$$D_{\mathrm{KL}}(p(z), q(z)) = \sum_{z \in \mathcal{Z}} p(z) \log \frac{p(z)}{q(z)} \tag{2.14}$$

Here D_{KL} denotes KL divergence between two probability distributions (Equation 2.14). In this case the KL divergence between the approximating distribution $q_\phi(z|x)$ and prior distribution $p(z)$ is measured. The second term on the RHS measures data likelihood from z sampled from distribution $q_\phi(z|x)$. The first KL term penalises deviation of $q_\phi(z|x)$ from $p(z)$ while the second term is a reconstruction loss penalising less accurately generated image reconstructions.

2.6 Graph Neural Networks

While Graph Structured Data and Graph Neural Networks (GNNs) are not the focus of this dissertation, I employed GNNs as tools to model relations between objects in an image. Therefore, I briefly introduce GNs in this section.

I start by first defining a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ denotes the set of vertices in the graph and $\mathcal{E}$ denotes the set of edges. For each node $v_i \in \mathcal{V}$ there is an associated feature $x_i \in \boldsymbol{X}$. There are two popular tasks for graph structured data, which are node classification and graph classification. I will briefly discuss both tasks next.

2.6.1 Node classification

In a node classification task, a GNN takes as input the following: node features $\boldsymbol{X} \in \mathbb{R}^{N \times F}$ with N nodes and feature embedding size F, and an adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ where the entry $\boldsymbol{A}_{i,j}$ denotes an edge $e_{i,j} \in \mathcal{E}$ connecting node v_i to v_j. For most datasets the graphs are undirected and unweighted, which means $\boldsymbol{A}$ is a binary symmetric matrix. For transductive learning, part of the nodes are labelled with a category, and the goal is to classify other unlabelled nodes. For inductive learning, the training graphs are fully labelled while the test graphs are unseen and unlabelled. A popular node classification task is citation networks [89] where nodes are papers and edges indicate citations between papers. The labels are fields to which the paper belongs.

While there are many variants of GNN developed for node classification tasks, I only discuss a GNN family used in this dissertation that includes the majority architectures, namely Message Passing Neural Networks (MPNN). In a MPNN, nodes aggregate messages

passed by neighbour nodes such that the processing of the node features takes the neighbourhood information into account. Formally, a GNN layer in an MPNN can be defined as:

$$\begin{aligned} m_j^l &= F^l(h_j^{l-1}) \\ e_i^l &= \mathsf{AGGREGATE}_{j \in N(i)}(a_{ij}^l m_j^l) \\ h_i^l &= \sigma(\mathsf{COMBINE}(e_i^l, F^l(h_i^{l-1}))) \end{aligned} \tag{2.15}$$

here m_j^l are messages of node j (neighbours of node i according to Adjacency matrix) at layer l. F^l is an MLP layer processing the feature h_j^{l-1} from the previous layer. e_i^l are the aggregated features from all neighbours of node i, weighted by an attention parameter a_{ij}^l. AGGREGATE can be any type of pooling function such as mean or max pooling. e_i^l are then combined with the features of node i with COMBINE function, which is typically a summation or MLP. σ is the activation function of the GNN layer. Variants of MPNN differ in different stages of the information processing pipeline. For example, Graph Convolutional Network [60] have a_{ij}^l as a constant equal to the inverse of node degrees of i and j, while Graph Attention Networks [103] have a_{ij}^l learnt based on features of node i and j.

2.6.2 Graph classification and regression

In Graph classification tasks, the target is to classify a whole graph instead of individual nodes. For example, a protein molecule can be formulated as a graph of atoms, and the goal is to either classify the protein molecule (e.g., toxic or non-toxic) for classification tasks or to predict some continuous properties (e.g., binding affinity) for regression tasks. Formally, a GNN takes as input node feature $\boldsymbol{X} \in \mathbb{R}^{N \times F}$ with N nodes and feature embedding of size F, and an adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, and output a single label vector $o \in \mathbb{R}^K$ where K is the label size.

Many GNNs developed for graph classification and regression tasks are also in the MPNN family. The difference is that GNNs for graph-level tasks include coarsening and pooling layers that summarise learnt node features. Most approaches fall into two categories. In the first approach, the GNNs (e.g., [19, 16]) use a global pooling to aggregate node features either at each layer or at the final layer. Formally, the GNNs make predictions as:

$$o = f_{out}(\mathsf{POOL}(h_i^l | l \in \mathbb{L}, i \in N)) \tag{2.16}$$

where $\mathbb{L}$ denotes the set of layers to be pooled, POOL is a pooling function and f_{out} is a final MLP processing the pooled features. In the second approach, the GNNs (*e.g.* [10, 115]) aggregate node representations into clusters which coarsen the graph in a hierarchical manner. The coarsening layer takes a set of nodes $\mathbb{V}$ and the adjacency matrix $\boldsymbol{A}$, and

combine nodes in clusters into new nodes in a reduced set of nodes $\mathbb{V}_r$. A popular coarsening method is GraClus [17]. Pooling approaches (e.g., [115]) can be used to aggregate node features in the cluster into the features of the nodes in the coarsened graph.

Chapter 3

Investigating Euler Diagram Syllogism with Deep Neural Networks

Euler diagrams, introduced in Section 2.2.1, are simple, yet widely-used, diagrams for ontological reasoning. While several mechanised rule-based reasoning systems [93, 98] have been developed for Euler diagram reasoning, they all reason with symbolic representations of Euler diagrams, rather than raw diagram images. In this chapter, I test the following **hypothesis**: *if DNN-based reasoning systems can be trained to reason with raw Euler diagrams with robustness and interpretability.* To test this hypothesis, I design and build EulerNet, a DNN-based diagrammatic reasoning system that performs syllogism reasoning with Euler diagrams. Recall that a syllogism, as introduced in Section 2.2.1, has the structure $(\mathsf{MajorPremise}, \mathsf{MinorPremise}) \rightarrow \mathsf{Conclusion}$. EulerNet takes two Euler diagrams representing the premises as input. Contours in each Euler diagram represent sets and overlapping between contours indicates set relationships. EulerNet can generate a categorical conclusion (subset, intersection, or disjointedness) about the relationship between the sets with 99.5% accuracy. EulerNet can also learn to generate Euler diagrams that represent the set relationships without using any additional drawing tools. I further test EulerNet's robustness by adding significant noise and random deformation to input diagrams. Experiments show that EulerNet is very robust to noisy input, with just a 0.4% decrease in accuracy.

EulerNet, while learning to perform the reasoning task, simultaneously learns intermediate feature representations of the input diagrams and the reasoning problem. Additional experiments are performed to show that the learned representations encode essential information of diagrams and reasoning tasks. Firstly, I showed via a t-SNE [69] plot that the feature representations of same-category Euler diagrams and same-category syllogisms are clustered together in the feature space. This means EulerNet correctly learns which diagrams are more similar to each other. Secondly, I manipulated neural codes produced

by the reasoning network for conclusion diagram generation, and discovered that certain elements control essential statistics in the generated diagram, such as the area of contour intersection and the size of contour. Lastly, I showed that the reasoning network can be cast as a rule-based system, similar to what is currently used in automated reasoning systems, with only 0.7% loss in accuracy.

In the rest of this chapter I first present EulerNet in Section 3.1. Next, I evaluate EulerNet in Section 3.2, and finally discuss these results in Section 3.3.

3.1 EulerNet Architecture

3.1.1 EulerNet for categorical output

I built a system, EulerNet, which is a neural network trained to solve syllogisms represented with Euler diagrams. EulerNet takes two Euler diagrams (see detailed discussion of Euler diagrams in Section 2.2.1) representing the premises of the syllogism as input, and outputs a categorical conclusion (subset, intersection or disjointedness) for the syllogism. Figure 3.1 shows the architecture of EulerNet. The first input diagram shows a relationship between a set *Red* and a set *Green*. The second diagram shows a relationship between sets *Green* and *Blue*. There are four possible categories for a categorical conclusion output from EulerNet, namely:

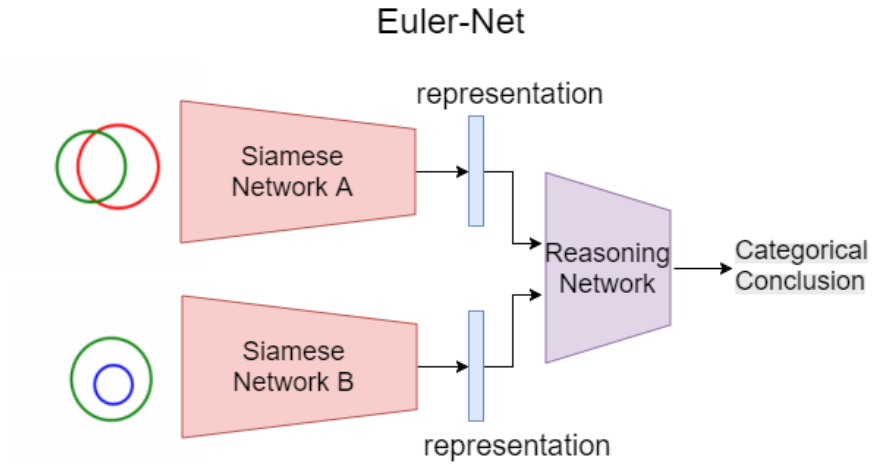


Figure 3.1: Overview of the EulerNet neural network architecture for Euler diagram syllogism reasoning. Siamese Networks are two convolutional networks that share the same weights. The Reasoning network takes diagram feature embeddings and output the predicted conclusion.

1. *Red* is a subset of *Blue*.
2. *Blue* is a subset of *Red*.

3. *Red* intersects *Blue*.

4. *Red* is disjoint from *Blue*.

In the case of indeterminacy, when no single logical conclusion can be drawn (e.g., All *Green* are *Red*, some *Green* are *Blue*), the neural network outputs all conclusions that are consistent (not contradicting) with the premises. Neural networks are trained with pairs of premise diagrams and labels that encode correct conclusions into a binary (0 or 1) vector of length 4, with 4 positions in the vector representing each conclusion category in the order of $A \supset B$, $A \subset B$, $A \cap B \neq \emptyset$ and $A \cap B = \emptyset$. For example, vector [0100] encodes $A \subset B$. While developed on the classical syllogism, EulerNet can be applied to diagram tasks with an arbitrary number of contours in the diagram and an arbitrary number of diagrams in the task. I also applied EulerNet on tasks where there are 3 contours in each diagram. Namely, the first diagram contains *Red*, *Green*, and *Blue* contours, while the second diagram contains *Green*, *Blue*, and *Yellow* contours. The task is to infer consistent relationships between classes *Red* and *Yellow*. In Section 3.2.1, I show that, for the 3-contour task, the decrease in reasoning accuracy is negligible.

EulerNet is composed of two modules. The first module is a Siamese convolutional network that recognises the diagrams and encodes them into high-level neural feature representations. Siamese networks share the same weights so each diagram is processed in the same way. This network has a similar function to the visual cortex in the human brain [113], which transforms visual stimuli into neural code. The convolutional network consists of 7 convolutional layers extracting increasingly abstract features from the input diagrams. The second module is a reasoning network that performs inferences on the neural presentations of diagrams. This reasoning network extracts useful information from the neural representations in order to achieve accurate inferences. The reasoning network consists of fully-connected layers that densely process the neural representations. The reasoning network outputs the probability for each categorical conclusion. EulerNet can be trained to minimise error rates in reasoning with standard Stochastic Gradient Descent (SGD) and a back-propagation algorithm [65]. Formally, the training objective is to minimise the loss function, as in Equation 3.1:

$$\mathcal{L}(\mathbf{D}, \mathbf{T}) = - \sum_{(\mathbf{d},\mathbf{t}) \in (\mathbf{D},\mathbf{T})} \sum_{i} t_i \log f(\boldsymbol{d}) + (1 - t_i)(1 - \log f(\boldsymbol{d})) \tag{3.1}$$

Where D are input premise diagrams, T are labels, (d, t) is a training sample of the problem set, t_i is the i^{th} element in the label vector, and $f(d)$ represents EulerNet as a function of d.

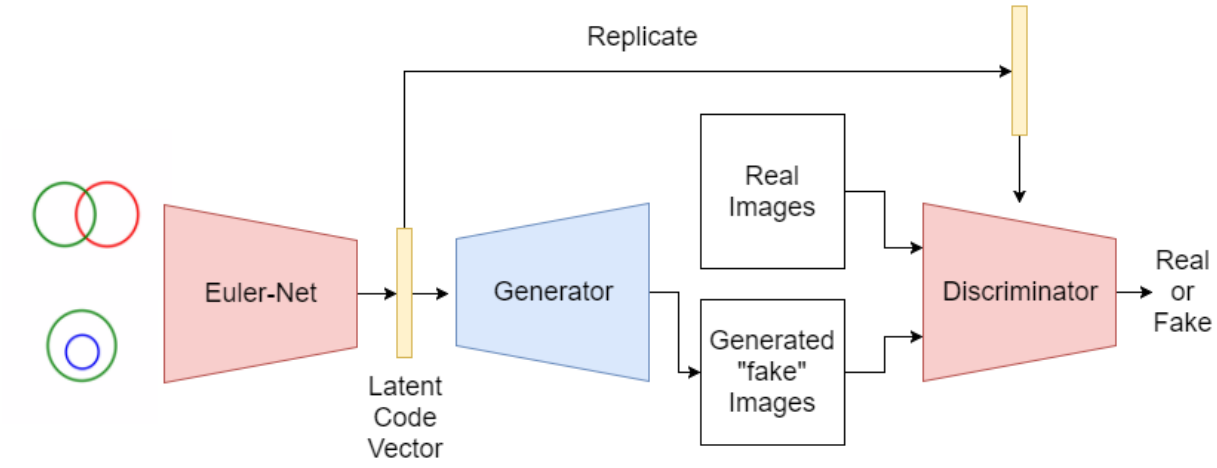


Figure 3.2: Diagram generation module for EulerNet. The neural code from the last fully connected layer in the reasoning network is fed into the diagram generator network. This generator is trained in combination with a discriminator that tries to distinguish between real and correct images from generated images, and thereby enables the generator to improve.

3.1.2 EulerNet for diagram generation

Instead of generating a categorical conclusion, EulerNet can also generate diagrammatic conclusions of a syllogism, as in diagram 3 in Figure 2.2 (page 27). This allows EulerNet to perform complete diagrammatic inferences. Diagrams can be generated from the neural representations of the syllogism problem without any human intervention or established drawing tools. This is accomplished by concatenating an image generator network to EulerNet. Figure 3.2 illustrates the architecture of EulerNet for diagram generation. This generator network uses latent neural code vectors extracted from the last layer of EulerNet to generate Euler diagrams that are consistent with the given premises. The latent neural code vectors encode consistent conclusions for the reasoning task. The generator network then uses several deconvolutional layers [118] to transform this neural representation to an Euler diagram consistent with the given premises.

The generator network is trained with the Generative Adversarial Network (GAN) [30] training objective, which recently became popular for generating high definition and sharp images. GAN consists of a generator network and a discriminator network that are jointly trained in a minimax game. The generator tries to generate images as real and accurate as possible, while the discriminator tries to distinguish between the generated and the correct images. The GAN training objective can be mathematically formulated as in Equation 3.2:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{\boldsymbol{x} \sim p_{data}(\boldsymbol{x})}[\log \mathcal{D}(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z})))] \tag{3.2}$$

Where G is the generator, D is the discriminator, x is a correct data sample, and z is the latent code vector. This can be viewed as a minimax game between G, which

Second Premise \ First Premise	$A \supset B$	$A \subset B$	$A \cap B \neq \emptyset$	$A \cap B = \emptyset$
$B \supset C$	$A \subset C$	$A \subset C$ $A \supset C$ $A \cap C \neq \emptyset$ $A \cap C = \emptyset$	$A \supset C$ $A \cap C \neq \emptyset$ $A \cap C = \emptyset$	$A \cap C = \emptyset$
$B \subset C$	$A \subset C$ $A \supset C$ $A \cap C \neq \emptyset$ $A \cap C = \emptyset$	$A \subset C$	$A \supset C$ $A \subset C$ $A \cap C \neq \emptyset$	$A \subset C$ $A \cap C \neq \emptyset$ $A \cap C = \emptyset$
$B \cap C \neq \emptyset$	$A \supset C$ $A \cap C \neq \emptyset$	$A \subset C$ $A \cap C \neq \emptyset$ $A \cap C = \emptyset$	$A \subset C$	$A \subset C$ $A \supset C$ $A \cap C \neq \emptyset$ $A \cap C = \emptyset$
$B \cap C = \emptyset$	$A \supset C$ $A \cap C \neq \emptyset$ $A \cap C = \emptyset$	$A \cap C = \emptyset$	$A \supset C$ $A \cap C \neq \emptyset$ $A \cap C = \emptyset$	$A \subset C$ $A \supset C$ $A \cap C \neq \emptyset$ $A \cap C = \emptyset$

Table 3.1: Logic table for syllogism inference. Each cell contains all consistent conclusions from the given first and second premises.

tries to minimise the objective, and D which tries to maximise it. During training, the parameters of the generator and the discriminator are updated alternatively to converge towards a dynamic equilibrium. The generator converges after 50000 iterations, and is able to generate an accurate and clear Euler diagram conclusion consistent with the given premises.

3.2 Evaluation

3.2.1 Syllogism reasoning performance

EulerNet is trained with syllogism problems generated from an Euler diagram syllogism task generator. This generator first generates two random logical relationships for the first two premises, and then generates two Euler diagrams representing the two logical relationships with random size and position, as long as the logical relationships are not violated. Subsequently, the task generator generates consistent conclusions from a manually constructed logic table, as shown in Table 3.1, that maps any two premises to a set of consistent conclusions. Columns of the truth table correspond to the first premise; rows correspond to the second premise. The entries in the table are sets of classes containing corresponding conclusions.

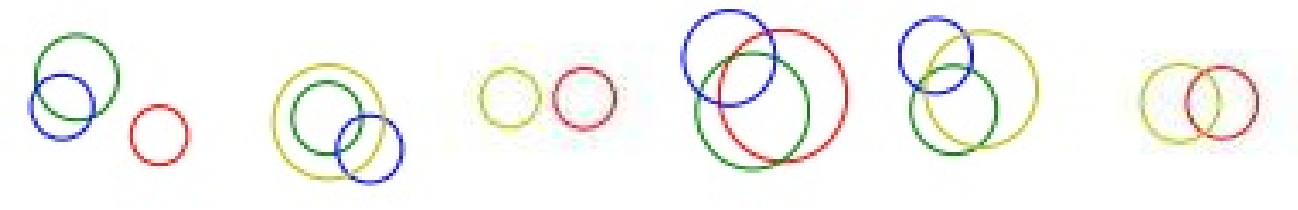
(a) p1 (b) p2 (c) c (d) p1 (e) p2 (f) c

Figure 3.3: Two examples of the EulerNet generated diagrams. p1 and p2 are premises and c are EulerNet generated conclusion diagrams.

For each consistent conclusion, corresponding Euler diagrams will also be generated with random size and position. In total, 96000 Euler syllogism reasoning problems are generated for neural network training. For the 3-contour dataset, the diagrams and conclusions are generated in the same fashion. The truth table is larger as each premise contains relationships between 3 classes, giving 4^3 possible cases.

During training, I followed the standard procedure of dividing the dataset into 3 sets, namely the training set, the validation set and the test set with a split ratio of 8:1:1. I trained EulerNet with the training dataset, tuned its performance with reported scores on the validation dataset, and finally, I evaluated the final performance on the test dataset. I report here the percentage accuracy, which is defined as the number of syllogism problems correctly solved over the total number of problems. EulerNet is able to achieve a nearly perfect accuracy of 99.5% on the 2-contour Euler syllogism tasks, and 99.4% accuracy on the 3-contour tasks. In order to understand the performance results further, I separate test results for conclusive syllogism groups (a single logical conclusion) and inconclusive groups (multiple consistent conclusions). I found that EulerNet achieves 100% accuracy for conclusive groups, which could indicate that conclusive syllogisms are relatively simpler than the inconclusive ones.

EulerNet, with an added diagram generator can create high quality Euler diagrams from neural representations mostly without image artefacts. Figure 3.3 shows two examples of Euler conclusion diagrams generated by EulerNet for the 2-contour task. For the first example, only the correct Euler diagram is generated. For the second example, which is inconclusive, an Euler diagram consistent with the premises is generated. Figure 3.4 shows two examples of generated diagrams (both examples are inconclusive so consistent conclusions are generated) for the 3-contour task. This shows that EulerNet learns neural representations that encode essential information of the syllogism reasoning problem. Such neural representations can be interpreted by a diagram generator network to create a diagrammatic conclusion.



(a) p1 (b) p2 (c) c (d) p1 (e) p2 (f) c

Figure 3.4: Two examples of the EulerNet generated diagrams for the reasoning task with 3 contours per diagram. p1 and p2 are premises and c are EulerNet generated conclusion diagrams.

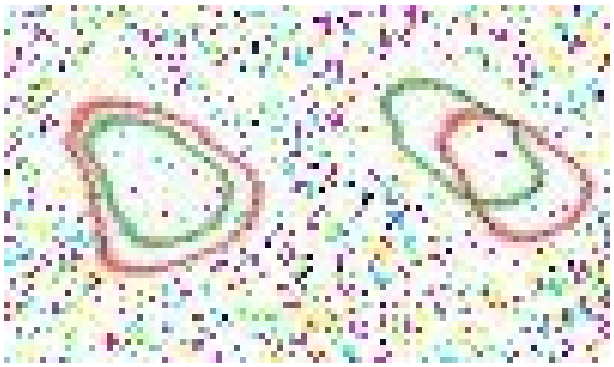


Figure 3.5: An example of Euler diagrams with injected random noise and deformation.

3.2.2 Robustness evaluation

One of the advantages of a DNN-based system is its robustness to noisy input. To test if EulerNet has this advantage, I add random noise and deformation to the input diagrams and measure EulerNet's decrease in performance. Figure 3.5 shows an example of Euler diagrams with injected random noise and deformation. In practice, I add Gaussian Noise sampled from $\mathcal{N}(0, 0.003)$ and random vertical sinusoidal deformation with amplitude uniformly randomly sampled between 0 to 8 pixels and angular frequency sampled between 0 to 4. With such noise and deformation, EulerNet can still achieve 99.1% test accuracy, with only a 0.4% decrease in accuracy. Thus, EulerNet is shown to be very robust to noisy input.

3.2.3 Decoding neural representations

It is interesting to further understand what exactly is learned in the neural representation of diagrams. Firstly, I applied t-SNE visualisation [69], a technique that enables visualising high-dimensional data in a 2-dimensional format, on neural representations of input premise diagrams. The t-SNE technique can efficiently compute 2-dimensional distances that are relatively proportional to distances between points in high dimensional space. With t-SNE I am able to visualise distances between feature representations of diagrams. Figure 3.6

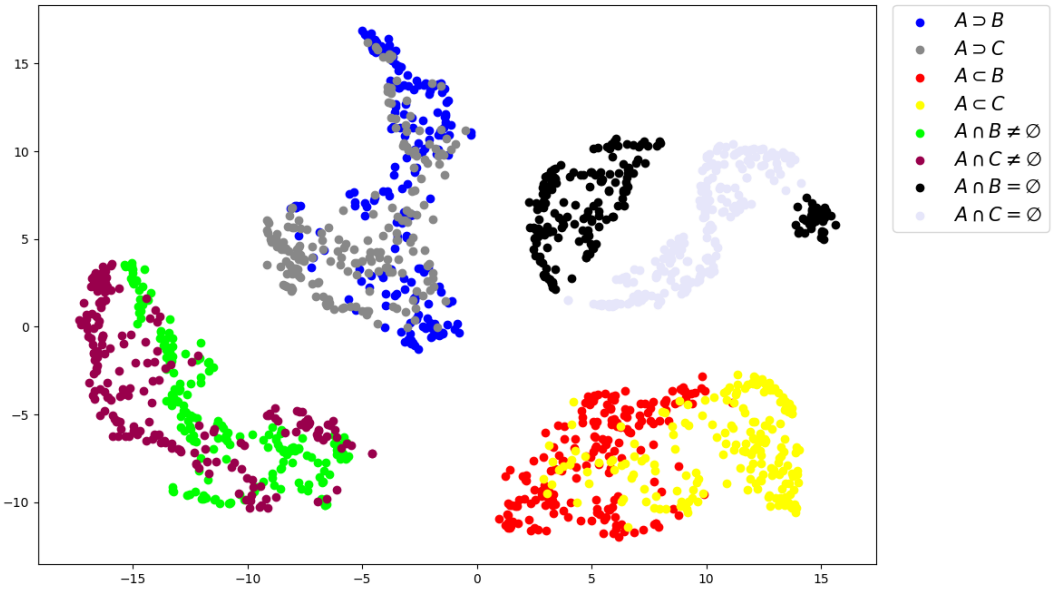


Figure 3.6: t-SNE plot of neural representations (output of Siamese Convolutional Networks) of input premise diagrams. In the legend, $\supset$ and $\subset$ denotes superset and subset, $A \cap B = \emptyset$ denotes intersection and $A \cap B \neq \emptyset$ denotes disjoint sets. Units in the plot are distances computed proportionally to the differences in feature presentation vectors of diagrams.

shows the t-SNE plot of neural representations of input premise diagrams, including diagrams showing relationships between the first set A and the second set B, and also between the second set B and the third set C. In the plot, one can observe that neural representations of diagrams with the same logical relationships are clustered together. This illustrates that the learned neural representations indeed encode the logic relationships in the diagrams. In Figure 3.7, I also plotted the neural representations extracted from the reasoning network in EulerNet, which should encode the syllogism reasoning problems as a whole. It can be observed again that the same syllogism problems are clustered together.

Besides the t-SNE plot, I would also like to investigate if elements in the neural representation vector encode functionally disentangled information, meaning that a single or a small group of dimensions in the representation vector solely encodes certain statistics of the image (e.g., contour size) without being entangled with other elements.

I manually varied elements of the neural vector that are extracted from the reasoning network and fed them into the generator network for diagram generation. I observed that indeed some elements do encode meaningful information of the generated diagrams. For example, I identified that the magnitude of Element 10 in the vector is correlated with the

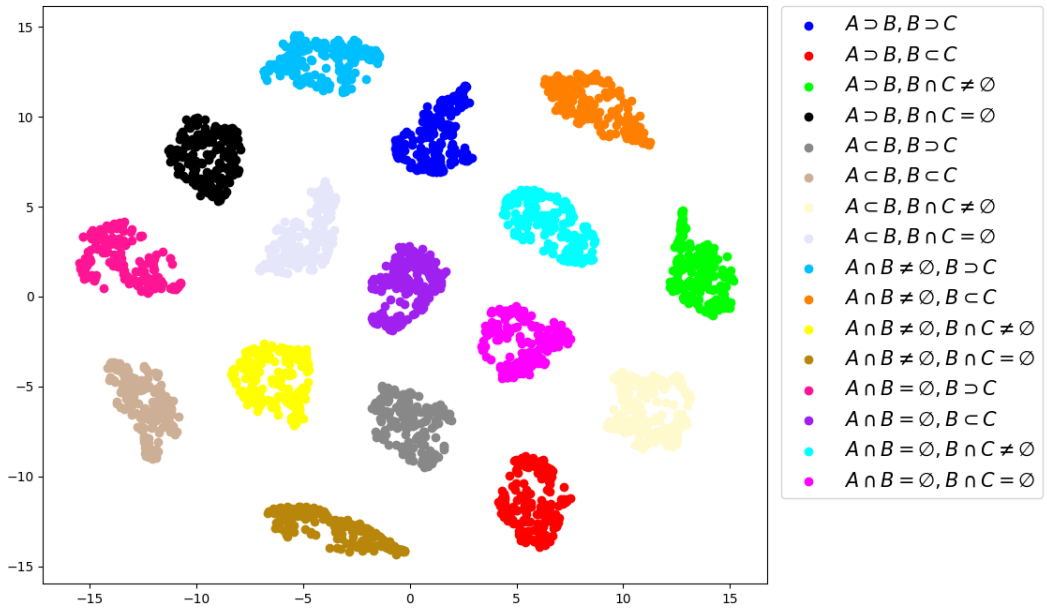
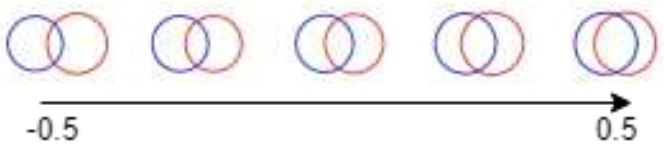


Figure 3.7: t-SNE plot of neural representations of the syllogism reasoning problems as a whole. In the legend, $\supset$ and $\subset$ denote superset and subset, $A \cap B = \emptyset$ denotes intersection, and $A \cap B \neq \emptyset$ denotes disjoint sets. Units in the plot are distances computed proportionally to the differences in feature presentation vectors of diagrams.

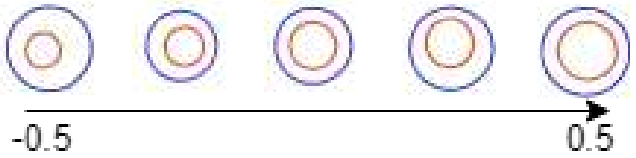
area of intersection between two circles, as illustrated in Figure 3.8 (a). I also observed that element 41 is correlated with the size of the inner Euler circle in the subset case, as illustrated in Figure 3.8 (b).

3.2.4 Extracting rules from reasoning networks

While EulerNet performs well on the diagrammatic reasoning tasks, it is not clear how the reasoning network processes the neural representations of each premise diagram. I introduce a method to transform a reasoning network into a rule-based system similar to what is widely used in automated reasoning. The method makes the reasoning network small and sparse without significant loss of accuracy. In this way, the parameters of the reasoning network can be distilled into a very small number, and then transformed into a rule-based program with a relatively small number of rules. I make the reasoning network small by reducing the number of layers and neurons in each layer with a loss in accuracy smaller than 0.5%. In this way, the reasoning is reduced to a network with 1 hidden layer and 1 output layer, each consisting of 4 neurons. The network parameters are then sparsified by adding an L-1 Norm regularisation term to the loss function in Equation 3.1.



(a)



(b)

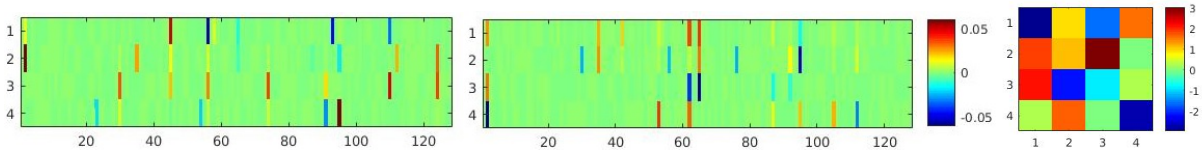
Figure 3.8: (a) Varying Element 10 affects the area of intersection between two circles. (b) Varying Element 41 affects the size of inner circle in the subset case. The diagrams, from left to right, correspond to varying codes from value -0.5 to 0.5.

L-1 regularisation term is mathematically formulated as in Equation 3.3:

$$||\mathbf{w}||_1 = \sum_i |\boldsymbol{w}_i| \tag{3.3}$$

where w_i is the i^{th} parameter, and $|w|$ is an absolute value of w. I force all weights with absolute values below a certain threshold ϵ to 0 to leave only parameters important to the reasoning task.

Figure 3.9 shows a heatmap plot of the sparsified weights of the reasoning network. With 128-dimensional neural representation of input premise diagrams, there are two weight matrices of 128×4 size mapping neural representations to the first layer in the reasoning network. There is also a 4×4 weight matrix connecting the first layer to the second layer. With ϵ value of 0.02, a reasoning network with only 53 effective parameters is obtained. The sparsified reasoning network can be more easily re-formulated as a rule-based system where each neural layer is simply a weighted sum with bias (implemented as matrix multiplication/addition) with a ReLU function $ReLU(x) = \max(x, 0)$. This rule-based system, while simple, can still achieve 98.8% accuracy. However, the full interpretability of this rule-based system depends on the fully interpretable diagram feature embedding produced by the perception module.



(a) p1 weights (b) p2 weights (c) reasoning weights

Figure 3.9: Visualising weights of the small and sparsified reasoning network. (a) and (b) are 128×4 weight matrices mapping neural representations to the first layer of the reasoner network. Weight values are indicated with colour, with green indicating values closer to 0. Note that most of the weight in the sparsified weight matrices are 0. (c) is a 4×4 weight matrix connecting the first layer (4 hidden units) to the second layer(4 output units).

3.2.5 Ablation studies

In this section, I perform ablation studies on components of EulerNet to identify parts of the model that affect the performance the most. I have performed the following ablation experiments:

1. I changed the reasoning network to a linear layer. Test accuracy dropped to 62.0%, showing that the Euler syllogism task cannot be solved as a simple linear classification problem, and a higher-order complex neural network is essential for improving the performance.

2. I changed the Siamese CNN from a 7-layer network to 3-layer network. Test accuracy dropped to 87.0%. This shows that deep neural networks are needed for learning a better representation of the input diagrams.

3. I remove the perception CNN module, and directly feed symbolic representations (in the form of encoding vectors) to the reasoning network. The test accuracy is 99.9%, showing that the Siamese CNN learns neural representations of similar quality to the symbolic representations.

3.3 Discussion

In this section I will first concisely discuss the pros and cons of EulerNet, and then expand on some of the points.
Pros:

- EulerNet has the ability to learn from the data rather than relying on manually-defined rules. Thus it can be more easily adapted to other types of diagrams. This is discussed in detail in Section 3.3.1.

- EulerNet, compared against traditional mechanised reasoning systems, does not require the definition of a diagram-to-symbol mapping system. EulerNet can implicitly learn such mapping. This is further discussed in Section 3.3.2.
- EulerNet is robust to noisy and deformed input diagrams, with only small decrease in accuracy. This has been shown in Section 3.2.2.

Cons:

- EulerNet encodes input diagrams into a fixed length vector, which is not scalable to diagrams with arbitrary numbers of entities. A solution is proposed in Chapter 4.
- EulerNet requires large amount of labelled data for training. However, this can be ameliorated with unsupervised pre-training on vastly available unlabelled data, which will be discussed in Chapter 5.
- EulerNet is currently only trained on a dataset with 16 distinct conclusion categories. It is possible that EulerNet simply learns a mapping between input classes to conclusion category. This is further discussed in Section 3.3.3

3.3.1 Applicability to other types of diagrams

While I showed that DNNs can perform diagrammatic reasoning and learn useful representations on the relatively simple Gergonne's system of Euler diagram syllogism solving, DNNs are not limited to a particular type of diagram. DNNs provide a universal method for encoding all types of diagrams into neural codes that can subsequently be analysed. While simple diagrams like Euler diagrams can be conveniently formalised symbolically into sets of zones, labels, and shadings, there are many types of diagrams that are more difficult to formalise, such as the diagram reasoning tasks like Raven Progressive Matrices. DNNs can be applied to learn representations of such diagrams, which enable us to analyse aspects of those representations. In the next Chapter (Chapter 4), I will discuss applying DNNs to the much more complex Raven Progressive Matrices task, and introduce MXGNet, an architecture I developed that achieves state-of-the-art accuracy on two RPM datasets (true at the time of publication).

3.3.2 Comparison with logical symbolic reasoner

While EulerNet achieves 99.5% accuracy, it is still not on-par with a symbolic logic reasoner which is 100% accurate. However, conceptually logical symbolic reasoners only reason with the symbols that represent diagrams, while EulerNet reasons directly with the raw diagram input. The strengths of DNN are its ability to capture feature representations for any type of diagram (as discussed in Section 3.3.1), the reduced need for human expert

inputs, and its robustness to noise. Such DNN-based diagrammatic reasoning systems make the most sense when applied to types of diagrams that have not been formalised, or are too complex to be formalised, like reasoning with natural images where objects in the image have logical meanings.

One of the obvious drawbacks of DNN-based systems is their need for labelled data. Labelling data can be a tedious and labour-intensive process. In Chapter 5, I discuss an unsupervised learning method that allows diagram-to-representation mapping to be learned without any labels. Moreover, the mapping outputs symbolic representation, which can be directly used for rule-based reasoning, thus providing a potential way of amalgamating DNN with mechanised reasoning systems.

3.3.3 Limitations of the syllogism dataset

The proposed dataset, while containing non-repeating 2-contour Euler syllogism diagram tasks, is limited in that there are only 16 distinct outcomes due to the definition of syllogism tasks. It is possible that EulerNet learns a table-mapping mechanism for mapping the 2 input diagram types to the conclusion. For future work, it is possible to generate Euler diagram inference tasks that contain an arbitrary number of input diagrams, and test if EulerNet can be modified to perform chained inference on this dataset. Such test will show if EulerNet can really learns to perform logic-based inference rather than just input-type classification.

Chapter 4

Abstract Diagrammatic Reasoning with Multiplex Graph Networks

Diagrams, while frequently used to facilitate reasoning like Euler diagrams, are also often used for measuring human's fluid intelligence. Many tests have been proposed to measure human fluid intelligence, and most of them are diagrammatic tests. The most popular test in the visual domain is the Raven Progressive Matrices (RPM) test [79]. In the RPM test, the participants are asked to view a sequence of contextual diagrams, usually given as a 3×3 matrix of diagrams with the bottom-right diagram left blank. Participants should infer abstract logic rules in rows or columns of the diagram matrix, and pick correctly from a set of candidate answers to fill in the blank. Figure 4.1a shows an example of an RPM task containing the XOR rule across diagrams in rows. For more detailed discussions and examples of RPM tasks, please refer to Section 2.2.2.

EulerNet (Chapter 3) works well for Euler diagrams in which there are fixed numbers of contours. However, in RPM task, there can be an arbitrary number of objects in the diagrams. EulerNet encodes each diagram into a fixed-length embedding vector, which are thereby not scalable to diagrams with an arbitrary number of objects. In this chapter, I test the **hypothesis**: *if scalable neural architectures can be developed for reasoning with input diagrams with varying number of entities*. For this purpose, I propose MXGNet, an end-to-end scalable architecture that treats each individual object as a node and formulates graphs of objects in the RPM diagrams. Developing a DNN-based reasoning system for RPM task makes two contributions. Firstly, applying a traditional mechanised reasoning system on RPM tasks is a complex endeavour that requires tremendous human efforts in designing the diagram-to-symbol mapping and defining the set of rules for reasoning. In fact, such a system [76] has only been applied to the simplest case of RPM, where there can only be a single object existing in each diagram. Instead, a DNN-based system can learn to solve much more complex tasks without human expert inputs. Secondly, for DNN

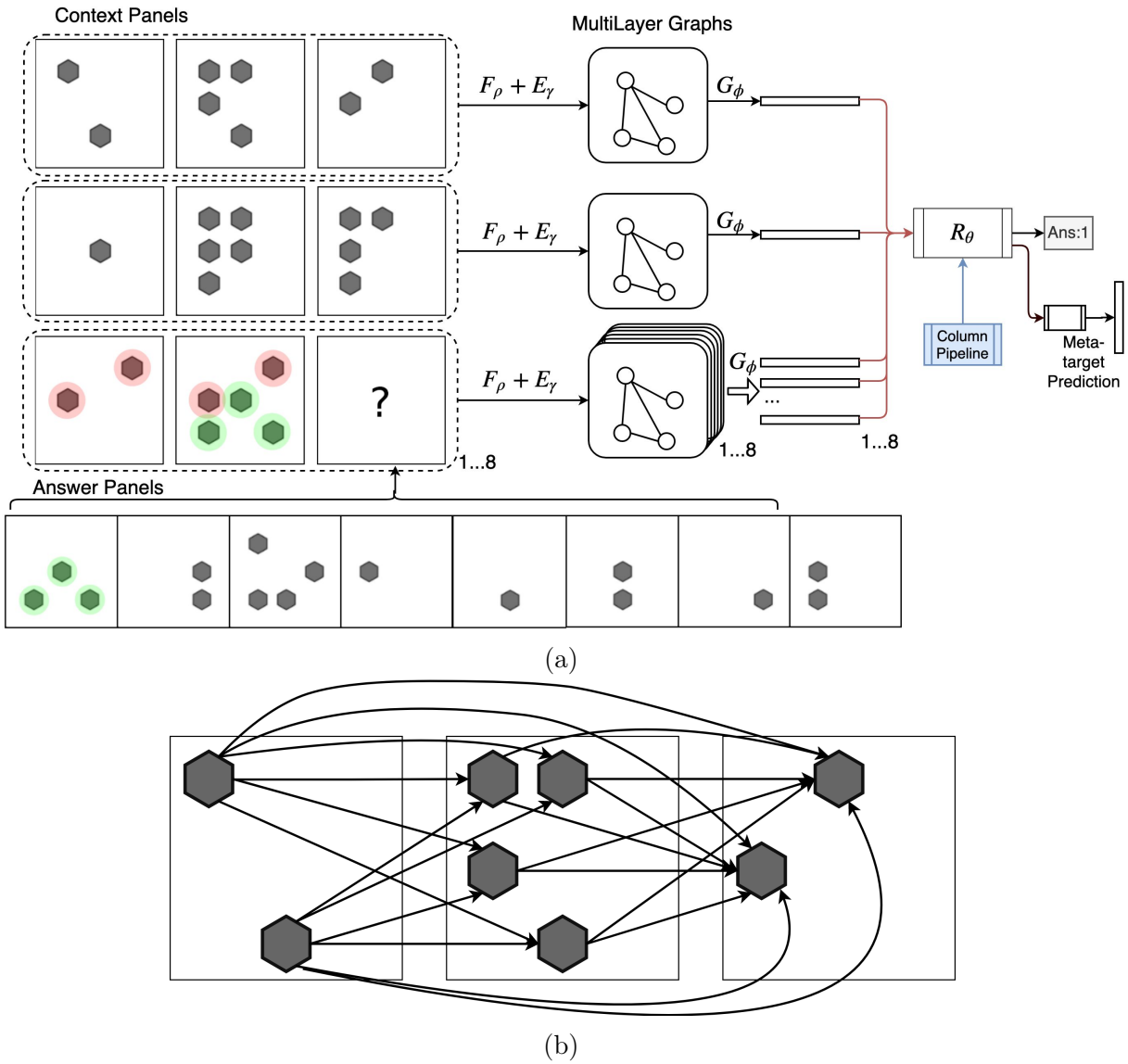


Figure 4.1: (a) shows an example of an RPM task containing the XOR rule across diagrams in rows and the overview of MXGNet architecture. Here, F_ρ is the object representation module, E_γ is the edge embedding module, G_ϕ is the graph summarisation module and R_θ is the reasoning network. (b) shows the multi-layer graph formed from objects in the first row of diagrams in the example shown in (a). Edges can capture relations such as if two objects in different diagrams have the same spatial positions.

to achieve Artificial General Intelligence, it needs to possess the same abstract reasoning capability as humans(e.g., higher-level planning). Measuring DNN-based system's performance on human IQ tests provides an indicator of how far today's state-of-the-art DNN architectures are from achieving a certain level of General Intelligence beyond perception tasks. It is also an indication of whether DNN, thought by many to just be advanced curve fitting [78], can actually learn chains of reasoning needed to solve many higher-level tasks.

The first attempt to solve complex RPM tasks with more than one object in diagrams was by Barrett et al [5]. In their work, they published a large and comprehensive RPM-style dataset named Procedurally Generated Matrices 'PGM', and proposed the Wild Relation Network (WReN), a state-of-the-art neural network for RPM-style tasks. While WReN outperforms other state-of-the-art vision models such as Residual Network [39], the performance is still far from deep neural networks' performance on other vision or natural language processing tasks. Recently, there has been a focus on object-level representations ([114, 43, 47, 73, 96, 119]) for visual reasoning tasks, which enable the use of inductive-biased architectures such as symbolic programs and scene graphs to directly capture relations between objects. Here relations denotes a similarity metric between attributes of a pair of objects. For RPM-style tasks, symbolic programs are less suitable as these programs are generated from given questions in the Visual-Question Answering setting (discussed in Section 1.1). In RPM-style tasks there are no explicit questions. Encoding RPM tasks into graphs is a more natural choice. However, previous works on scene graphs ([96, 119]) model a single image as graphs, which is not suitable for RPM tasks as there are many different layers of relations across different subsets of diagrams in a single task.

In this chapter, I introduce MXGNet, a multi-layer multiplex graph neural net architecture for abstract diagram reasoning. Figure 4.1a shows an overview of MXGNet architecture. Here "multi-layer" means the graphs are built across different diagram panels, where each diagram is a layer. 'Multiplex' means that the edges of the graphs are composed of sub-edges that encode multiple relations between different element attributes, such as colour, shape, and position. Multiplex networks are discussed in details in [55]. For RPM tasks, MXGNet encodes subsets of diagram panels into multi-layer multiplex graphs, and combines summarisation of several graphs to predict the correct candidate answer. With a hierarchical summarisation scheme, each graph is summarised into feature embeddings representing relationships in the subset. These relation embeddings are then combined to predict the correct answer.

For the PGM dataset [5], MXGNet outperforms WReN, the previous state-of-the-art model, by a considerable margin. For "neutral" data split, in which training and test datasets are sampled **i.i.d**, MXGNet achieves 89.6% test accuracy, 12.7% higher than WReN's 76.9%. For other splits where training and test data distributions differ, MXGNet consistently performs better, but with smaller margins. For the RAVEN dataset ([120]), MXGNet, without any auxiliary training with additional labels, achieves 83.91% test accuracy, outperforming the 59.56% accuracy by the best model with auxiliary training for the RAVEN dataset. I also show that MXGNet is robust to the variations in forms of

object-level representations. Both variants of MXGNet achieve higher test accuracies than existing best models for the two datasets.

Next, I will describe MXGNet in detail. In Section 4.2 I will present experimental results of MXGNet on the PGM and RAVEN datasets.

4.1 MXGNet Architecture

MXGNet is comprised of three main components: an object-level representation module, a graph processing module, and a reasoning module. Figure 4.1a shows an overview of the MXGNet architecture. The object-level representation module F_ρ, as the name suggests, extracts representations of objects in the diagrams as nodes in a graph. Recall that, as introduced in Section 2.2.2, RPM tasks consist of context diagrams $\mathbf{C}$ and answer candidate diagrams $\mathbf{A}$. For each diagram $\boldsymbol{d}_i \subset \mathbf{C} \cup \mathbf{A}$, a set of nodes $\boldsymbol{v}_{i,j}; i = 1 \dots L, j = 1 \dots N$ is extracted where L is the number of layers and N is the number of nodes per layer. I experimented with both fixed and dynamically learned N values. I also experimented with an additional 'background' encoder that encodes background lines (examples in Figure 2.4) into a single vector, which can be considered as a single node. The multiplex graph module G_ϕ, for a subset of diagrams, learns the multiplex edges capturing multiple parallel relations between nodes in a multi-layer graph where each layer corresponds to one diagram in the subset, as illustrated in Figure 4.1b. In MXGNet, I consider a subset of cardinality 3 for 3×3 diagram matrices. While prior knowledge of RPM rules existing in rows and/or columns of the matrix allows one to naturally treat rows and columns in RPM as subsets, this prior knowledge does not generalise to other types of visual reasoning problems. Considering all possible diagram combinations as subsets is computationally expensive. To tackle this, I developed a relatively quick pre-training method to greatly reduce the search space of subsets, as described below.

Search Space Reduction: I can consider each diagram as node $\boldsymbol{v}_i^d$ in a graph, where relations between adjacent diagrams are embedded as edges $\boldsymbol{e}_{ij}^d$. Note here I am considering the graph of 'diagrams', which is different from the graph of 'objects' in the graph processing modules. Each subset of 3 diagrams in this case can be considered as a subset of 2 edges. I here make weak assumptions that edges exist between adjacent diagrams (including vertical, horizontal, and diagonal directions) and edges in the same subset must be adjacent (defined as two edges linking the same node), which are often used in other visual reasoning problems. Subsets of edges are denoted as $\{\boldsymbol{e}_{ij}^d, \boldsymbol{e}_{jk}^d\}$. 3 neural nets are used to embed nodes, edges, and subsets. CNNs are used to embed diagram nodes into feature vectors, and MLPs to embed edges based on node embeddings as well as

subsets based on edge embeddings. While it is possible to include graph architectures for better accuracy, I found that simple combinations of CNNs and MLPs train faster while still achieving the search space reduction results. This architecture first embeds nodes, then embeds edges based on node embedding, and finally embeds subsets based on edge embedding. The subset embeddings are summed and passed through a reasoning network to predict answer probability, similar to WReN ([5]). For the exact configuration of the architecture used, refer to Appendix B.1. For each subset$\{\boldsymbol{e}^d_{ij}, \boldsymbol{e}^d_{jk}\}$, I define a gating variable g_{ijk}, which controls how much each subset contributes to the final result. In practice, I use a *tanh* function, which allows a subset to contribute both positively and negatively to the final summed embeddings. In training, an L1 regularisation constraint is used on the gating variables to suppress g_{ijk} of non-contributing subsets close to zero. This architecture can quickly discover rows and columns as contributing subsets, while leaving gating variables of other subsets not activated. The experiment results are described in Section 4.2.1. While this method was developed for discovering reasoning rules for RPM tasks, it can be readily applied to any other multi-frame reasoning task for search space reduction. In the rest of this Chapter, I hard-gate subsets by rounding the gating variables, thereby reducing subset space to only treat rows and columns as valid subsets.

The first 2 rows and columns are treated as contextual subsets $\mathbb{C}^{row}_i$ and $\mathbb{C}^{col}_i$ where i are row or column indices. For the last row and column, where the answers should be filled in, each of the 8 answer candidates are filled in to make 8 row subsets $\mathbb{A}^{row}_i; i \in 1, \ldots, 8$ and 8 column subsets $\mathbb{A}^{col}_i; i \in 1, \ldots, 8$.

The graph module then summarises the graph of objects in a subset into embeddings representing relations present in the subset. The reasoning module R_θ takes embeddings from context rows/columns and last rows/columns with different candidate answers filled in, and produces normalised probability of each answer being true. It also predicts meta-target for auxiliary training using context rows/columns. Next, I describe each module in detail.

4.1.1 Object-Level Representation

In the PGM dataset, there are two types of objects, namely "shapes" and "background lines". While it is a natural choice to use object-level representation on shapes as they vary in many attributes such as position and size, it is less efficient on background lines, since they only vary in colour intensity. In this section, I first describe the object-level representation applied to 'shape' objects, and then discuss the object-level representation of 'lines' and an alternative background encoder that performs better.

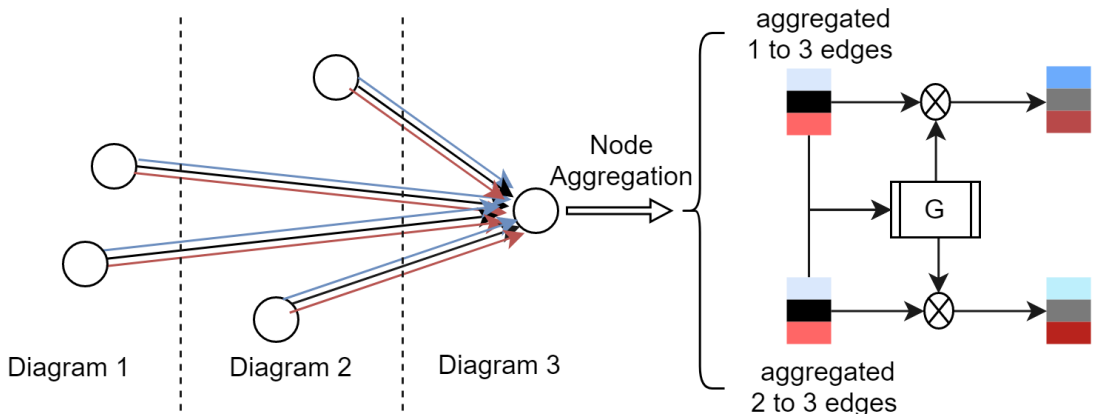


Figure 4.2: Illustration of multiplex edge embeddings and cross-gating function. Each edge contains a set of different sub-connections (coloured differently). Multiplex edges connecting nodes in previous layers to nodes in the last layer are aggregated according to their originating layers. We denote the aggregated feature vectors from these nodes as aggregated embedding. Aggregated embeddings are then passed to a gating function G, which outputs gating variables that regulate information coming from each part of the aggregated embeddings.

In MXGNet, I experiment with two types of object-level representations for 'shapes', namely CNN grid features and representation obtained with spatial attention. For CNN grid features, each spatial location in the final CNN feature map is used as the object feature vector. Thus, for each feature map of width W and height H, $N = W \times H$ object representations are extracted. This type of representation is used widely, such as in Relation Networks [85] and VQ-VAE [100]. For representations obtained with attention, spatial attention is used to attend to locations of objects, and extract representations for each object attended. This is similar to objection detection models such as faster R-CNN [82], which use a Region Proposal Network to propose bounding boxes of objects in the input image. For each attended location, a presence variable z_{pres} is predicted by the attention module indicating whether an object exists in the location. Thus the total number of objects N can vary depending on the sum of z_{pres} variables. Since object-level representation is not the main innovation of this work, I leave exact details for Appendix B.1.1.

For background 'line' objects, which do not vary in position and size, spatial attention is not needed. I experimented with a recurrent encoder with Long-Short Term Memory [42] on the output feature map of CNN, outputting M number of feature vectors. However, in the experiment, I found that this performs less well than just feature map embeddings produced by the feed-forward conv-net encoder.

4.1.2 Multiplex Graph Network

Multiplex Edge Embedding: The object-level representation module outputs a set of representations $\{\boldsymbol{v}_{i,j}; i \in 1,\ldots,L, j \in 1,\ldots,N\}$ for 'shape' objects, where L is the number of layers (cardinality of a subset of diagrams) and N is the number of nodes per layer. MXGNet uses a multiplex edge-embedding network E_γ to generate edge embeddings encoding multiple parallel relation embeddings:

$$\boldsymbol{e}^t_{(i,j),(l,k)} = E^t_\gamma(P^t(\boldsymbol{v}_{i,j}, \boldsymbol{v}_{l,k})); i \neq l, t = 1 \ldots T \tag{4.1}$$

Here P^t is a projection layer projecting concatenated node embeddings to T different embeddings. E^t is a small neural network processing t^{th} projection to produce the t^{th} sub-layer of edge embeddings. Here, edges are restricted to be inter-layer only, as intra-layer edges are found to have no effect on improving performance but increase computational costs. Figure 4.2 illustrates these multiplex edge embeddings between nodes of different layers. I hypothesise that different layers of the edge embeddings encode similarities/differences in different feature spaces. Such embeddings of similarities/differences are useful in comparing nodes for subsequent reasoning tasks. For example, for *Progessive* relation of object sizes, part of theembeddings encoding size differences can be utilised to check if nodes in later layers are larger in size. This is similar to the 'Mixture of Experts' layers [20, 90] introduced in Neural Machine Translation tasks. However, in this work, I developed a new cross-multiplexing gating function at the node message aggregation stage, which is described below.

Graph Summarisation: After edge embeddings are generated, the graph module then summarises the graph into a feature embedding representing relations present in the subset of diagrams. Information in the graph is aggregated to nodes of the last layer corresponding to the third diagram in a row or column because, in RPM tasks, the relations are in the form $Diagram3 = Function(Diagram1, Diagram2)$. All edges connecting nodes in a particular layer $v_{i,j}; i \neq L$, to a node $\boldsymbol{v}_{L,k}$ in the last layer L are aggregated by a function F_{ag} composed of four different types of set operations, namely max, min, sum, and $mean$:

$$\boldsymbol{q}_{i,k} = F_{ag}(\boldsymbol{e}_{(i,1),(L,k)} \ldots \boldsymbol{e}_{(i,N),(L,k)}); F_{ag}(x) = Concat(max(x), min(x), sum(x), mean(x)) \tag{4.2}$$

These multiple aggregation functions are used together because different sub-tasks in reasoning may require different types of summarisation. For example, counting the number of objects is better suited for sum, while checking if there is an object with the same size is better suited for max.

The aggregated node information from each layer is then combined with a cross-multiplexing gating function. It is named 'cross-multiplexing' because each embedding in the set is 'multiplexing' other embeddings in the set with gating variables that regulate which stream of information passes through. This gating function accepts a set of summarised node embeddings $\{\boldsymbol{q}_{1,k} \dots \boldsymbol{q}_{L-1,k}\}$ as input, and outputs gating variables for each layer of node embeddings in the set:

$$\mathbf{g}_{1,k} \dots \mathbf{g}_{L-1,k} = G(\boldsymbol{q}_{1,k} \dots \boldsymbol{q}_{L-1,k}); \mathbf{g}_{i,k} = \{\boldsymbol{g}_{i,k}^1 \dots \boldsymbol{g}_{i,k}^T\} \tag{4.3}$$

In practice, G is implemented as an MLP with multi-head outputs for different embeddings and Sigmoid activation, which constrains the gating variable g within the range of 0 to 1. The node embeddings of different layers are then multiplied with the gating variables, concatenated and passed through a small MLP to produce the final node embeddings: $\boldsymbol{q}_k = MLP(concat(\{\boldsymbol{q}_{i,k} \times \mathbf{g}_{i,k} | i = 1 \dots L-1\}))$. Node embeddings and background embeddings are then concatenated and processed by a residual neural block to produce the final relation feature embedding $\boldsymbol{r}$ of the diagram subset.

4.1.3 Reasoning network

The reasoning network takes relation feature embeddings $\boldsymbol{r}$ from all graphs and infers the correct answer based on these relation embeddings. I denote the relation embeddings for context rows as $\boldsymbol{r}_i^{cr}; i = 1, 2$ and context columns as $\boldsymbol{r}_i^{cc}; i = 1, 2$. The last row and column filled with each answer candidate $\boldsymbol{a}_i$ are denoted $\boldsymbol{r}_i^{ar}; i = 1, \dots, 8$ and $\boldsymbol{r}_i^{ac}; i = 1, \dots, 8$. For the RAVEN dataset, only row relation embeddings $\boldsymbol{r}^{cr}$ and $\boldsymbol{r}^{ar}$ are used, as discussed in Section 2.2.2. The reasoning network R_θ is a multi-layer residual neural net with a Softmax output activation that processes concatenated relation embeddings and outputs class probabilities for each answer candidate.

Figure 4.3 shows the reasoning network configuration for RPM tasks. I experimented with the approach introduced in [5], which computes scores for each answer candidate and finally normalises the scores. I found this approach leads to severe overfitting on the RAVEN dataset, and therefore used a simpler approach to just concatenate all relation embeddings and process them with a neural net. In practice, I used two residual blocks of size 128 and 256, and a final, fully connected layer with 8 units corresponding to 8 answer candidates. The output is normalised with a Softmax layer. Meta-targets (encoding meta-information of the tasks such as types of objects and relations present in the task) are also used as auxiliary labels in some experiments. In PGM datasets, there are 12 different meta-targets, while in RAVEN there are 9 (details in Section 2.2.2). For meta-target prediction, all relation information is contained in the context rows and columns of the

RPM task. Therefore, I apply a meta-predicting network R_{meta} with Sigmoid output activation to all context rows and columns to obtain probabilities of each meta-target category:

$$p_{meta} = R_{meta}(\boldsymbol{r}_1^{cr} + \boldsymbol{r}_2^{cr} + \boldsymbol{r}_1^{cc} + \boldsymbol{r}_2^{cc}) \tag{4.4}$$

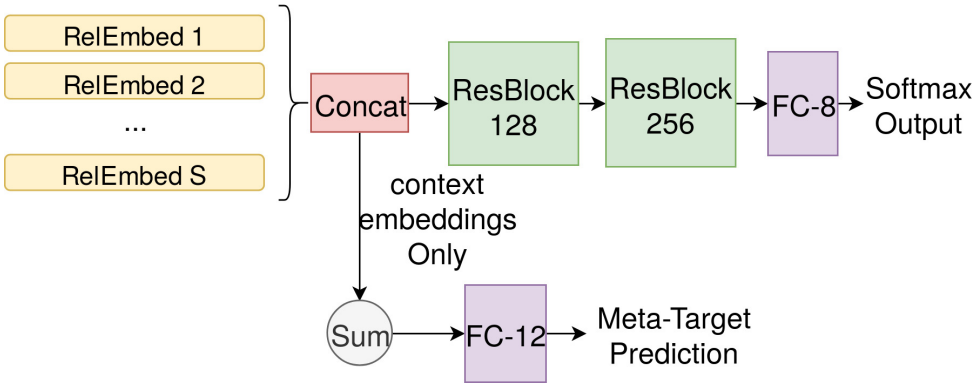


Figure 4.3: The architecture overview of the reasoning module. 'RelEmbed' are relation embeddings, 'Concat' is the concatenation layer. 'ResBlock' is a Residual Convolutional Block. 'FC' is a fully connected layer.

4.1.4 Training

The full pipeline of MXGNet is end-to-end trainable with any gradient descent optimiser. In practice, I used the RAdam optimiser [68] for its fast convergence and robustness to learning rate differences. The loss function for the PGM dataset is the same as used in WReN [5]: $\mathcal{L} = \mathcal{L}_{ans} + \beta\mathcal{L}_{meta-target}$ where β balances the training between answer prediction and meta-target prediction. For the RAVEN dataset, while the loss function can include auxiliary meta-target and structured labels as $\mathcal{L} = \mathcal{L}_{ans} + \alpha\mathcal{L}_{struct} + \beta\mathcal{L}_{meta-target}$, I found that both auxiliary targets do not improve performance, and thus set α and β to 0.

4.2 Experiments

4.2.1 Search Space Reduction

The Search Space Reduction model is applied on both the PGM and RAVEN datasets to reduce the subset space. The gating variable, with a possible range of $[0, 1]$, shows the contribution of subsets to the final prediction. After 10 epochs, only gating variables of rows and columns subsets for PGM and of rows for RAVEN have absolute values larger

than 0.5. The gating variables for the three rows are 0.884, 0.812 and 0.832. The gating variables for the three columns are 0.901, 0.845 and 0.854. All other gating variables are below the threshold value of 0.5. Interestingly all activated (absolute value > 0.5) gating variables are positive. This is possibly because it is easier for the neural net to learn an aggregation function than a comparator function. Exact experiment statistics can be found in Appendix B.3.

4.2.2 RPM task performances

In this section, I compare all variants of MXGNet against the state-of-the-art models (SOTA results correct at the time of paper submission). for the PGM and the RAVEN datasets. For the PGM dataset, MXGNet is tested against results of WReN [5] in the auxiliary training setting with β value of 10. In addition, MXGNet is also compared to VAE-WReN [94]'s result without auxiliary training. For the RAVEN dataset, MXGNet is compared to the WReN and ResNet model's performance as reported in the original paper [120]. I evaluated MXGNet with different object-level representations (Section 4.1.1) on the test data in the 'neutral' split of the PGM dataset.

Table 4.1a shows test accuracies of model variants compared with WReN and VAE-WReN for the case without auxiliary training ($\beta = 0$) and with auxiliary training ($\beta = 10$) for the PGM dataset. Both model variants of MXGNet outperform other models by a considerable margin, showing that the multi-layer graph is indeed a more suitable way to capture relations in the reasoning task. Model variants using grid features from the CNN feature maps slightly outperform models using spatial-attention-based object representations, both with and without auxiliary training settings. This is possibly because the increased number of parameters for the spatial attention variant leads to over-fitting, as the training losses of both model variants are very close. In the subsequent experiments for PGM, I use model variants with CNN grid features to report performances.

Table 4.1b shows test accuracies of model variants compared with WReN, which is the best performing ResNet model for the RAVEN dataset. WReN surprisingly only achieves 14.69% accuracy as tested by [120]. I include results of the ResNet model with and without Dynamic Residual Trees [120], which utilise additional structure labels of relations. I found that, for the RAVEN dataset, auxiliary training of MXGNet with meta-target or structure labels (see Section 4.1.4) does not improve performance. Therefore, I report test accuracies of models trained only with the target-prediction objective. Both variants of MXGNet significantly outperform the ResNet models. Models with spatial attention object-level representations under-perform simpler CNN features slightly, most probably due to overfitting, as the observed training losses of spatial attention models are lower

than CNN feature models. It is notable that MXGNet performance is very close to human performance, with an only 0.5% lower accuracy.

Model	WReN [5]	VAE-WReN [94]	ARNe [37]	MXGNet CNN	MXGNet Sp-Attn
acc. (%) $\beta = 10$	76.9	N/A	88.2	**89.6**	88.8
acc. (%) $\beta = 0$	62.6	64.2	N/A	**66.7**	66.1

(a) PGM

Model	WReN [120]	ResNet [120]	ResNet+DRT [120]	MXGNet CNN	MXGNet Sp-Attn	Human [120]
acc. (%)	14.69	53.43	59.56	**83.91**	82.61	84.41

(b) RAVEN

Table 4.1: (a) shows results comparing MXGNet model variants against WReN for the PGM dataset. ARNe's result with $\beta = 0$ is N/A because authors did not report this result and there is no avaiable code for reproduction. (b) shows results comparing MXGNet model variants against ResNet models for the RAVEN dataset. The object-level representation has two variations which are (o1) CNN features and (o2) Spatial Attention features (Section 4.1.1).

4.2.3 Generalisation evaluation for PGM

In the PGM dataset, other than the neutral data regime in which the test dataset's sampling space is the same as the training dataset, there are also other data regimes that restrict the sampling space of training or test data to evaluate the generalisation capability of a neural network. Table 4.2 shows validation and test accuracies of MXGNet for all data regimes with and without auxiliary training. For the 'interpolation' regime, in the training dataset, when attribute $a = \textit{colour}$ and $a = \textit{size}$, the values of a are restricted to even-indexed values in the spectrum of a values. This tests how well a model can 'interpolate' for missing values. For the 'Extrapolation' regime, in the training dataset, the value of a is restricted to the lower half of the value spectrum. This tests how well a model can 'extrapolate' outside of the value range in the training dataset. Other regimes with "H.O." in their names are data splits where particular relation tuples are held out during training. For example, in "H.O. `shape-color`" split, the training dataset does not contain any relations in colours of the shape objects, while the test dataset contains only relations in colours of the shape objects. For the exact details please refer to [5].

In addition, differences between validation and test accuracies are also presented to show how well models can generalise. MXGNet models consistently perform better than WReN for all regimes tested. MXGNet outperforms WReN on all data splits except the H.O. `shape-color` data split with no auxiliary training (i.e., $\beta = 0$). These results show

that MXGNet has a better capability of generalising outside of the training space.

Model	Regime	$\beta = 0$			$\beta = 10$		
		Val.(%)	**test%**	**Diff.**	**Val.(%)**	**test%**	**Diff.**
WReN	Neutral	63.0	62.6	-0.4	77.2	76.9	-0.3
	Interpolation	79.0	64.4	-14.6	92.3	67.4	-24.9
	Extrapolation	69.3	17.2	-52.1	93.6	15.5	-79.1
	H.O. Attribute Pairs	46.7	27.2	-19.5	73.4	51.7	-21.7
	H.O. Triple Pairs	63.9	41.9	-22.0	74.5	56.3	-18.2
	H.O. Triples	63.4	19.0	-44.4	80.0	20.1	-59.9
	H.O. `line-type`	59.5	14.4	-45.1	78.1	16.4	-61.7
	H.O. `shape-color`	69.3	**17.2**	-52.1	93.6	15.5	-78.1
MXGNet	Neutral	67.1	**66.7**	-0.4	89.9	**89.6**	-0.3
	Interpolation	74.2	**65.4**	-8.8	91.5	**84.6**	-6.9
	Extrapolation	69.1	**18.9**	-50.2	94.3	**18.4**	-75.9
	H.O. Attribute Pairs	68.3	**33.6**	-34.7	81.9	**69.3**	-12.6
	H.O. Triple Pairs	67.1	**43.3**	-23.8	78.1	**64.2**	-13.9
	H.O. Triples	63.7	**19.9**	-43.8	80.5	**20.2**	-60.3
	H.O. `line-type`	60.1	**16.7**	-43.4	85.2	**16.8**	-61.5
	H.O. `shape-color`	68.5	16.6	-51.9	89.2	**15.6**	-73.6

Table 4.2: Generalisation performance comparing MXGNet model variants against WReN. **'Diff.'** is the difference between the test and the validation performances.

4.2.4 Ablation study

I performed ablation study experiments to test how much the multiplex edges affect performance. I tested two model variants, one without any graph modules, and the other with graphs using vanilla edge embeddings produced by MLPs, on the PGM dataset. I found that, without graph modules, the model only achieved 83.2% test accuracy. While this is lower than MXGNet's 89.6%, it is still higher than WReN's 76.9%. This is possibly because the search space reduction, by trimming away non-contributing subsets, allows the model to learn more efficiently. The graph model with vanilla edge embeddings achieves 88.3% accuracy, only slightly lower than MXGNet with multiplex edge embeddings. This shows that, while a general graph neural network is a suitable model for capturing relations between objects, the multiplex edge embedding does so more efficiently by allowing parallel relation multiplexing.

Additionally, I performed an ablation study on the reasoning network. As shown in Figure 4.3, the original reasoning module consists of two residual blocks with a final output layer. Reducing the number of residual blocks to 1 decreases the PGM neutral split accuracy from 89.6% to 85.3%. Increasing the number of residual blocks by 1 does not increase the accuracy; thus it is not considered due to increased computational cost.

4.3 Discussion

I presented MXGNet, a new graph-based approach to diagrammatic reasoning problems in the style of Raven Progressive Matrices (RPM). MXGNet combines three powerful ideas, namely, object-level representation, graph neural networks and multiplex graphs, to capture relations present in the reasoning task.
Pros of MXGNet include:

- Experiments show that MXGNet performs better than previous models on two RPM datasets. The results validate the research aim of this dissertation to prove that neural diagrammatic reasoning systems can learn complex diagrammatic reasoning tasks and achieve human-level performances.
- MXGNet can deal with input diagrams with an arbitrary number of entities (within a limit)
- MXGNet generalises better than previous methods for out-of-distribution dataset.

Cons of MXGNet include:

- A large amount of labelled data are required for training.
- While it is shown that MXGNet has better generalisation performance, the performance on some data splits, such as "extrapolation", is still only slightly above random chance (12.5% as there are 8 answer candidates). To improve the generalisation performance of DNN on reasoning tasks, I developed a DNN module that can be integrated into many architectures for improving **o.o.d** generalisation performance on reasoning tasks. This module is discussed in Chapter 6.

One important direction for future work is to make MXGNet interpretable, and thereby extract logic rules from MXGNet. Currently, the learned representations in MXGNet are still entangled, providing little in the way of understanding its mechanism of reasoning. Rule extraction can provide people with a better understanding of the reasoning problem, and may allow neural networks to work seamlessly with more programmable traditional logic engines. While extracting rules may hurt performance (as discussed in Section 3.3.2) for **i.i.d** training and test datasets, it may actually improve performance on **o.o.d** data, as extracted rules may better capture the underlying data structure while removing spurious correlations in the training dataset.

While the multi-layer multiplex graph neural network is designed for RPM style reasoning tasks, it can be readily extended to other diagrammatic reasoning tasks where relations are present between multiple elements across different diagrams. One example

of a real-world application scenario is robots assembling parts of an object into a whole, such as building a LEGO model from LEGO blocks. MXGNet provides a suitable way of capturing relations between parts, such as ways of piecing and locking two parts together. I leave this to future works.

Chapter 5

Unsupervised Diagram Summarisation with Deep Generative Models

While EulerNet (Chapter 3) and MXGNet (Chapter 4) achieved good, human-level performance on diagrammatic reasoning tasks, they require a considerable amount of labelled data to achieve this performance. In contrast, humans can learn to reason well with Euler diagrams or Raven Progressive Matrices with only hundreds of examples [87, 120]. It is argued (e.g., by Zador et al [116]) that humans can learn from only a few examples because the brain has already done a large amount of unsupervised learning by seeing, exploring, and interacting with the world. For the majority of tasks done by both humans and machines, unlabelled data is more accessible and in a larger amount than labelled data. Many leading researchers such as Geoffrey Hinton [62] and Yann Lecun [104] argue that unsupervised learning has an important role in the future of AI in enabling ML systems to learn with much less data.

Diagrammatic reasoning and more broadly visual reasoning tasks can also benefit from unsupervised learning. Such tasks require identifying and learning a useful representation of elements in the visual input, which can be accomplished in an unsupervised way. Objects in the visual input can be conveniently encoded into a representation containing its category, attributes and spatial positions and orientations. For example, an object can be of category vehicle, with attributes such as red colour and 4 doors, and positioned at the bottom right of the scene in a specific orientation. Humans, when recognising objects or trying to draw them, are believed to have attentional templates [11] of different categories of objects in mind that are augmented by different attributes and selected spatially via attention.

Machine approaches to unsupervised representation learning often use generative models such as Variational Auto-Encoders (VAE) [57], which use an inference network to infer latent codes corresponding to the representation, and a generator network to reconstruct

data given the representation. Recurrent versions of VAE such as the Attend-Infer-Repeat (AIR) model by Eslami et al [21] have been developed to decompose a scene into multiple objects, where each is represented by latent code $\boldsymbol{z} = (\boldsymbol{z}_{what}, \boldsymbol{z}_{where}, \boldsymbol{z}_{pres})$. While this latent code disentangles spatial information $\boldsymbol{z}_{where}$ and object presence $\boldsymbol{z}_{pres}$, for most of the tasks, the object representation $\boldsymbol{z}_{what}$ is an entangled real-valued vector that is thus difficult to interpret. While AIR does propose the possibility of using discrete latent code as $\boldsymbol{z}_{what}$, it only experimented with the discrete code with specifically-designed graphics engine as decoder. In this chapter, I test the **hypothesis** of whether *if it is possible to learn good and interpretable neuro-symbolic representations of diagrams for downstream reasoning tasks without supervision.* Here, I propose Discrete-AIR, an end-to-end trainable autoencoder that structures latent representation $\boldsymbol{z}_{what}$ into $\boldsymbol{z}_{cat}$, representing the category of objects and $\boldsymbol{z}_{attr}$ representing attributes of objects. Figure 5.1 illustrates how a scene of different shapes can be separately identified as different categories with varying attributes. This decomposition is similar to InfoGAN by Chen et al. [14], which also decomposes representation into style and shape using a modified Generative Adversarial Network [30]. However, there are two main differences. Firstly, while InfoGAN uses a mutual information objective in addition to a GAN objective to encourage disentangled coding, Discrete-AIR only uses the Variational Lower Bound (ELBO) objectives (See Section 2.5.4) and encourages disentanglement through the inductive-bias structure of latent code. Secondly, while InfoGAN is only applied to images containing single objects, Discrete-AIR is developed for scenes with multiple objects.

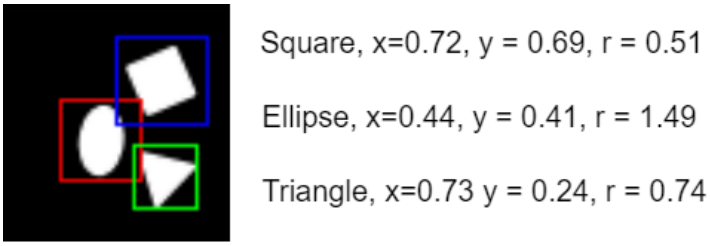


Figure 5.1: Illustration of encoding scenes into category and attribute latent code. From a scene containing three different shapes, Discrete-AIR separately identifies each of the shapes (with a different coloured bounding box). It also estimates spatial x-axis and y-axis locations, and the orientation of the shape.

The decomposition of latent codes into categorical variable $\boldsymbol{z}_{cat}$ and attribute variables $\boldsymbol{z}_{attr}$ has two immediate benefits. Firstly, the decomposition gives direct interpretability of the latent codes, as $\boldsymbol{z}_{cat}$ captures discrete latent variables like the categories of the objects (e.g., dog or cat) while $\boldsymbol{z}_{attr}$ captures continuous latent variables like colour and size. This disentanglement property is further illustrated in Section 5.3. Secondly, the decomposition generates semi-symbolic latent variables, which can be directly integrated with rule-based systems. A simple example is using an arithmetic operator to directly cal-

culate the sum of all numbers present in the image based on the digit category variables $\boldsymbol{z}_{cat}$.

Related to this work are other approaches that decompose scenes into different categories. Neural Expectation Maximisation (NEM) by Greff et al [33] implemented an Expectation-Maximization algorithm with an end-to-end trainable neural network. NEM is able to perceptually group pixels of an image into different clusters. However, it only learns a clustering model, rather than a generative model that allows controllable generation similar to using $\boldsymbol{z}_{cat}$ and $\boldsymbol{z}_{where}$ in Discrete-AIR. Ganin et al [25] train a neural network to synthesise programs that can be fed into a graphics engine to generate scenes. While it learns an inference model for the generative model, a graphics engine that can provide learning gradients is pre-defined and not learned. In contrast, Discrete-AIR jointly learns an inference model and a generative model from scratch.

In Section 5.3, I show that Discrete-AIR can decompose scenes into a set of disentangled and interpretable latent codes for two multi-object datasets, namely the Multi-MNIST dataset, as used in the original AIR model [21], and a multi-object dataset in a similar style as the dSprites dataset [74]. By disentangling discrete variables from continuous variables, I show that unsupervised training of the Discrete-AIR model is able to effectively capture the categories of objects in the scene for the Multi-MNIST and Multi-Sprites datasets. This can be viewed as a form of disentanglement achieved via enforcing structural constraints on the latent distribution.

5.1 Attend Infer Repeat

The Attend-Infer-Repeat (AIR) model, introduced by Eslami et al [21], is a recurrent version of Variational Auto-Encoder (VAE) [57], which decomposes a scene into multiple objects represented by the latent code $\boldsymbol{z}^i = (\boldsymbol{z}^i_{what}, \boldsymbol{z}^i_{where}, \boldsymbol{z}^i_{pres})$ at each recurrent time step i. Among them, $\boldsymbol{z}^i_{pres}$ is a binary discrete variable encoding whether an object is inferred in current step i. If $\boldsymbol{z}^i_{pres}$ is 0, the inference will be stopped. The sequence of $\boldsymbol{z}^i_{pres}$ for all i can be concatenated into a vector of n ones and a final zero. Therefore, n is a variable representing the number of objects in the scene. $\boldsymbol{z}^i_{where}$ is a spatial attention parameter used to locate a target object in the image, and $\boldsymbol{z}^i_{what}$ is the latent code of the target object. In AIR an amortised variational approximation $q_\phi(z|x)$, as computed in Equation 5.1, is used to approximate true posterior $p(\boldsymbol{z}|\boldsymbol{x})$ by minimising KL divergence $KL[q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})]$. In AIR implementation, $\boldsymbol{z}_{what}$ and $\boldsymbol{z}_{where}$ are parametrised as Gaussian distributions with

diagonal covariance $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

$$q_\phi(\boldsymbol{z}|\boldsymbol{x}) = q(\boldsymbol{z}^{n+1}_{pres}|\boldsymbol{z}^{1:n}, x) \prod_{i=1}^{n} q_\phi(\boldsymbol{z}^i, \boldsymbol{z}^i_{pres} = 1|\boldsymbol{x}, \boldsymbol{z}^{1:i-1}) \tag{5.1}$$

Where z^i is the latent code at i^{th} generation steps. In the generative model of AIR, the number of objects n can be sampled from a prior such as geometric prior, and then form the sequence of $\boldsymbol{z}^i_{pres}$. Next, $\boldsymbol{z}^i_{what}$ and $\boldsymbol{z}^i_{where}$ are sampled from $\mathcal{N}(0, \boldsymbol{I})$. An object $\boldsymbol{o}^i$ is generated by processing $\boldsymbol{z}^i_{what}$ through a decoder. $\boldsymbol{o}^i$ is then written to the canvas, gated by $\boldsymbol{z}^i_{pres}$, and with scaling and translation specified by $\boldsymbol{z}^i_{where}$ using Spatial Transformer [49], a powerful spatial attention module. Spatial Transformer uses attention mechanism to extract local regions from input images, or write to local regions in the canvas for image generation. The generative model can be summarised in Equation 5.2 and 5.3, where f_{dec} is the decoder, ST is the spatial transformer and $\odot$ is the element-wise product.

$$p_\theta(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{y}, \sigma_x \boldsymbol{I}) \tag{5.2}$$

$$\boldsymbol{y} = \sum_{i=1}^{n} ST(f_{dec}(\boldsymbol{z}^i_{what}), \boldsymbol{z}^i_{where}) \odot \boldsymbol{z}^i_{pres} \tag{5.3}$$

Inference and generative models of AIR are jointly optimised by maximising the lower bound $\mathcal{L}(q_\phi, p_\theta) = E_{q_\phi}[\log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z},n)}{q_\phi(\boldsymbol{z},n|\boldsymbol{x})}]$. While the sampling operation of $\boldsymbol{z}$ is not differentiable (which is a requirement for gradient-based training), there are various ways to circumvent this. For the continuous latent codes, the re-parametrisation trick for VAE [57] is applied, which lets parameters estimated from the inference model deterministically modify a sampled distribution, thereby allowing back-propagation through the deterministic function. For discrete latent codes, AIR uses the NVIL likelihood ratio estimator introduced by Mnih et al [77] to produce an unbiased estimate of the gradient for discrete latent variables.

5.2 Discrete-AIR

While the AIR model can encode objects in a scene into latent code z_{what}, it is still entangled and therefore not interpretable. In Discrete-AIR, I introduce structure into the latent distribution to encourage disentanglement. The latent variable z_{what} is decomposed into z_{cat} and z_{attr}, where z_{cat} is a discrete latent variable that captures the category of the object, while z_{attr} is a combination of continuous and discrete latent variables that capture attributes of the object. Note that no objective function or regularisation is used to encourage z_{cat} to capture category and z_{attr} to capture attributes. Rather, the model is allowed to automatically learn the best way of using these discrete and latent variables

through the process of likelihood maximisation.

5.2.1 Sampling discrete variable

In Discrete-AIR, binary discrete variables are treated as scalars of 0/1 values and multi-class categorical discrete variables are treated as one-hot vectors. As sampling from a discrete distribution is non-differentiable, I model discrete latent variables with Gumbel Softmax [70, 52], a continuous approximation to the discrete distribution from which an approximate one-hot discrete vector $\boldsymbol{y}$ can be sampled:

$$y_i = \frac{exp(\frac{log\, a_i + g_i}{\tau})}{\sum_{j=1}^{k} exp(\frac{log\, a_j + g_j}{\tau})} \quad ; i = 1, \ldots, k \tag{5.4}$$

here, a_i is a parametrisation of the distribution, g_i is Gumbel noise sampled from the Gumbel distribution $Gumbel(0, 1)$, and τ is a temperature parameter controlling smoothness of the distribution. The added Gumbel noise is important in making y_i a probabilistic distribution instead of a deterministic function. As $\tau \to 0$, the distribution converges to a discrete distribution. For binary discrete variables such as $\boldsymbol{z}_{pres}$, I use Gumbel Sigmoid, which is essentially Gumbel Softmax with the Softmax function replaced with the Sigmoid function:

$$y = \frac{exp(\frac{log\, a + g}{\tau})}{1 + exp(\frac{log\, a + g}{\tau})} \tag{5.5}$$

In contrast to the NVIL estimator [77] used in the original AIR model, I found that Gumbel Softmax/Sigmoid is more stable during training, experiencing no model collapse during all the training experiments.

5.2.2 Generative model

The probabilistic generative model is shown in Figure 5.2. From $\boldsymbol{z}_{cat}$, an high-level object template $T_{\boldsymbol{z}_{cat}}$ of this object category is generated. This template represents a prototype of the object category. This template is then modified by attributes $\boldsymbol{z}_{attr}$ into an image of object o that is subsequently drawn onto the canvas using spatial write attention. $\boldsymbol{z}^i_{cat}$, $\boldsymbol{z}^i_{attr}$, $\boldsymbol{z}^i_{where}$, and $\boldsymbol{z}^i_{pres}$, jointly as $\boldsymbol{z}^i$, are estimated from the inference model for each time step i of inference.

I replace the decoder function $f_{dec}(\boldsymbol{z}^i_{what})$ from the original AIR model with a new function $f_{dec}(\boldsymbol{z}^i_{cat}, \boldsymbol{z}^i_{attr})$ parametrised by the category variables $\boldsymbol{z}_{cat}$ and $\boldsymbol{z}_{attr}$. There are various candidate functions for combining $\boldsymbol{z}_{cat}$ and $\boldsymbol{z}_{attr}$. I have experimented with three different variations, which are:

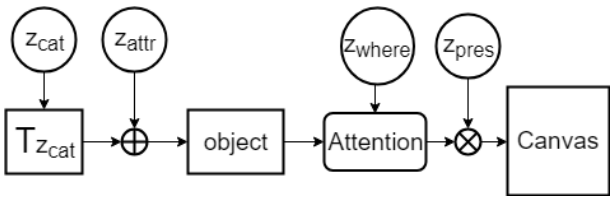


Figure 5.2: VAE decoder (Generative model) of Discrete-AIR.

- **Additive**: $f_{dec}(\boldsymbol{z}^i_{cat}, \boldsymbol{z}^i_{attr}) = f(f_t(\boldsymbol{z}^i_{cat}) + f_a(\boldsymbol{z}^i_{attr}))$ where $f_t(\boldsymbol{z}^i_{cat})$ generates a template, while $f_a(\boldsymbol{z}^i_{attr})$ generates an additive modification of template. Here f is implemented as transposed convolution networks for generating images. $+$ means element-wise addition of vectors.

- **Multiplicative**: $f_{dec}(\boldsymbol{z}^i_{cat}, \boldsymbol{z}^i_{attr}) = f(f_t(\boldsymbol{z}^i_{cat}) \odot f_a(\boldsymbol{z}^i_{attr}))$ where $f_t(\boldsymbol{z}^i_{cat})$ generates a template, while $f_a(\boldsymbol{z}^i_{attr})$ generates a multiplicative modification of template. $\odot$ means element-wise multiplication.

- **Convolutional**: $f_{dec}(\boldsymbol{z}^i_{cat}, \boldsymbol{z}^i_{attr}) = f(f_{t(conv)}(\boldsymbol{z}^i_{cat}) * f_{a(filter)}(\boldsymbol{z}^i_{attr}))$ where $f_{t(conv)}(\boldsymbol{z}^i_{cat})$ generates a template, while $f_{a(filter)}(\boldsymbol{z}^i_{attr})$ generates a set of convolution kernels that can be convolved with a template to modify it. Here $*$ means convolution operation.

In the experiments, I found that the choice of combining functions has only a small effect on the model performance. I found that the additive combining function performs slightly better, thus, I use this function in all of the presented experiments.

In the original AIR model, the spatial transformation operation specified by attention variable $\boldsymbol{z}_{where}$ only contains translation and scaling. Affine transformations such as rotation and shearing are accounted for in the latent variable in an entangled way. In Discrete-AIR, I explicitly introduce additional spatial transformer networks that account for rotation and skewing, thereby allowing $\boldsymbol{z}_{attr}$ to have a reduced number of variables. The spatial attention for the generative decoder is thus factorised as in Equation 5.6,

$$\boldsymbol{T}^d = \boldsymbol{T}^d_{st}\boldsymbol{T}^d_r\boldsymbol{T}^d_k = \tag{5.6}$$
$$\begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\omega) & sin(-\omega) & 0 \\ sin(\omega) & cos(\omega) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 + k_x k_y & k_x & 0 \\ k_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where $\boldsymbol{T}^d_{st}$ is the combined transformation matrix of translation and scaling used in the original AIR model, $\boldsymbol{T}_r$ is the transformation matrix for rotation and $\boldsymbol{T}_k$ is the transformation matrix for skewing. In the matrix, s_x and s_y are for scaling, t_x and t_y are horizontal and vertical translations, ω is an angle of rotation, and k_x and k_y are parameters

for shearing in the horizontal and vertical axes.

5.2.3 Inference

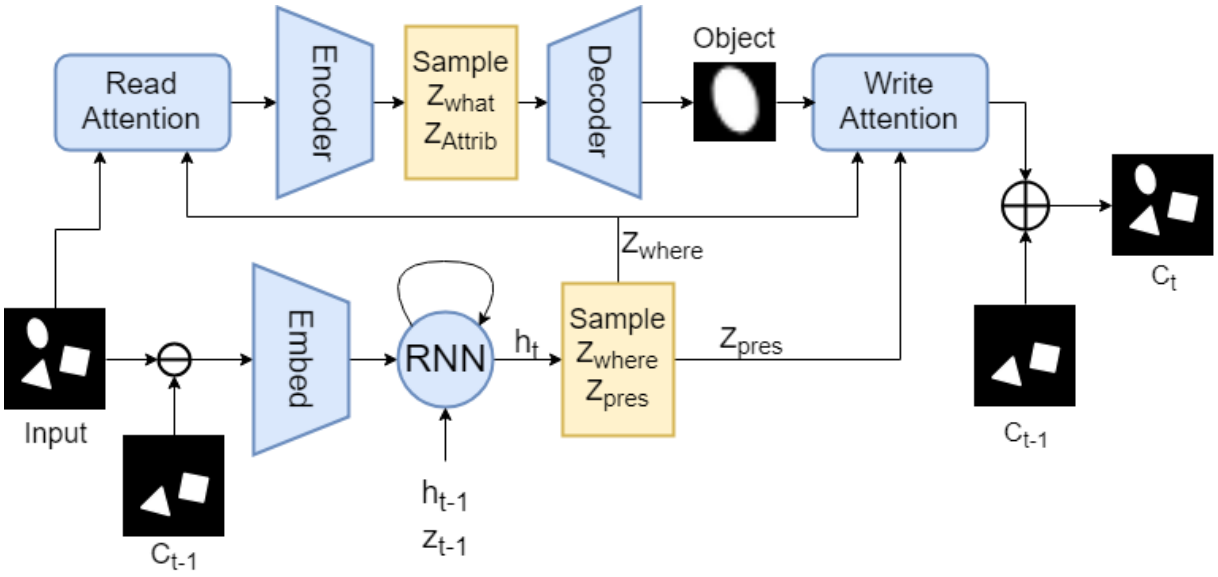


Figure 5.3: Overview of Discrete-AIR architecture. Blue parts are neural-network trainable modules and the yellow parts are sampling processes. Here, z_t is the latent code at time step t and h_t is the RNN hidden states at time step t. c_{t-1} is the canvas from the previous step that is compared with the input image for difference. c_t is the canvas generated at the current step.

Figure 5.3 shows an overview of the Discrete-AIR architecture. At inference step t, a difference image between input image D and previous canvas C_{t-1} is fed together with previous latent code $\boldsymbol{z}^{t-1}$ into a Recurrent Neural Network (RNN), implemented as Long Short-Term Memory (LSTM) [42] to generate parameters of the distribution for $\boldsymbol{z}_{where}$ and $\boldsymbol{z}_{pres}$. The Spatial Attention module then attends to parts of the image and applies transformation according to $\boldsymbol{z}_{where}$. I enforce the encoding transformation $\boldsymbol{T}^e$ to be the inverse of decoding transformation $\boldsymbol{T}^d$, which means $\boldsymbol{T}^e\boldsymbol{T}^d = \boldsymbol{I}$. This constraint forces the model to match attended objects in the scene with the invariant template specified by $\boldsymbol{z}_{cat}$. In practice, I compute $\boldsymbol{T}^e$ as the product of inverses of the transformation matrices composing $\boldsymbol{T}^d$:

$$\boldsymbol{T}^e = \boldsymbol{T}^{d^{-1}} = \boldsymbol{T}_k^{d^{-1}}\boldsymbol{T}_r^{d^{-1}}\boldsymbol{T}_{st}^{d^{-1}} \tag{5.7}$$

The transformed image is then processed by an encoder to estimate parameters of distributions for $\boldsymbol{z}_{cat}$ and $\boldsymbol{z}_{attr}$. $\boldsymbol{z}_{cat}$ are sampled from Gumbel Softmax as discussed in Section 5.2.1. $\boldsymbol{z}_{attr}$ can be sampled from any distribution that is suitable for the paradigm of tasks. For tasks presented, continuous variables such as the colour intensity or part deformation of an object can be sampled from a multi-variate Gaussian distribution using

the re-parameterisation trick [57], which allows a gradient to pass through the originally un-differentiable sampling function. The generative model described in Section 5.2.2 then samples $\boldsymbol{z}_{cat}$ and $\boldsymbol{z}_{attr}$ from the distributions in order to generate an object that will be written to canvas C_t using the spatial attention module.

5.2.4 Learning

Similar to the original AIR model, I train Discrete-AIR model end-to-end by maximising the lower bound on the marginal likelihood of data:

$$log\, p_\theta(x) \leq \mathcal{L}(\theta, \phi) = \mathbb{E}_{q_\phi}\Big[\log \frac{p_\theta(x, z, n)}{q_\phi(z, n|x)}\Big] \tag{5.8}$$

While in the original AIR model, one cannot further arrange this equation due to undifferentiable discrete variable sampling process used. For Discrete-AIR, by using Gumbel-Softmax as a repameterised sampling process, Equation 5.8 can be rearranged as:

$$\mathcal{L}(\theta, \phi) = \mathbb{E}_{q_\phi}\Big[\log p_\theta(\boldsymbol{x}|\boldsymbol{z}, n)\Big] - D_{KL}(q_\phi(\boldsymbol{z}, n|\boldsymbol{x})||p(\boldsymbol{z}, n)) \tag{5.9}$$

where $p_\theta(\boldsymbol{x}|\boldsymbol{z}, n)$ is data likelihood and D_{KL} is Kullback-Leibler (KL) divergence. This is the same as implemented in the original VAE [57]. Computing $\frac{\partial \mathcal{L}}{\partial \theta}$, the loss derivative with respect to parameters of the generative model, is relatively straightforward as it is fully differentiable. With a sampled batch of latent codes $\boldsymbol{z} = (\boldsymbol{z}_{cat}, \boldsymbol{z}_{attr}, \boldsymbol{z}_{where}, \boldsymbol{z}_{pres}) \sim q(\cdot|\boldsymbol{x})$, the partial derivative $\frac{\partial}{\partial \theta} p_\theta(]vx|\boldsymbol{z}, n)$ can be directly computed.

When computing $\frac{\partial \mathcal{L}}{\partial \phi}$, the re-parameterisation trick [57] can be used to re-parametrise the sampling of both, continuous and discrete latent variables as a deterministic function in the form $h(\omega^i, \epsilon^i)$. ω^i are the parameters of the distributions for $\boldsymbol{z}$ at time step i, and ϵ^i is random noise at time step i. In this way, the chain rule can be used to compute the gradient with respect to ϕ as:

$$\frac{\partial \mathcal{L}}{\partial \phi} = \frac{\partial \mathcal{L}}{\partial h} \times \frac{\partial h}{\partial \omega^i} \times \frac{\partial \omega^i}{\partial \phi} \tag{5.10}$$

For the experiments, I parametrise continuous variables as multivariate Gaussian distributions with a diagonal covariance matrix. Thus $h(\omega^i, \epsilon^i) = \mu^i + \sigma^i * \epsilon^i$. For discrete variables, I use Gumbel Softmax as introduced in Section 5.2.1, which is itself a re-parametrised differentiable sampling function. For the KL-divergence term, assuming all latent variables $\boldsymbol{z}$ are conditionally independent, I can factorise $q(\boldsymbol{z}|\boldsymbol{x})$ as $\prod_i q(\boldsymbol{z}_i|\boldsymbol{x})$ and thereby separate the KL-terms, as discussed in [18]. While this simplistic assumption may not capture the correlations between different latent dimensions, it has the effect of further encouraging disentanglement. I use Gaussian priors for all continuous variables. While KL divergence between two Gumbel-Softmax distributions is not available in closed form,

I approximate with a Monte-Carlo estimation of KL divergence with a categorical prior for $\boldsymbol{z}_{cat}$, similar to [52]. For $\boldsymbol{z}_{pres}$, I used a geometric prior and computed a Monte-Carlo estimation of KL divergence [70].

5.3 Evaluation

I evaluate Discrete-AIR on two multi-object datasets, namely the Multi-MNIST dataset as used in the original AIR model [21] and a multi-object shape dataset comprised of simple shapes similar to the dSprites dataset [74], a dataset of 2D shapes developed for studying disentangling latent representations. I perform experiments to show that Discrete-AIR, while retaining the original strength of the AIR model in discovering the number of objects in a scene, can additionally categorise each discovered object. In order to evaluate how accurately Discrete-AIR can categorise each object, I compute the correspondence rate between the best permutation of category assignments from the Discrete-AIR model and the true labels of the dataset.

To explain the metric used, I will first define a few notations. For each input image x_i, Discrete-AIR generates a corresponding category latent code $\boldsymbol{z}^i_{cat}$ and the presence variable $\boldsymbol{z}^i_{pres}$. From this I can form a set of predicted object categories $O^i = \{o^i_1, \ldots, o^i_n\}$ for n predicted objects, where o^i_k is the k^{th} object category. For each image there is also a set of true labels of existing objects $T^i = \{t^i_1, ..t^i_m\}$. Due to the non-identifiability problem of unsupervised learning where a simple permutation of a best cluster assignment produces the same optimal result, the category assignments produced by Discrete-AIR do not necessarily correspond to the labels. For example, an image patch of digit 1 could fall into Category 4. I thus permute the category assignments and use the permutation that corresponds best with the true label as the category assignment. For example, for the predicted category set $\{1, 4, 2, 2, 3\}$ and true label set $\{4, 0, 1, 1, 5\}$, the following permutation of category can be used for the predicted category set $(1 \to 4, 4 \to 0, 2 \to 1, 3 \to 5)$ to achieve best correspondence. To put it more formally, I define a function $f_p(C, p)$ where C is a set or array of sets, and p is an index permutation function to map elements in C. For the entire dataset, there is an array of predicted category set O and an array of true label set T. Correspondence rate is defined as $in(O, T)/size(T)$, where $in(O, T)$ gives the number of true labels t in T that are correctly identified in O. $size(T)$ gives the total number of labels. Thus the best correspondence rate is computed as:

$$R_{corr} = \max_{p \in P} \frac{in(O, T)}{size(T)} \tag{5.11}$$

where P is the set of all possible permutations of predicted categories. This score ranges from 0 to 1, and the score of a random category assignment should have the expected score of $1/k$ where k is the number of categories.

I train Discrete-AIR with the ELBO objective as presented in Equation 5.8. I use the Adam optimiser [56] to optimise the model with batch size of 64 and learning rate of 0.0001. For Gumbel Softmax, I also applied temperature annealing [52] of *tau* to start with a smoother distribution and gradually approximate to discrete distribution. For more details about training, please see Appendix C.1.

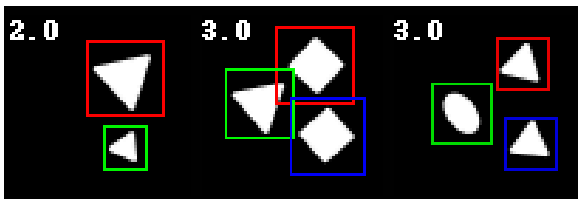
5.3.1 Multi-Sprites

To evaluate Discrete-AIR, I built a multi-object dataset named "Multi=Sprites" in a similar style to the dSprites dataset [74]. This dataset consists of 90000 images of pixel size 64×64. In each image there are 0 to 3 objects with shapes in the categories of square, triangle, and ellipse. The objects' spatial locations, orientations and sizes are all sampled randomly from uniform distributions. Details about constructing this dataset can be found in Appendix C.2. Figure 5.4 illustrates the application of Discrete-AIR on the Multi-Sprites dataset. Figure 5.4a shows samples of input data from the dataset with each object detected and categorised (with differently coloured bounding box). The number at the top-left corner shows the estimated number of objects in the scene. Figure 5.4b shows reconstructed images by the Discrete-AIR model. Figure 5.4c shows the fully interpretable latent code of each object in the scene. For this dataset, I used a discrete variable of 3 categories for $\boldsymbol{z}_{cat}$, together with spatial attention variables $\boldsymbol{z}_{where}$. I did not include $\boldsymbol{z}_{attr}$ for this dataset because the attributes of each object, including location, orientation, and size, can all be controlled by $\boldsymbol{z}_{where}$. I did not include shear transformation $\boldsymbol{T}_k^d$ in the spatial attention as the dataset generation process does not have a shear transformation. For more details about the architecture, please see Appendix C.1.

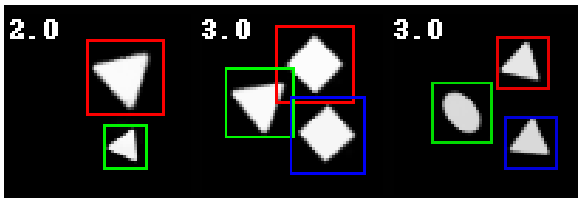
For quantitative evaluation of Discrete-AIR I use three metrics, namely, Reconstruction Error in the form of Mean Squared Error (MSE), count accuracy of the number of objects in the scene and categorical correspondence rate. I also compare Discrete-AIR with AIR for the first two metrics. Table 5.1 shows the performance for these three objectives. Mean performance across 10 independent runs is reported. Discrete-AIR has slightly better count accuracy than AIR, and is able to categorise objects with a mean category correspondence rate of 0.945. The best achieved correspondence rate is 0.967 Discrete-AIR does have increased reconstruction MSE compared to the AIR model. However Discrete-AIR only uses a category latent variable of dimension 3, while the original AIR model uses 50 latent variables.

I also plot count accuracy during training for both Discrete-AIR and AIR in Figure 5.5. One can observe that both models converge towards similar accuracies, but the Discrete-AIR model has slightly better convergence speed and stability at the start of training.

Discrete-AIR can generate a scene with a given number of objects in a fully controlled way. I can specify categories of objects with $\boldsymbol{z}_{cat}$ and their spatial attributes with $\boldsymbol{z}_{attr}$.



(a) Data



(b) Reconstruction

Triangle x = 0.71,y = 0.62 size = 0.31 angle = 0.66	Square x = 0.70,y = 0.69 size = 0.27 angle = 0.72	Triangle x = 0.73,y = 0.65 size = 0.23 angle = 2.31
Triangle x = 0.70,y = 0.23 size = 0.18 angle = -1.81	Triangle x = 0.32,y = 0.49 size = 0.29 angle = 0.71	Ellipse x = 0.41,y = 0.40 size = 0.25 angle = -0.34
	Square x = 0.73,y = 0.27 size = 0.27 angle = 0.74	Triangle x = 0.72,y = 0.24 size = 0.23 angle = 0.18

(c) Latent codes

Figure 5.4: Input data from the Multi-Sprites dataset and reconstruction from the Discrete-AIR model. The coloured bounding boxes show each detected object. The number at the top-left corner shows the count of the number of objects in the image. Latent codes representing the scene, including object categories, sizes, spatial locations, and orientation are also presented.

Model	MSE	count acc.	category corr.
Discrete-AIR	0.096	0.985	0.945
AIR	0.074	0.981	N/A

Table 5.1: Quantitative evaluation of Discrete-AIR and comparison with the AIR model for the Multi-Sprites dataset.

Figure 5.6 shows a sampled generation process. Note that while the training data contains up to 3 objects, Discrete-AIR can generate an arbitrary number of objects in the generative model. I generate 4 objects in the sequence "square, ellipse, square, triangle" with specified locations, orientation, and size.

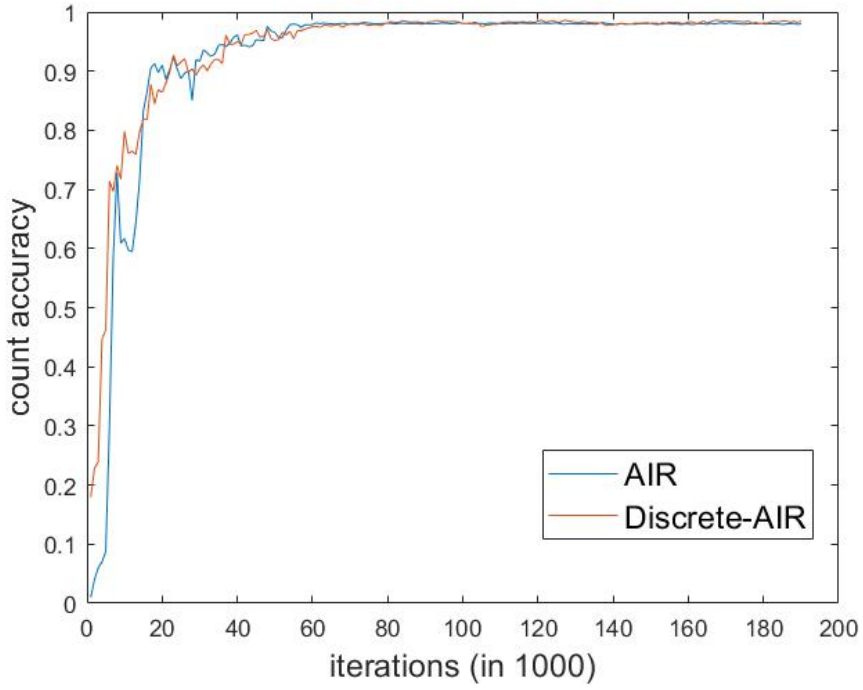


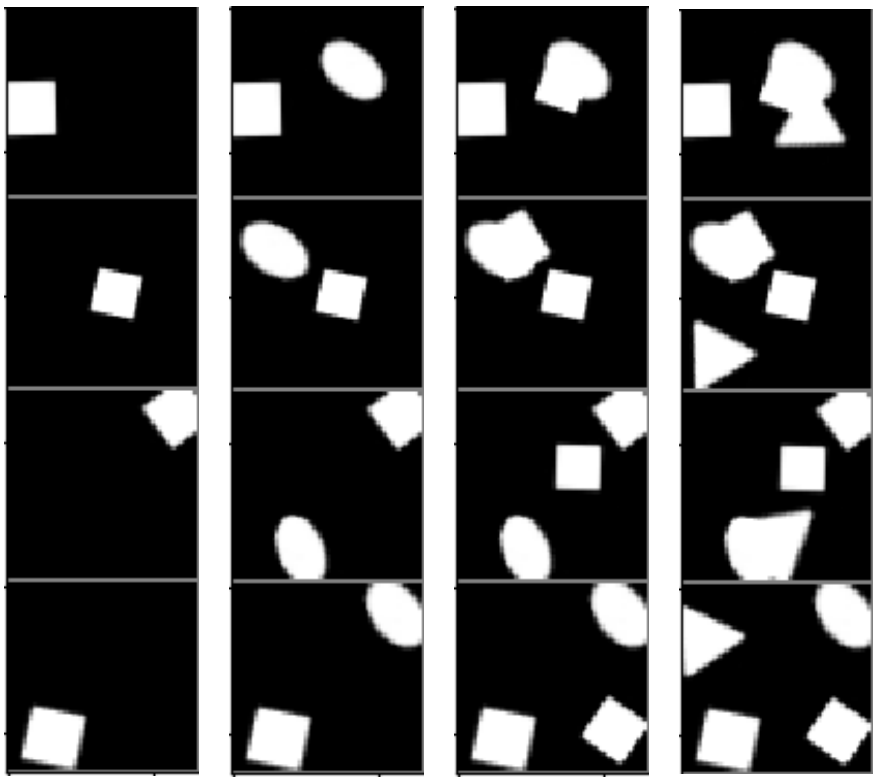
Figure 5.5: Plot of count accuracy for AIR and Discrete-AIR during training for Multi-Sprites dataset.

5.3.2 Multi-MNIST

I also evaluated Discrete-AIR on the Multi-MNIST dataset used by the original AIR model [21]. The dataset consists of 60000 images of size 50×50 pixels. Each image contains 0 to 2 digits sampled randomly from the MNIST dataset [65]. They are placed at random spatial positions. The dataset is publicly available in the 'observations' python package[1]. For this dataset, I choose a categorical variable with 10 categories as $\boldsymbol{z}_{cat}$ and 1 continuous variable with Normal distribution as $\boldsymbol{z}_{attr}$ because this gives the best category correspondence rate performance. I choose to combine transformation matrices T_r and T_k into one because this gives slightly better results. Figure 5.7 shows sampled input data from the dataset and reconstruction by Discrete-AIR. Figure 5.7c also shows interpretable latent codes for each digit in the image. From this figure, it can be clearly observed that Discrete-AIR learns to match templates of category $\boldsymbol{z}_{cat}$ with modifiable attributes $\boldsymbol{z}_{attr}$ to input data. For example, in the second image of input data, the digit '8' is written in a drastically different style from most other '8s' in the dataset. However, as it can be observed in the reconstruction, Discrete-AIR is able to match a template of digit '8' with modified attributes such as slantedness and stroke thickness.

I performed the same quantitative analysis from the Multi-Sprites dataset. Results are

[1] `https://github.com/edwardlib/observations`



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$ (d) $t = 3$

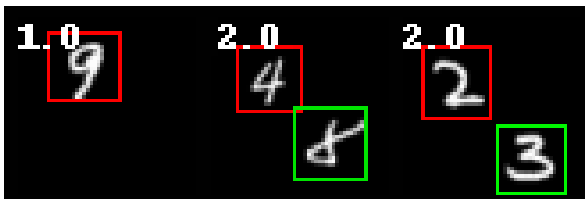
Figure 5.6: Generation of images of shapes by Discrete-AIR for the sequence "square, ellipse, square, triangle". Overlaps are due to randomised positioning for each generated object.

shown in Table 5.2. For correspondence rate measurements, I only use 10% subsampled data because permuting 10 digits requires 9! steps for evaluating across the dataset, which is too slow for the entire dataset. While the count accuracies of Discrete-AIR and the AIR model are very close, Discrete-AIR is able to categorise the digits in the image with a mean correspondence rate of 0.871. The best achieved correspondence rate is 0.913. I also plotted the count accuracy during training history, as shown in Figure 5.8. It can be observed that Discrete-AIR's increase rate of count accuracy is very close to that of the AIR model.

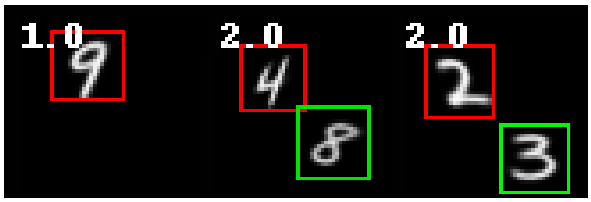
Model	MSE	count acc.	category corr.
Discrete-AIR	0.134	0.984	0.871
AIR	0.107	0.985	N/A

Table 5.2: Quantitative evaluation of Discrete-AIR and comparison with the AIR model for Multi-MNIST dataset.

Similar to the case of Multi-Sprites dataset, Discrete-AIR is able to generate images in a fully controlled way with given categories of digits $\boldsymbol{z}_{cat}$, attribute variable $\boldsymbol{z}_{attr}$ and



(a) Data



(b) Reconstruction

Digit 9
attr = 0.12
x = 0.38,y = 0.74
x_scale = 0.24
y_scale = 0.25
...

Digit 4
attr = -0.11
x = 0.42,y = 0.69
x_scale = 0.20
y_scale = 0.21
...

Digit 2
attr = -0.51
x = 0.41,y = 0.67
x_scale = 0.24
y_scale = 0.26
...

Digit 8
attr = 0.45
x = 0.72,y = 0.24
x_scale = 0.25
y_scale = 0.26
...

Digit 3
attr = 0.21
x = 0.76,y = 0.16
x_scale = 0.24
y_scale = 0.25
...

(c) Latent codes

Figure 5.7: Input data from the Multi-MNIST dataset and reconstruction from the Discrete-AIR model. The coloured bounding boxes show each detected object. The number at the top-left corner shows the count of the number of objects in the image. Latent codes representing the scene, including digit categories, attribute variable value, sizes, and spatial locations are also presented.

spatial variable $\boldsymbol{z}_{where}$. Figure 5.9 shows a sampled generated image. Two digits are generated in subsequent images with attribute variable increasing from top to bottom. In the first sequence, digits '5' and '2' are generated while in the second sequence digits '3' and '9' are generated. It can be seen that the learned attribute variable $\boldsymbol{z}_{attr}$ encodes attributes that cannot be encoded by affine transformation spatial variable $\boldsymbol{z}_{where}$. For example, increasing $\boldsymbol{z}_{attr}$ increases the size of the hook space in digit '5', the hook space in digit '2', and the hook curve in digit '9'.

5.4 Discrete-AIR for extracting interpretable scene graphs

Many real-world tasks, such as inferring physical relationships between objects in an image, and visual-spatial reasoning (e.g., inferring ways of piecing together two components in

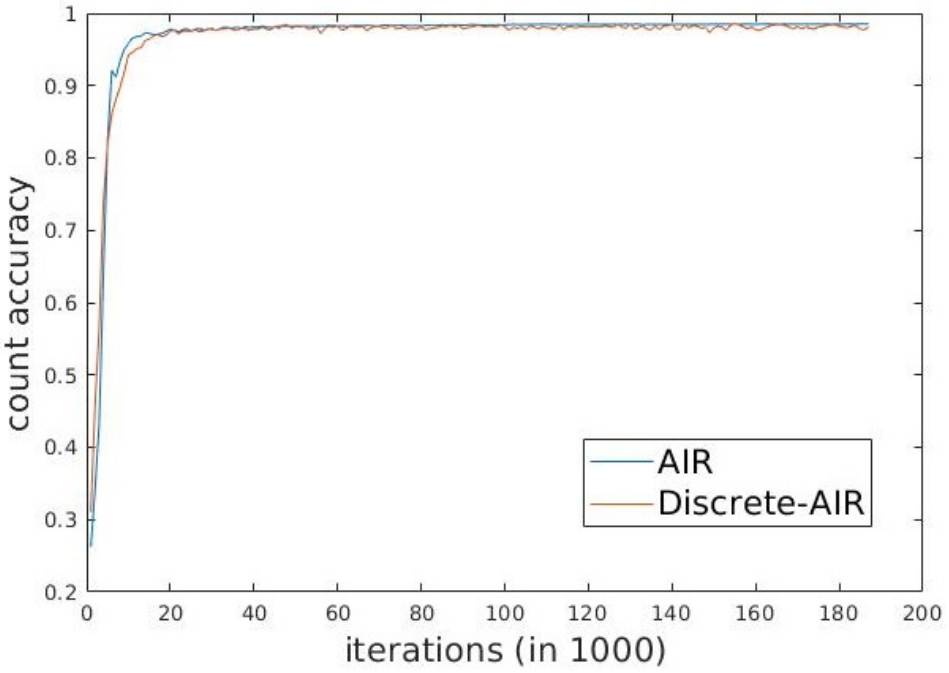
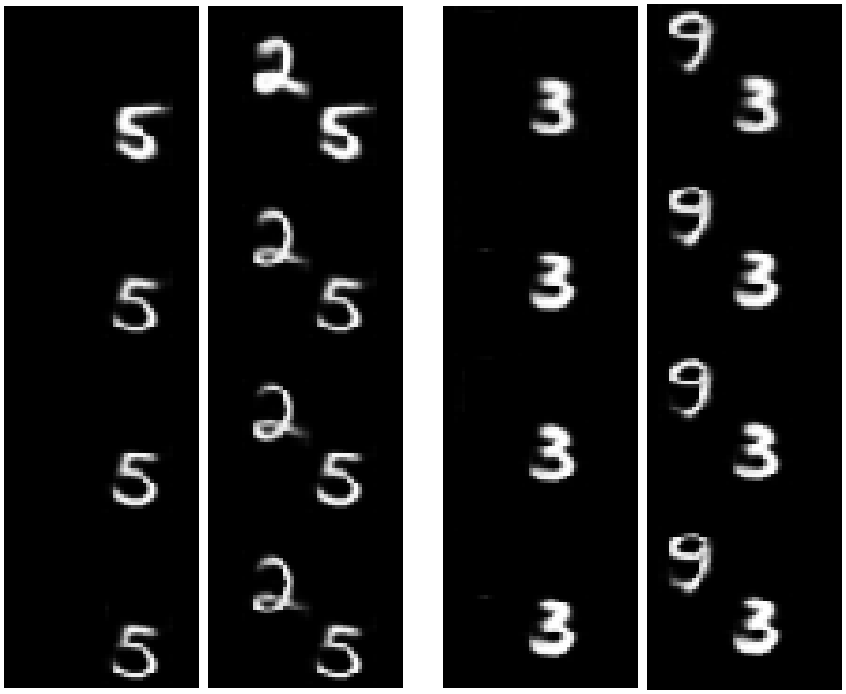


Figure 5.8: Plot of count accuracy for AIR and Discrete-AIR during training for the Multi-MNIST dataset.



(a) $t = 0$ (b) $t = 1$ (c) $t = 0$ (d) $t = 1$

Figure 5.9: Generation of Multi-MNIST images by Discrete-AIR for the sequence "5, 2" and "3, 9".

a furniture assembly task), require identifying and learning a useful representation of elements in the scene and representations of relations between elements. The elements

in the scene are essentially nodes in a graph and the relationships between elements are edges. While there are many proposed methods [60, 103, 7] in processing such graphs, there are few methods for extracting graph representations from raw input images. In this section I discuss how Discrete-AIR can be utilised for extracting scene graphs of input images with interpretable edge embedding initialisation.

The output of Discrete-AIR is a list of object-level embeddings $z = (z_{cat}, z_{attr}, z_{where}, z_{pres})$. A graph representation can be extracted by first instantiating nodes for embeddings with $z_{pres} = 1$ and subsequently instantiating embeddings of edges by computing distances between sub-variables $(z_{cat}, z_{attr}, z_{where}, z_{pres})$. For spatial variables z_{where} we can compute Euclidean L-2 distances with $D = (z^i_{where} - z^j_{where})^2$ with an edge direction indicator $d_{i,j}$. For object attributes z_{attr} one can use any distance metric in the continuous space. For z_{cat}, which is the categorical variable representing categories of objects, different distance metrics might be useful for different tasks. For example, the indicator function $\mathbf{1}(z^i_{cat}, z^i_{cat})$ maybe useful for logic reasoning. A customised distance matrix between each category pair may be useful for object query tasks such as visual question answering.

Figure 5.10a illustrates the application of Discrete-AIR for edge embedding initialisation on the Multi-Sprites dataset while Figure 5.10b illustrates interpretable node embedding for Multi-MNIST images. The left and middle figures show samples of input data from the dataset with each object detected and categorised (with differently coloured bounding boxes) and reconstructed images from the Discrete-AIR model. The number at the top-left corner shows the estimated number of objects in the scene. The right part illustrates interpretable edge and node embeddings between each object in the scene.

5.5 Discussion

In this chapter, I developed Discrete-AIR, an unsupervised auto-encoder that learns to model a scene of multiple objects with interpretable latent codes.
Pros of Discrete-AIR include:

- It is shown that Discrete-AIR can capture categories of each object in the scene and disentangle attribute variables from the categorical variable. This direct interpretability allows users of AI systems to directly understand how the DNN perceives and reasons with the input data. Such interpretability and explainability are important when DNN-based systems are applied on safety-critical tasks, such as autonomous driving or robotic surgery.

- The neuro-symbolic latent representation can be used directly by rule-based systems for subsequent processing. Thus Discrete-AIR provides a potential way of

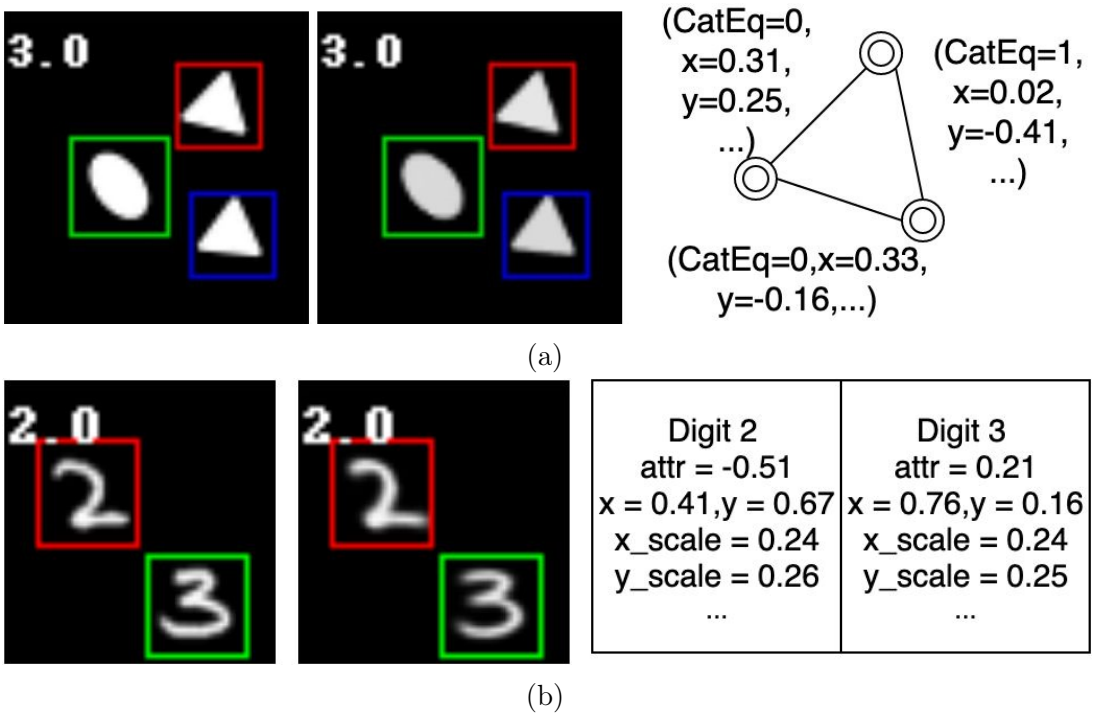


Figure 5.10: Input data, reconstruction, and learned latent codes from the Discrete-AIR model for the (a) Multi-Sprites and (b) Multi-MNIST datasets. The coloured bounding boxes show each detected object. The number at the top-left corner shows the count of the number of objects in the image. (a) shows interpretable edge embeddings. 'CatEq' indicates equality in category. (b) shows node embeddings. 'attr' shows z_{attr} value.

amalgamating neural perception front-ends with a rule-based logic backend.

Cons of Discrete-AIR include:

- The dimensions of the factorised latent distribution are treated as hyper-parameters. Thus, search methods such as grid search are needed to determine the setting that achieves the best performance. Currently, there is no golden rule of determining the best dimensions for a particular dataset.
- Discrete-AIR's reconstruction image quality is slightly worse than the original AIR model.

While Discrete-AIR has been proven effective in the two datasets tested, it is not clear if it can be readily scaled to more complex input types. One of the remaining future work is to apply Discrete-AIR on various visual reasoning problems where discrete representations are useful, such as solving Raving Progressive Matrices [5] and symbolic visual question answering [114]. These two problem domains approach visual reasoning with supervised learning methods where, for each object, its category, spatial parameters, and attributes are labelled. Discrete-AIR can be used as a symbolic encoder or an unsupervised pre-trained encoding model for subsequent reasoning tasks, thereby reducing or even completely

removing the requirements for labelled data. Discrete-AIR can also be used to capture the inherent data structures of reasoning tasks like RPM, and used to generate unseen new tasks.

In conclusion, Discrete-AIR enables Neural Diagrammatic Reasoning to learn with fewer labelled data, and to automatically generate interpretable semi-symbolic representations that can be readily integrated with rule-based systems, thereby providing a direction for combining the best of both worlds.

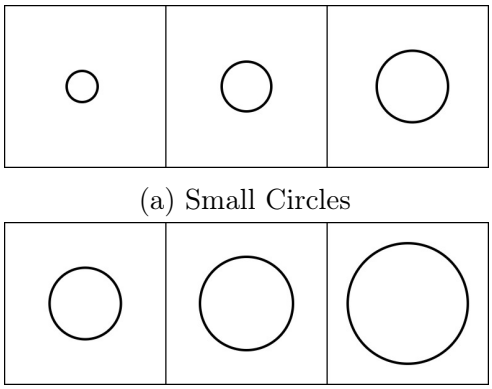
Chapter 6

Generalisable Neural Network for Relational Reasoning

While I discussed two DNN systems (EulerNet and MXGNet) that achieve human-level performance on diagrammatic reasoning tasks in Chapter 3 and Chapter 4, such systems only work well when the training data and the test data are from the same distribution. In Chapter 4, for the "extrapolation" data regime where the test data distribution does not overlap with the training data distribution for selected attributes, MXGNet, and other DNN based systems like WReN [5], suffer from drastically decreased performance that is only slightly better than random chance.

In fact, not only EulerNet and MXGNet but most of today's state-of-the-art DNN systems [80, 97, 5, 8] lack the out-of-distribution (**o.o.d**) generalisation ability. Here, **o.o.d** generalisation is defined as the case where test data distribution only partially overlaps (or does not overlap at all) with training data distribution. Moreover, it is observed that the generalisation error increases as the tasks become more abstract and require more reasoning than perception. This ranges from small drops (3% to 15%) in classification accuracy on ImageNet [80] to accuracy that is only slightly better than random chance for "extrapolation" data regimes in the PGM dataset [5].

In contrast, the human brain is observed to generalise better to unseen inputs [26], and typically requires only a small number of training samples. For example, a human, when trained to recognise that there is a progression relation of circle sizes in Figure 6.1a, can easily recognise that the same progression relation exists for larger circles in Figure 6.1b, even though such a size comparison has not been done between larger circles. However, today's state-of-the-art neural networks [5, 109] are not able to achieve the same. Researchers argue [92, 15, 7, 112] that the human brain evolved to develop special inductive-biases that adapt to the form of information processing needed for humans, thereby improving



(a) Small Circles

(b) Big Circles

Figure 6.1: Size Progression Relations for circles of different sizes.

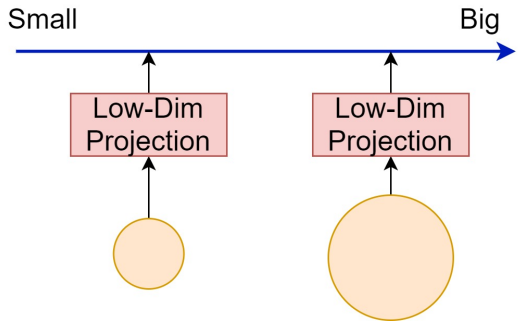


Figure 6.2: Illustration of projecting object representations onto a 1-dimensional manifold in which size comparison can be achieved by simply measuring the difference between two projections.

generalisation. Examples include convolution-like cells in the visual cortex [45, 35] for visual information processing, and grid cells [36] for spatial information processing and relational comparison between objects [7].

In this chapter, I propose a simple yet effective inductive bias which improves generalisation for relational reasoning. This inductive-bias is inspired by neuroscience and psychology research [23, 13, 95] showing that in the primate brain there are neurons in the Parietal Cortex that only respond to different specific attributes of perceived entities. For examples, certain Lateral Intraparietal Cortex (LIP) neurons fire at higher rates for larger objects, while other neurons' firing rate correlates with the horizontal position of objects in the scene (left vs right) [29]. From a computational perspective, this can be viewed as projecting object representations to low-dimensional manifolds, just as the low-dimension projection illustration in Figure 6.2. Based on these observations [95], I **hypothesise** that *these selectively-firing neurons evolved to learn low-dimensional representations of relational structure that are optimised for abstraction and generalisation, and the same inductive bias can be readily adapted for artificial neural network to achieve similar optimisation for abstraction and generalisation.*

I test this hypothesis by designing an inductive bias module that projects high-dimensional object representations into low-dimensional manifolds, and makes comparisons between different objects in these manifolds. I show that this module can be readily amalgamated with existing architectures to improve out-of-distribution generalisation performance for different relational reasoning tasks. Specifically, I perform experiments on three different out-of-distribution generalisation tasks, including maximum of a set, visual object comparison on the dSprites dataset [41] and extrapolation on Progressive Generated Matrices [5]. I show that models with the proposed inductive-bias low-dimensional

comparator modules perform considerably better than baseline models on all three tasks. In order to understand the effectiveness of comparison in low-dimensional manifolds, I analyse the projection space and corresponding function space of the comparator. This shows the importance of projection to low-dimensional manifolds for improving generalisation. Finally I perform analysis relating to algorithmic alignment theory [112], and propose an augmentation to the sample complexity criteria used by this theory to measure algorithmic alignment to better measure algorithmic alignment with generalisation.

6.1 Related works on o.o.d generalisation

The deep neural network's lack of **o.o.d** generalisation (sometimes termed domain generalisation or extrapolation) capability has recently come under scrutiny. Different approaches have been proposed to improve **o.o.d** generalisation, such as reducing superficial domain specific statistics of training data [111, 12], adversarially learning representations that are domain-invariant [67, 1], disentangling representations to separate functional variables with spurious correlations [40, 32], and constructing models with innate causal inference graphs to reduce dependence on spurious correlations [3, 9]. The work in this chapter aligns more with the line of work on discovering inductive bias that improves generalisation. Arguably, CNN [64] is such an inductive-bias that improves generalisation on image datasets, and Graph Neural Network is an inductive-bias that improves generalisation on graph-structured data [7]. Trask et al. [97] proposed Neural Arithmetic Logic Units (NALU), an inductive bias that allows neural networks to learn simple arithmetic with improved **o.o.d** generalisation. Madsen et al. [71] improve NALU for faster and more stable convergence. The low-dim comparators proposed in this chapter can be viewed as an orthogonal work that uses inductive-bias to improve **o.o.d** generalisation for relational reasoning tasks.

6.2 Low-dimensional comparators

In this section, I describe the inductive bias module developed to test my hypothesis that the inductive bias, namely, the low-dimensional representation observed in the Parietal Cortex can be readily adapted for artificial neural network to achieve similar optimisation for abstraction and generalisation. The proposed module learns to project object representations into low-dimensional manifolds and make comparisons in these manifolds. In Section 6.2.1, I describe the module in detail. In Section 6.2.2, 6.2.3, and 6.2.4, I discuss how this module can be utilised for three different relational reasoning tasks, which are finding the maximum of a set, visual object comparisons and Raven Progressive Matrices (RPM) reasoning.

6.2.1 Comparator in low-dimensional manifolds

The inductive-bias module is comprised of low-dim projection functions p and comparators c. Let $\{\boldsymbol{o}_i; i \in 1 \dots N\}$ be the set of object representations, obtained by extracting features from raw inputs such as applying Convolutional Neural Networks (CNN) on images. Pairwise comparison between object pair $\boldsymbol{o}_i$ and $\boldsymbol{o}_j$ can be achieved with a function f expressed as:

$$f(\boldsymbol{o}_i, \boldsymbol{o}_j) = g(\mathop{||}_{k=1}^{K} c_k(p_k(\boldsymbol{o}_i), p_k(\boldsymbol{o}_j))). \tag{6.1}$$

Here p_k is the k^{th} projection function that projects object representation $\boldsymbol{o}$ into the k^{th} low dimensional manifold, c_k is the k^{th} comparator function that compares the projected representations, $||$ is the concatenation symbol, and g is a function that combines the K comparison results to make a prediction. Having K parallel projection functions p_k and comparators c_k allows simultaneous comparison between objects with respect to their different attributes. Figure 6.2 shows an example of comparing sizes of circles by projection onto a 1-dimensional manifold. Both p and c can be implemented as feed forward neural networks. While the comparator c, implemented as a neural network, can theoretically learn a rich range of comparison metrics, I found that adding an additional inductive-bias of distance measure to c for the projection, such as vector distance $p(\boldsymbol{o}_i) - p(\boldsymbol{o}_j)$, or absolute distance $|p(\boldsymbol{o}_i) - p(\boldsymbol{o}_j)|$, improves generalisation performance.

Let $a_t(\boldsymbol{o}_i)$ be the ground truth mapping function from the i^{th} object's representation $\boldsymbol{o}_i$ to its t^{th} attributes (such as colour and size for a visual object). If such ground truth labels of object attributes exist, $f(\boldsymbol{o}_i, \boldsymbol{o}_j)$ can be trained to directly predict the differences in attributes by minimising the loss $\mathcal{L}(d(a_t(\boldsymbol{o}_i), a_t(\boldsymbol{o}_j)), f(\boldsymbol{o}_i, \boldsymbol{o}_j))$, where d is a distance function (e.g., $a_t(\boldsymbol{o}_i) - a_t(\boldsymbol{o}_j)$ for continuous attributes or $\mathbb{1}_{a_t(\boldsymbol{o}_i)=a_t(\boldsymbol{o}_j)}$ for categorical attributes). However, in real-world datasets, such ground truth attribute labels seldom exist. Instead, in many relational reasoning tasks, learning signals for attribute comparison are only provided implicitly in the training objective. For example, in visual question answering tasks, an example question might be 'Is the object behind Object A smaller?'. The learning signals for the required size and spatial position comparator is provided only through the correctness of the answers to the given questions. Thus, the proposed module is only useful and scalable if it can be integrated into neural architectures for relational reasoning and still learn to compare attributes with the weaker, implicit learning signal. Next, I describe 3 examples of such integrations for different relational reasoning tasks, and show in Section 6.3 that the proposed module can learn relational reasoning tasks with better generalisation capability.

6.2.2 Architecture: Maximum of a set

The first task considered is finding the maximum of a set of real numbers. Formally, given a set $\{x_i; i \in 1 \dots N\}$ where x_i is a real number represented as a scalar value, the aim is to train a function $h_{max}(\{x_i, \dots, x_N\})$ that gives the maximum value in the set. Many neural architectures have been applied to this task, including Deep Sets [117] and Set Transformer [66], but none of them test the out-of-distribution (**o.o.d**) generalisation capability. In order to test **o.o.d** generalisation, I create the training and test dataset in such a way that their ranges do not overlap. I sample from the range $(V_{low}^{train}, V_{high}^{train})$ for the training set, and from the range $(V_{low}^{test}, V_{high}^{test})$ for the test set, and restrict that $V_{high}^{train} < V_{low}^{test}$, so that the highest training number is less than the lowest testing number.

I integrate the proposed low-dim comparator module with Set Transformer [66], a state-of-the-art neural architecture for sets. Set Transformer first encodes each element in the set with respect to all other elements with a Multihead Attention Block (MAB). MAB is an attention module modified from self-attention used in language tasks [102]. MAB encodes input x_i into representation e_i based on other inputs in the set x_j as: $e_i = encode(x_i) = \mathsf{MAB}(x_i, x_j)$. MAB with parameters ω is defined as follow:

$$\mathsf{MAB}(X, Y) = \mathsf{LayerNorm}(H + \mathsf{rFF}(H)) \tag{6.2}$$

$$H = \mathsf{LayerNorm}(X + \mathsf{MultiHead}(X, Y, \omega)) \tag{6.3}$$

Here, $LayerNorm$ is layer normalisation [4], and rFF is row-wise feed forward network. The Set Transformer then uses Pooling with MultiHead Attention (PMA) to combine all encoded elements of the set as $\mathsf{PMA}(e_1, \dots, e_N)$. While MAB uses query and key embeddings to generate attention variables, which are then used as weights in the weight sum of the value embeddings of elements, I swap the query-key attention mechanism with the proposed low-dim comparator as:

$$e_i = MLP(\sum_{j=1}^{N} f(x_i, x_j)) \tag{6.4}$$

Here f is the low-dim comparator and MLP is a standard Multi-Layer Perceptron. Note that the scalar input x_i is directly used here as object representation $\boldsymbol{o}_i$ in Equation (6.1) since no feature extraction is needed. I then use attention-based pooling to combine projection of x_i as $\sum_{i=1}^{N} a(e_i)p(x_i)$, where a outputs attention values, while p is the 1-dim projection function. For detailed architecture configuration, please refer to Appendix D.1.

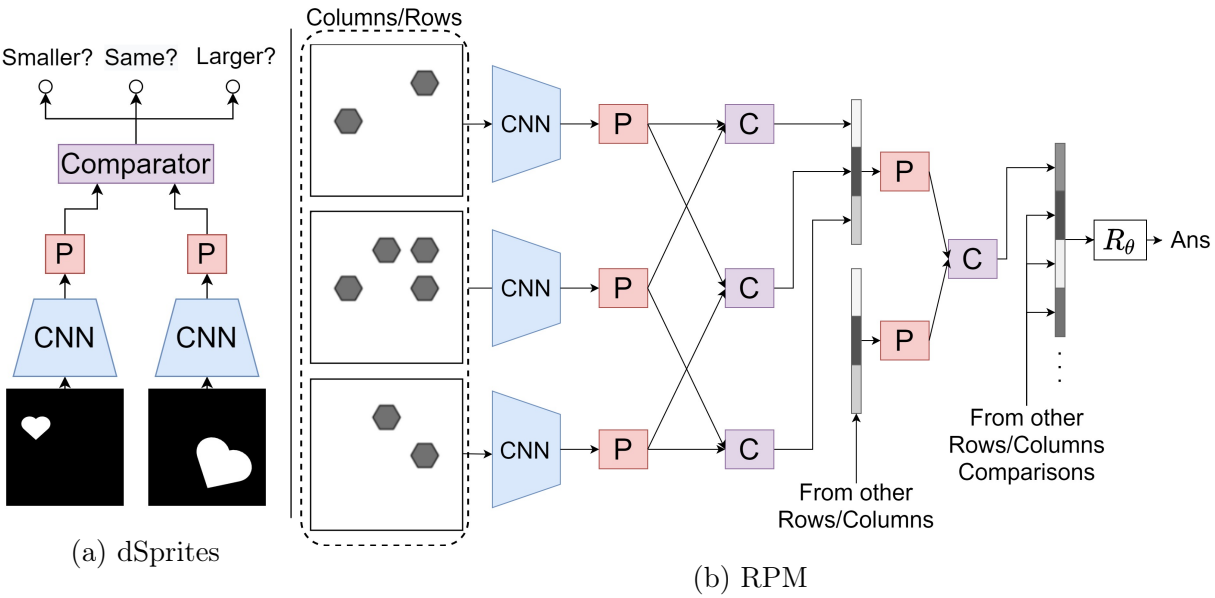
6.2.3 Architecture: Visual object comparison

The second task is comparing visual objects for different attributes such as size and spatial position. For this task, two images, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, containing single objects of randomly sampled attributes are given, and one is asked if a specific attribute of the second object is larger than, equal to, or smaller than the attribute of the first object. Figure 6.3a shows an example of this task comparing sizes between two heart-shaped objects. For implementation, I use the dSprites dataset [41], a widely used dataset for studying latent space disentangling, to sample images of objects. To test out-of-distribution generalisation, I sample the training set and the test set such that for the compared attribute, the training attribute range has no overlap with the test attributes. I leave details of the dataset construction to Appendix D.2.

Figure 6.3a shows an overview of the architecture integrated with the proposed low-dimensional comparator. The image pair $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ is first passed through a CNN to extract feature embeddings $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$. The feature embeddings are then projected to a low-dim manifold and compared as $c(p(\boldsymbol{e}_1), p(\boldsymbol{e}_2))$, where p is the projector and c is the comparator. The comparator has 3 output units with Softmax to predict probabilities that an attribute of the second object $a(\boldsymbol{x}_2)$ is smaller than, equal to, or larger than the attribute of the first object $a(\boldsymbol{x}_1)$. The architecture is trained with cross entropy loss with respect to ground truth labels. While I am testing the o.o.d generalisation of relational reasoning, it is reasonable to expect that the visual perception module is exposed to all possible scenarios of the input distribution in an unsupervised way. The same assumption also holds for humans, whose vision system has to be sufficiently exposed to inputs from the world after birth before they can associate objects with semantic meaning and perform relational reasoning [75]. Thus, I initialise the CNN with the pretrained encoder weights of Beta-VAE [41], a disentangled VAE model trained in the unsupervised setup on the dSprites dataset. For the detailed configuration of the architecture, please refer to Appendix D.3.

6.2.4 Architecture: visual reasoning for Raven Progressive Matrices

The third task is a more complex visual reasoning task named 'Raven Progressive Matrices' (RPM) (introduced in Section 2.2.2), which is a popular human IQ test. In this task, one is given 8 context diagrams with logic relations present in them, and is asked to pick an answer that best fits with the context diagrams. In this experiment, I use the PGM dataset [5], the largest RPM-style dataset available. In the PGM dataset, there is a special data split called 'extrapolation' that is designed to test for **o.o.d** generalisation.



(a) dSprites

(b) RPM

Figure 6.3: Figure (a) illustrates the architecture for comparing sizes of objects sampled from the dSprites dataset. "P" is the projection function. Figure (b) illustrates the architecture for logical reasoning on RPM-style tasks. "P" is the projection function and "C" is the comparator (g in Equation 6.1). CNN used for processing different diagrams in the same task share weights.

In the extrapolation split, colour and size values of objects in the training set are sampled from the lower half of the range, while the same attributes in the test sets are sampled from the upper half of the range. Thus the attribute ranges of training and test sets are non-overlapping. For a recap of the PGM dataset, please refer to Section 2.2.2 or Barrett et al [5].

The proposed architecture integrates the low-dim comparator with a Multi-Layer Relation Network [50]. Figure 6.3b shows an overview of the architecture developed for PGM tasks. I use a 2-layer relation network, with the first layer encoding pairs of diagrams within a row/column, and the second layer encoding pairs of encoded representations of rows or columns. Applying such prior knowledge, namely that rules only exist in rows and columns, has been standard practice in state-of-the-art methods for RPM reasoning [109, 121]. Following [109], each of the 8 candidate answers are filled into the third row and column to obtain, in total, 16 answer rows and columns. At each layer of the relation network, I use the low-dimensional comparator instead of the MLP in the original relation network [85] for diagram comparison. Diagram $\boldsymbol{x}_i$ is first passed through a CNN to produce embedding $\boldsymbol{o}_i$. Embedding pairs $(\boldsymbol{o}_i, \boldsymbol{o}_j)$ are then compared as $\boldsymbol{e}_{ij} = f(\boldsymbol{o}_i, \boldsymbol{o}_j)$, where f is the low-dimensional comparator described in Equation (6.1). Comparison results from the same rows/columns are then concatenated to form row/column embedding

$\boldsymbol{r}_{ijk} = \boldsymbol{e}_{ij}||\boldsymbol{e}_{ik}||\boldsymbol{e}_{jk}$. The row/column embeddings are compared with the second layer comparator. The comparison results are then concatenated and input into a reasoning network, R_θ, to predict the correct answer. Similar to the dSprites comparison task, as discussed in Section 6.2.3, I pre-train a CNN as the encoder of VAE, a technique that has also been previously explored for the PGM dataset [94]. For the detailed configuration of the architecture, please refer to Appendix D.4.

6.2.5 Algorithmic alignment and o.o.d generalisation

Xu et al [112] proposed to measure the algorithmic alignment of neural networks against a specific task with sample complexity $\mathcal{C}_{\mathcal{A}}(g, \epsilon, \delta)$, which is the minimum sample size M so that g, the ground truth label mapping function, is (M, ϵ, δ) learnable with a learning algorithm $\mathcal{A}$. This essentially says that a model is more algorithmically aligned with a task if it can learn the task more easily with fewer samples. However, in the original definition, both training and test data are independently and identically distributed (i.i.d) samples drawn from the same data distribution. Thus, the algorithmic alignment theory measures how well an NN can fit to a particular data distribution, but does not measure how well an NN can model perform in the **o.o.d** scenario. For example, for the visual object comparison task, an over-parameterised MLP can learn the following two algorithms with low complexity: (1) $m(hash(\boldsymbol{o}_i), hash(\boldsymbol{o}_j))$, where $hash$ is a hashing function and m is a memory read/write function based on the hash index; and (2) $c(p(\boldsymbol{o}_i), p(\boldsymbol{o}_j))$, which is the proposed comparator function. While both algorithms can fit well for the training data, the first algorithm clearly does not **o.o.d** generalise, as the memory function does not store unseen samples. In Section 6.3.5, I also show via experiment that algorithmic alignment is not indicative of **o.o.d** generalisation.

Intuitively, a more algorithmically-aligned model should generalise better, as it better captures the underlying algorithm of label generation. Here I propose an augmentation to the sample complexity metric (Definition 3.3 in Xu et al [112]) in order to measure algorithmic alignment with generalisation.

Definition 6.2.1. o.o.d metric. We first fix an error parameter $\epsilon > 0$ and failure probability $\delta \in (0, 1)$. Suppose $\{x_i^s, y_i^s\}_{i=1}^M$ are i.i.d samples from distribution $\mathcal{D}_s = \mathcal{T}(\mathcal{D}, \beta, \mathbf{u})$, where $\mathcal{D}$ is the full data distribution, $\mathcal{D}_s$ is the truncated distribution (s denotes subset), $\mathcal{T}$ is a truncating function, $\beta \in (0, 1)$ is the truncation ratio, and $\mathbf{u}$ is the set of dimensions for truncation. Additionally, let V be the value range for each dimension of distribution $\mathcal{D}$. The truncation function $\mathcal{T}$ selects dimension $\mathbf{u}$, and truncates $\mathcal{D}$ to only keep probability mass in the range $(V_{min}^{\mathbf{u}}, \beta V_{min}^{\mathbf{u}})$, and lastly normalise the probability distribution. Let g be the underlying data function $y_i = g(x_i)$, and $f = \mathcal{A}(\{x_i, y_i\}_{i=1}^M)$ be

the function learned with learning algorithm $\mathcal{A}$. Then g is $(M, \epsilon, \delta, V, \beta, \mathbf{u}) - learnable$ with $\mathcal{A}$ if:

$$\mathbb{P}_{x \sim \mathcal{D}}[||f(x) - g(x)|| < \epsilon] \geq 1 - \delta \tag{6.5}$$

The sample complexity is the minimum M for g to be $(M, \epsilon, \delta, V, \beta, \mathbf{u}) - learnable$ with $\mathcal{A}$. In Section 6.3.5 I experimentally show that this metric better measures an NN's ability to generalise.

6.3 Evaluation

6.3.1 Maximum of a set

For the task of finding the maximum number in a set, I randomly sample number sets of cardinality ranging from 2 to 20 for training, and 2 to 40 for testing. For number sets for training, I uniformly sample numbers in the real value range $[0, 100)$. For testing I sample numbers in the range $[100, 200]$. In this way, I both test if the model can generalise for sets of larger cardinality and for numbers sampled from an unseen range. 10000 sets are sampled for training and 2000 sets for testing. For hyper-parameters of this and subsequent experiments, please refer to Appendix D.5. Table 6.1 shows the test error of the proposed model compared against Deep Sets [117] and Set Transformer [66], two previous state-of-the-art architectures for sets. The proposed model achieves much lower **o.o.d** generalisation error than other methods, even lower than Deep Sets with a built-in Max-Pooling function.

Table 6.1: **o.o.d** generalisation test error for learning to find the maximum number in a set of numbers ($mean \pm std$ for 10 runs). M.S.E means Mean Squared Error. The high M.S.E of Deep Sets with Mean pooling shows that architectures with mean pooling cannot effectively learn max pooling functions.

Model	Deep Sets [117](Mean)	Deep Sets [117](Max)	Set Transformer [66]	OURS
M.S.E	73.22 ± 17.11	0.51 ± 0.29	1.62 ± 0.76	$\mathbf{0.0015 \pm 0.0008}$

6.3.2 Visual object comparison

For the visual object comparison task, I set three sub-tasks for comparing different attributes of the object, including size, horizontal position, and colour intensity. For each task I sampled visual objects with a different range for the compared attributes from the dSprites dataset [41]. Given the compared attribute range $[V_{low}, V_{high}]$, I sample training data from range $[V_{low}, \frac{2}{3}V_{high})$ and test data from range $[\frac{2}{3}V_{high}, V_{high}]$. As ground truth attribute value is provided in the dSprites dataset, comparison labels can be built as $(\mathbb{1}_{a_1<a_2}, \mathbb{1}_{a_1=a_2}, \mathbb{1}_{a_1>a_2})$. For all experiments, I sample 60000 training pairs and 20000 test

pairs. I test the proposed model against an MLP baseline, which directly processes the object representations o_i and o_j extracted by CNN as $MLP(o_i, o_j)$. I select the best MLP by hyper-parameter search over the number of layers and layer sizes. I use 1-dimensional projection as this is found to give the highest accuracy. Table 6.2 shows the **o.o.d** generalisation test accuracies of the proposed model compared against the baseline. The proposed model with the low-dimensional comparator significantly outperforms baselines for all three compared attributes.

Table 6.2: **o.o.d** generalisation test accuracies of baseline and our proposed model for the dSprites attribute comparison task (10 runs). X-Coord is horizontal position.

Model \ Attributes	Size	X-Coord	Colour
Baseline	$79.52 \pm 6.71\%$	$66.14 \pm 5.03\%$	$78.45 \pm 5.04\%$
OURS	$\mathbf{94.05 \pm 3.03\%}$	$\mathbf{79.11 \pm 1.92\%}$	$\mathbf{91.39 \pm 5.84\%}$

6.3.3 Visual reasoning for Raven Progressive Matrices

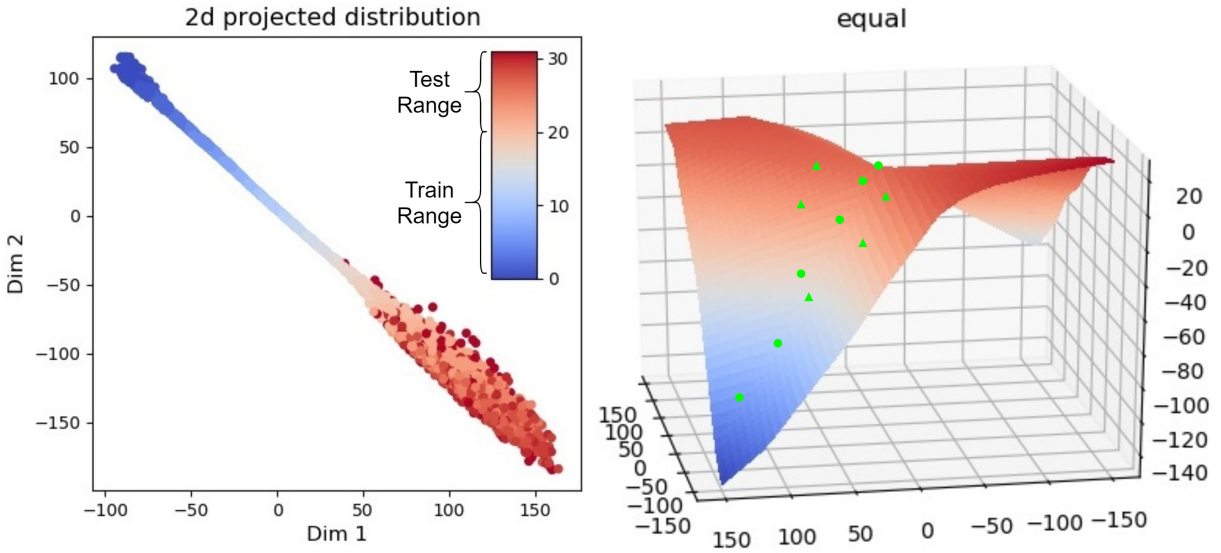
For the RPM-style task, I use the extrapolation split of PGM dataset [5], which is already a well-defined **o.o.d** generalisation task. I compare the proposed model against all previous methods (to the best of my knowledge) that have reported results on the extrapolation data split. I additionally include a baseline model named "MLRN-P", which is a 2-layer MLRN [50] with prior knowledge of the relations only present in rows/columns and with pre-training. Table 6.3 shows the test accuracy comparison. The proposed model outperforms all other baselines. Note here that a vanilla CNN is used as the perception module, which is the same as in most previous methods [5, 121, 50] on RPM tasks. Multiple-object representation learning methods [34, 62, 105], which achieve better results for multi-object scene learning than CNN, can be investigated for potential improvement in generalisation performance. This is left as future work.

Table 6.3: **o.o.d** generalisation test accuracies for the extrapolation split of the PGM dataset.

Model	WReN [5]	MXGNet [109]	MLRN [50]	MLRN-P	OURS
Accuracy	17.2%	18.9%	14.9%	18.1%	**25.9%**

6.3.4 Why low dimension?

While I show that a comparator in low-dimensional manifolds improves **o.o.d** generalisation for a range of relational reasoning tasks, the reason behind it is still not clear. In this section, I analyse the projection space and comparator function landscapes of the visual



(a) 2D projection space

(b) 2D comparator function

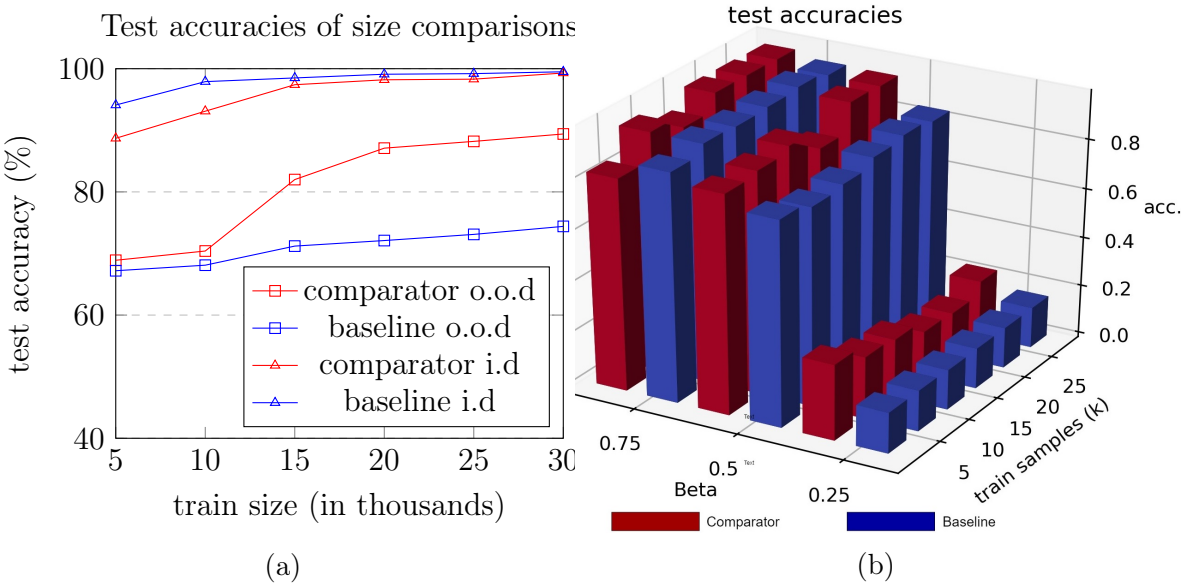
Figure 6.4: (a) shows a scatter plot of the 2D projected distribution of objects in the task of comparing vertical positions of objects in the image. The x-axis and y-axis are 2 dimensions of the projection manifold. Latent vertical position (ranging from 0 to 31) is indicated by colour. The training range is $[0, 19]$ and the test range is $[20, 31]$. (b) plots the comparator's function landscape for the output unit checking if attributes of compared objects are equal, in the space of vector differences between 2D projections of objects. Green circles represent vector difference sampled from the training set while triangles represent vector difference sampled from the test set.

object comparison task to shed light on this. Figure 6.4 shows plots of projection space and comparator function landscapes for comparing spatial positions. For plots of comparisons of other attributes, please refer to Appendix D.6. I first state 3 observations invariant across different sub-tasks comparing different attributes:

1. *When the ground truth attribute can be represented in a 1-dimensional manifold (such as vertical position), comparators in higher dimensions learn to project the object representation into a 1-dimensional manifold.* Figure 6.4a illustrates this with a plot of projection distribution for the task of comparing vertical positions. It can be observed that, even though the projection space is 2-dimensional, the projected points cluster around a line.

2. *The projection of the test data in the manifold is less clustered around the sub-manifold of attributes than that of training data.* This can be observed from Figure 6.4a, where the points projected from the test set are more spread out than those from the training set. There is also less order in the distribution of test points, where points of noticeably different intensity appear next to each other.

3. *The function landscape becomes less defined outside of the sub-manifold.* Figure 6.4b



(a) (b)

Figure 6.5: Figure (a) shows o.o.d and i.d. (identically distributed) test accuracy of the baseline and comparator for different training sample sizes. (b) shows test accuracies of the baseline and comparator for different training sample sizes and with different β-rates for truncating training distribution.

plots the comparator function landscape for the "equal" ($\mathbb{1}_{a_1=a_2}$) output unit (checking if attributes of compared objects are equal) over the space of vector differences between 2D projected representation $p(o_2) - p(o_1)$. Green circles and triangles represent vector differences of sampled points from the training and test sets respectively. The equality function is well defined in the sub-manifold in which training points (circles) lie, peaking close to the $(0, 0)$ point. However, outside of the training points' sub-manifold, the function is more random, with a significant region with a higher function value than at $(0, 0)$ point. Note that the vector differences of test points (triangles) may be in this region.

From the above observations, I conclude that, when comparators are of higher dimension than the intrinsic dimension of compared attributes, the projection tends to lie in a sub-manifold of the same dimension as the attributes, resulting in the comparator function only being well defined in that manifold. However, the projection of test data tends to escape from this sub-manifold into the region where the comparator function is not trained, resulting in incorrect prediction, and therefore low **o.o.d** generalisation performance. The take away from these observations is that comparison in manifolds of dimensions closer to the dimension of the latent variable will result in less test data falling into unseen space during training, and thereby improve **o.o.d** generalisation.

6.3.5 Algorithmic alignment

Figure 6.5a shows the **o.o.d** and **i.d** (identically distributed) test accuracies for the size comparison task of the baseline and the proposed comparator model for different training sample sizes. It can be observed that the **i.d** test accuracies' sample complexity (training samples needed to achieve the same accuracy), which measures algorithmic alignment, is not indicative of **o.o.d** test accuracies. This motivates the proposition of the augmented algorithmic alignment theory in Section 6.2.5.

Figure 6.5b shows the size comparison of the test accuracies of the baseline and comparator for different training sample sizes with different β rates for truncating the training distribution along the latent dimension 'size'. This corresponds to the proposed metric in Section 6.2.5. The new metric reflects that the model, which learns better with truncated training distribution, is the one with better **o.o.d** generalisation.

6.3.6 Ablation Studies

We performed ablation studies of different hyper-parameters in our experiments. and show the results in Table 6.4, Table 6.5, and Table 6.6. Table 6.4 shows the test accuracy of architectures with different numbers of projection functions for the PGM task. The number of projection functions is indeed another hyper-parameter to tune, but we think the improved performance is definitely worth it. Table 6.5 shows the test accuracy on the extrapolation split of the PGM dataset with different pre-training for the perception module. We picked the hyper-parameters giving the best extrapolation test accuracy. Table 6.6 shows the effectiveness of different comparator input types on the test accuracy for comparing 'colour' attributes in the visual object comparison task.

Number of Projector Functions	Accuracy
128	20.1
256	24.2
512	25.9

Table 6.4: Ablation Study on Number of Projector/Comparator functions for the PGM task.

Pre-trained Encoder	Accuracy
β-VAE (β=4, dim=64) [94]	23.5
β-VAE (β=4, dim=128)	25.1
β-VAE (β=1, dim=128) (OURS)	25.9

Table 6.5: Ablation study on different pre-trained encoders for the PGM task.

We also performed an ablation study on the dimensionality of the projected embedding space. Figure 6.6 shows the plot of test accuracy against different dimension sizes of

Comparator Input Type	Accuracy
Vector Difference	$97,71 \pm 2.81\%$
L1 Difference	$69.2 \pm 14.9\%$
Concatenation	$91.2 \pm 6.52\%$

Table 6.6: Ablation study on different input types of comparator functions for 'colour' comparison task.

the projection functions for the visual object comparison tasks. It can be observed that increasing dimension sizes (x-axis) reduce the test performance. This further validate that low-dimensional embeddings are crucial in improving neural architecture's extrapolation performance.

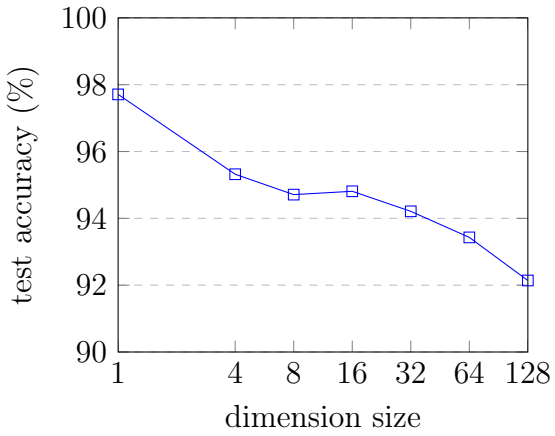


Figure 6.6: Projection dimension ablation study for visual object comparison tasks.

6.4 Discussion

In this chapter, I introduced an inductive-bias module that can be readily integrated into DNN architectures to improve the **o.o.d** generalisation performance of DNNs on a range of relational reasoning tasks. The tasks range from simpler, non-visual "maximum of a set", to "Raven Progressive Matrices" with complex visual input.
Pros of the proposed module include:

- This module reduces the existing relational reasoning architectures' drawback in low performance when test data is out of training range, and thereby allows DNNs to be more applicable to more real-world scenarios where building unbiased approximated data distributions of real-world processes is usually very costly.
- The proposed module can be easily integrated into existing neural architectures with negligible increase in computational complexity.

- I also proposed an augmented algorithmic alignment theory for better measuring the algorithmic alignment by taking into account the **o.o.d** generalisation capability. The augmented theory no longer measures how well the architecture fits the training distribution, but measures how well the architecture fits the underlying data generating process instead.

Cons of the proposed module include:

- The number of comparators, dimensions of each comparator and the type of comparator functions must be searched as hyper-parameters for each dataset. This could be a time-consuming process for large datasets and large models.
- It is not clear if the proposed module improves interpretability and explanability of the neural architecture.

Future Directions: While the module works well in the simpler setting of finding the maximum of a set and the visual object comparison of a single attribute, it does not achieve the same improvement for RPM tasks, where the input diagrams are much more complex and the comparisons are made on many different attributes in a parallel way. In fact, the **o.o.d** performance gains decrease as the input gets more complex. This is most likely because I only implemented a simple, proof-of-concept integration of low-dimensional comparators into MLRN. There are many parts that could be improved when tackling multi-object multi-relation reasoning tasks. I list below two potential directions for future research:

- **More generalisable perception module**: While the low-dimensional comparator is a generalisable relational reasoning module, its generalisation capability is conditioned on a similar level of generalisation capability in the lower-stream perception module. If the perception module is not able to summarise unseen input into a useful representation, the reasoning module will not be able to function as well.
- **Disentangled parallel comparison of multiple attributes**: I did not investigate how the low-dimensional comparator projects object representations when there are multiple attributes to be compared simultaneously. An interesting question to ask is how well the low-dimensional comparator can learn a disentangled projection mechanism with each projector module processing a specific attribute. Research into disentanglement and potential ways to encourage disentanglement might be a promising direction in further improving **o.o.d** generalisation performances on the multi-object, multi-relation reasoning tasks.
- **Extension to other reasoning tasks**: While I evaluated the module on the three tasks above, there are many other relational reasoning tasks to tackle. For example,

visual question answering is an important task that is more applicable to real-world scenarios. I plan to extend the evaluation to the "CLEVR" dataset [54], a VQA dataset that emphasises visual relational reasoning.

Chapter 7

Conclusion

In this dissertation I investigated the application of Deep Neural Networks on diagrammatic reasoning tasks. The core argument that *Deep Neural Networks can successfully learn to reason with diagrams with little human prior input, are robust to noise, and are easily adaptable to other diagram domains* has been validated throughout the chapters. I also discussed ways of reducing the short-comings of DNN-based reasoning systems, such as using generative modelling to reduce the need for labelled data, and new architectures with improved **o.o.d** generalisation capability.

To conclude, I briefly summarise the main contributions in this dissertation, and discuss potential future directions for improving DNN-based diagrammatic reasoning systems.

7.1 Main contributions

- In Chapters 3 and 4, I introduced two DNN-based diagrammatic reasoning systems, namely EulerNet and MXGNet, which tackle Euler diagram syllogisms and Raven Progressive Matrices tasks. EulerNet is one of the first DNN architectures ever developed for diagrammatic reasoning, and achieves near perfect accuracy. As EulerNet has the drawback of less scalability to diagrams with an arbitrary number of entities, I also propose MXGNet, a new way of formulating objects in diagrams and their relations into multiplex graphs that achieves performance on par with human test takers. While I demonstrated that DNN can learn to reason with diagrams with little human-prior input in defining diagram domain or reasoning rules, I also showed that EulerNet can be very robust to random noise and deformations, a clear advantage over mechanised reasoning systems.

- EulerNet and MXGNet both require large amounts of labelled training data to achieve good accuracy. However labelled data collection is a very expensive process. To remedy this, in Chapter 5, I introduced Discrete-AIR, a completely unsupervised

generative model that learns to summarise diagrams of multiple objects into a list of neuro-symbolic representations. Discrete-AIR tackles DNN-based systems' disadvantage of requiring a large amount of labelled data by allowing the perception module of the system to be pre-trained without any labels, thereby drastically reducing the cost of collecting labelled data. Moreover, the neuro-symbolic representations obtained from Discrete-AIR give direct interpretability of both the learned representations and the potential subsequent reasoning systems. The neuro-symbolic representations can also be readily integrated with a rule-based reasoning system to get the best of both worlds.

- MXGNet, while achieving SOTA performance, still perform poorly when the test data is out of the training distribution. To tackle this, in Chapter 6, I introduced a novel inductive-bias module for improving DNN architecture's **o.o.d** performance on relational reasoning tasks. This module learns to project object representations onto lower dimensional manifolds for more robust attribute comparison. Experiments show that this module can be integrated into different architectures to improve **o.o.d** generalisation on a range of relational reasoning tasks. I also developed an augmented version of the algorithmic alignment theory that better measures algorithmic alignment with consideration for generalisation performance.

7.2 Future directions

This dissertation shows that DNN systems can be successfully applied to diagrammatic reasoning tasks, and moreover, some of the DNN diagrammatic reasoning system's disadvantages (such as limited interpretability and generalisation capability) can be addressed. There are areas that can be improved and potential directions that are worth exploring. Here, I list four potential directions that I think will benefit the machine learning and visual reasoning community, and suggest plausible research ideas.

7.2.1 Interpretability

While EulerNet and MXGNet perform well on diagrammatic reasoning tasks, they are not fully interpretable. EulerNet is only interpretable in the part of the learned representations that correlate with attributes of the input diagram. I have not yet investigated the interpretability of MXGNet. A fully interpretable DNN reasoning system will allow the extraction of reasoning rules from the DNN. This will have a large impact, as such interpretable systems could be applied to any new diagrammatic or visual reasoning tasks to automatically extract diagram-to-symbol mapping and a set of reasoning rules. Moreover, the interpretability would also allow users to understand the mechanisms underlying the

reasoning, thus making it easier to debug and making users more confident in operating these systems.

Here I point out two potential research ideas for those interested. First, Discrete-AIR is only applied on synthetic images from the Multi-Sprites and Multi-MNIST datasets. It is not clear if it can learn structured latent distribution for real-world images where objects are deformable, occluded, and view-point dependent. New architectural designs are potentially necessary for such real-world applications. Secondly, it would be interesting to investigate how well Discrete-AIR can be used together with a traditional mechanised reasoning system to get the best of both worlds. Such works would not only provide direct interpretability of the whole reasoning system, but also improve generalisation as rule-based systems have universally accurate rules in the defined system. This will be further discussed in Section 7.2.3.

7.2.2 Generalisation

While I have made progress in improving DNN reasoning system's generalisation performance in Chapter 6, the results on more complex RPM tasks are still below expectation. There is still much work to be done to further improve DNN's generalisation performance on reasoning tasks.

Two potential directions, also outlined in Section 6.4, are developing more generalisable perception module and improving comparators for parallel comparison of multiple attributes. For the more generalisable module, I think there are two interesting directions to explore, which are self-supervised learning and neuro-symbolic models. Self-supervised learning allows model to learn more generalisable representations using a much larger unlabelled dataset, while neuro-symbolic models restrict the model's output to a pre-defined symbolic space, and thereby reduce variance. On improving the comparator, currently all parallel comparators are initialised using the same hyper-parameters. It is possible that, for certain attributes, the comparator might need more layers for processing, while for some other attributes, MLP may not be the most suitable module. Thus it would be interesting to design a more diverse parallel comparator system.

7.2.3 Integration of neural and symbolic systems

DNN-based systems do not require human prior input, and are robust to noise. Symbolic rule-based systems do not need labelled training data, and guarantee correctness and interpretability. An interesting question to ask is if we can get the best of both worlds. While a preliminary exploration of neural symbolic integration has been done for visual question answering [114], such explorations are still much needed for other types of visual reasoning tasks, such as RPM tasks. Discrete-AIR, introduced in Chapter 5, provides a

potential way of training a front-end neuro-symbolic DNN module that can be readily integrated with rule-based systems.

7.2.4 Generative Modelling for more tasks

In Chapter 5 I applied Discrete-AIR on the Multi-MNIST and Multi-Sprites datasets. Both datasets contain a limited number of un-occluded objects. It will be interesting to test how Discrete-AIR will perform on more complex scenes such as RPM and visual question answering tasks. In particular, it will be interesting to investigate what representations can be learned for these more complex scenes, and if the generative models can be used for generating unseen scenes.

One thing that potentially limits Discrete-AIR in scaling to more complex scenes is the LSTM backbone. It will be interesting to see if some more recent sequential models, such as transformers, can be applied here to improve its scalability.

Bibliography

[1] Isabela Albuquerque, João Monteiro, Tiago H Falk, and Ioannis Mitliagkas. Adversarial target-invariant representation learning for domain generalization. *arXiv preprint arXiv:1911.00804*, 2019.

[2] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433, 2015.

[3] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[5] David Barrett, Felix Hill, Adam Santoro, Ari Morcos, and Timothy Lillicrap. Measuring abstract reasoning in neural networks. In *International Conference on Machine Learning*, pages 511–520, 2018.

[6] Jon Barwise and John Etchemendy. *Hyperproof*. CSLI Press, 1994.

[7] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[8] Yonatan Belinkov and Yonatan Bisk. Synthetic and natural noise both break neural machine translation. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=BJ8vJebC-`.

[9] Yoshua Bengio, Tristan Deleu, Nasim Rahaman, Nan Rosemary Ke, Sebastien Lachapelle, Olexa Bilaniuk, Anirudh Goyal, and Christopher Pal. A meta-transfer objective for learning to disentangle causal mechanisms. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=ryxWIgBFPS`.

[10] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[11] Nancy B Carlisle, Jason T Arita, Deborah Pardo, and Geoffrey F Woodman. Attentional templates in visual working memory. *Journal of Neuroscience*, 31(25): 9315–9322, 2011.

[12] Fabio M Carlucci, Antonio D'Innocente, Silvia Bucci, Barbara Caputo, and Tatiana Tommasi. Domain generalization by solving jigsaw puzzles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2229–2238, 2019.

[13] Matthew V Chafee. A scalar neural code for categories in parietal cortex: Representing cognitive variables as "more" or "less". *Neuron*, 77(1):7–9, 2013.

[14] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[15] François Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

[16] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*, pages 2702–2711, 2016.

[17] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11):1944–1957, 2007.

[18] Emilien Dupont. Learning disentangled joint continuous and discrete representations. In *Advances in Neural Information Processing Systems*, pages 708–718, 2018.

[19] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[20] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314*, 2013.

[21] SM Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Geoffrey E Hinton, et al. Attend, infer, repeat: Fast scene understanding with generative models. In *Advances in Neural Information Processing Systems*, pages 3225–3233, 2016.

[22] Leonhard Euler. *Lettres à une princesse d'Allemagne: sur divers sujets de physique & de philosophie*. PPUR presses polytechniques, 2003.

[23] Jamie K Fitzgerald, David J Freedman, Alessandra Fanini, Sharath Bennur, Joshua I Gold, and John A Assad. Biased associative representations in parietal cortex. *Neuron*, 77(1):180–191, 2013.

[24] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[25] Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, SM Ali Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. In *International Conference on Machine Learning*, pages 1666–1675, 2018.

[26] Robert Geirhos, Carlos RM Temme, Jonas Rauber, Heiko H Schütt, Matthias Bethge, and Felix A Wichmann. Generalisation in humans and deep neural networks. In *Advances in Neural Information Processing Systems*, pages 7538–7550, 2018.

[27] Joseph Diaz Gergonne. Essai de dialectique rationelle. *Annuales de Mathematiques pures et appliqukes*, 7:189–228, 1817.

[28] Ben Goertzel and Cassio Pennachin. *Artificial general intelligence*, volume 2. Springer, 2007.

[29] Mengyuan Gong and Taosheng Liu. Biased neural coding of feature-based attention in human brain. *bioRxiv*, page 688226, 2019.

[30] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[32] Sven Gowal, Chongli Qin, Po-Sen Huang, Taylan Cemgil, Krishnamurthy Dvijotham, Timothy Mann, and Pushmeet Kohli. Achieving robustness in the wild via adversarial mixing with disentangled representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1211–1220, 2020.

[33] Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Neural expectation maximization. In *Advances in Neural Information Processing Systems*, pages 6691–6701, 2017.

[34] Klaus Greff, Raphaël Lopez Kaufman, Rishabh Kabra, Nick Watters, Christopher Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner. Multi-object representation learning with iterative variational inference. In *International Conference on Machine Learning*, pages 2424–2433, 2019.

[35] Umut Güçlü and Marcel AJ van Gerven. Deep neural networks reveal a gradient in the complexity of neural representations across the ventral stream. *Journal of Neuroscience*, 35(27):10005–10014, 2015.

[36] Torkel Hafting, Marianne Fyhn, Sturla Molden, May-Britt Moser, and Edvard I Moser. Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436(7052): 801–806, 2005.

[37] Lukas Hahne, Timo Lüddecke, Florentin Wörgötter, and David Kappel. Attention on abstract visual reasoning, 2020. URL `https://openreview.net/forum?id=Bkel1krKPS`.

[38] Eric Hammer and Sun-Joo Shin. Euler's visual logic. *History and Philosophy of Logic*, 19(1):1–29, 1998.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[40] Christina Heinze-Deml and Nicolai Meinshausen. Conditional variance penalties and domain shift robustness. *arXiv preprint arXiv:1710.11469*, 2017.

[41] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017.

[42] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[43] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 804–813, 2017.

[44] Ronghang Hu, Jacob Andreas, Trevor Darrell, and Kate Saenko. Explainable neural computation via stack neural module networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 53–69, 2018.

[45] David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

[46] Drew A Hudson and Christopher D Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6693–6702. IEEE, 2019.

[47] Drew Arad Hudson and Christopher D. Manning. Compositional attention networks for machine reasoning. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=S1Euwz-Rb`.

[48] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[49] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015.

[50] Marius Jahrens and Thomas Martinetz. Solving raven's progressive matrices with multi-layer relation networks. *arXiv preprint arXiv:2003.11608*, 2020.

[51] Mateja Jamnik, Alan Bundy, and Ian Green. On automating diagrammatic proofs of arithmetic arguments. *Journal of logic, language and information*, 8(3):297–321, 1999.

[52] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[53] Michael Janner, Sergey Levine, William T. Freeman, Joshua B. Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-centric models. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HJx9EhC9tQ`.

[54] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2901–2910, 2017.

[55] Ta-Chu Kao and Mason A Porter. Layer communities in multiplex networks. *Journal of Statistical Physics*, 173(3-4):1286–1302, 2018.

[56] Diederik P Kingma and Jimmy Lei Ba. Adam: A method for stochastic gradient descent. In *ICLR: International Conference on Learning Representations*, 2015.

[57] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[58] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *International Conference on Machine Learning*, pages 2688–2697, 2018.

[59] Thomas Kipf, Elise van der Pol, and Max Welling. Contrastive learning of structured world models. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=H1gax6VtDB`.

[60] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[61] Ulrich Kortenkamp and Jürgen Richter-Gebert. Using automatic theorem proving to improve the usability of geometry software. In *Proceedings of MathUI*, volume 2004, 2004.

[62] Adam Kosiorek, Sara Sabour, Yee Whye Teh, and Geoffrey E Hinton. Stacked capsule autoencoders. In *Advances in Neural Information Processing Systems*, pages 15512–15522, 2019.

[63] Jill H Larkin and Herbert A Simon. Why a diagram is (sometimes) worth ten thousand words. *Cognitive science*, 11(1):65–100, 1987.

[64] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[65] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[66] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753, 2019.

[67] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Deeper, broader and artier domain generalization. In *Proceedings of the IEEE international conference on computer vision*, pages 5542–5550, 2017.

[68] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rkgz2aEKDr`.

[69] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[70] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[71] Andreas Madsen and Alexander Rosenberg Johansen. Neural arithmetic units. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=H1gNOeHKPS`.

[72] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJgMlhRctm`.

[73] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJgMlhRctm`.

[74] Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017.

[75] Daphne Maurer. How the baby learns to see: Donald o. hebb award lecture, canadian society for brain, behaviour, and cognitive science, ottawa, june 2015. *Canadian Journal of Experimental Psychology/Revue canadienne de psychologie expérimentale*, 70(3):195, 2016.

[76] Keith McGreggor and Ashok Goel. Confident reasoning on raven's progressive matrices tests. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 380–386. AAAI Press, 2014.

[77] Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In *International Conference on Machine Learning*, pages 1791–1799, 2014.

[78] Judea Pearl and Dana Mackenzie. *The book of why: the new science of cause and effect*. Basic Books, 2018.

[79] John Raven. The raven's progressive matrices: change and stability over culture and time. *Cognitive psychology*, 41(1):1–48, 2000.

[80] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? In *International Conference on Machine Learning*, pages 5389–5400, 2019.

[81] Mengye Ren, Ryan Kiros, and Richard Zemel. Exploring models and data for image question answering. In *Advances in neural information processing systems*, pages 2953–2961, 2015.

[82] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[83] David Rolnick, Andreas Veit, Serge Belongie, and Nir Shavit. Deep learning is robust to massive label noise. *arXiv preprint arXiv:1705.10694*, 2017.

[84] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[85] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4967–4976, 2017.

[86] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493, 2018.

[87] Yuri Sato, Sayako Masuda, Yoshiaki Someya, Takeo Tsujii, and Shigeru Watanabe. An fmri analysis of the efficacy of euler diagrams in logical reasoning. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 143–151. IEEE, 2015.

[88] Yuri Sato, Yuichiro Wajima, and Kazuhiro Ueda. Strategy analysis of non-consequence inference with euler diagrams. *Journal of Logic, Language and Information*, 27(1):61–77, 2018.

[89] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3): 93–93, 2008.

[90] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[91] Atsushi Shimojima. *Semantic properties of diagrams and their cognitive potentials*. Center for the Study of Language & Information, 2015.

[92] Elizabeth S Spelke and Katherine D Kinzler. Core knowledge. *Developmental science*, 10(1):89–96, 2007.

[93] Gem Stapleton, Judith Masthoff, Jean Flower, Andrew Fish, and Jane Southern. Automated theorem proving in euler diagram systems. *Journal of Automated Reasoning*, 39(4):431–470, Dec 2007. ISSN 1573-0670. doi: 10.1007/s10817-007-9069-y. URL `https://doi.org/10.1007/s10817-007-9069-y`.

[94] Xander Steenbrugge, Sam Leroux, Tim Verbelen, and Bart Dhoedt. Improving generalization for abstract reasoning tasks using disentangled feature representations. *arXiv preprint arXiv:1811.04784*, 2018.

[95] Christopher Summerfield, Fabrice Luyckx, and Hannah Sheahan. Structure learning and the posterior parietal cortex. *Progress in neurobiology*, 184:101717, 2020.

[96] Damien Teney, Lingqiao Liu, and Anton van den Hengel. Graph-structured representations for visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2017.

[97] Andrew Trask, Felix Hill, Scott E Reed, Jack Rae, Chris Dyer, and Phil Blunsom. Neural arithmetic logic units. In *Advances in Neural Information Processing Systems*, pages 8035–8044, 2018.

[98] M. Urbas and M. Jamnik. Heterogeneous proofs: Spider diagrams meet higher-order provers. In M. van Eekelen, H. Geuvers, J. Schmaltz, and F. Wiedijk, editors, *ITP*, volume 6898, pages 376–382. Springer, 2011.

[99] Matej Urbas, Mateja Jamnik, and Gem Stapleton. Speedith: a reasoner for spider diagrams. *Journal of Logic, Language and Information*, 24(4):487–540, 2015.

[100] Aaron van den Oord, Oriol Vinyals, et al. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pages 6306–6315, 2017.

[101] Sjoerd van Steenkiste, Francesco Locatello, Jürgen Schmidhuber, and Olivier Bachem. Are disentangled representations helpful for abstract visual reasoning? In *Advances in Neural Information Processing Systems*, pages 14222–14235, 2019.

[102] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[103] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`.

[104] Huy V Vo, Francis Bach, Minsu Cho, Kai Han, Yann LeCun, Patrick Pérez, and Jean Ponce. Unsupervised image matching and object discovery as optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8287–8296, 2019.

[105] D. Wang, M. Jamnik, and P. Lio'. Unsupervised and interpretable scene discovery with discrete-attend-infer-repeat. In *ICML workshop on Self-Supervised Learning*, page 9pp, 2019. URL `https://arxiv.org/abs/1903.06581`.

[106] Duo Wang, Mateja Jamnik, and Pietro Liò. Investigating diagrammatic reasoning with deep neural networks. In Peter Chapman, Gem Stapleton, Amirouche Moktefi, Sarah Perez-Kriz, and Francesco Bellucci, editors, *International Conference on Theory and Application of Diagrams*, LNCS 10871, pages 390–398. Springer, 2018. URL `https://link.springer.com/chapter/10.1007/978-3-319-91376-6_36`.

[107] Duo Wang, Mateja Jamnik, and Pietro Lio. Unsupervised extraction of interpretable graph representations from multiple-object scenes. *International Conference of Machine Learning, Learning and Reasoning with Graph-Structured Representations Workshop*, 2019. URL `https://graphreason.github.io/papers/20.pdf`.

[108] Duo Wang, Mateja Jamnik, and Pietro Lio. Unsupervised and interpretable scene discovery with discrete-attend-infer-repeat. *International Conference of Machine Learning, Self-Supervised Learning Workshop*, 2019. URL `https://drive.google.com/file/d/0B4M2lUVyJzS4d29IN2pyUDR5ME8zV0RzdlJtZGdzM2xLV2Vv/view`.

[109] Duo Wang, Mateja Jamnik, and Pietro Lio. Abstract diagrammatic reasoning with multiplex graph networks. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=ByxQB1BKwH`.

[110] Duo Wang, Mateja Jamnik, and Pietro Lio. Generalisable relational reasoning with comparators in low-dimensional manifolds. *Arxiv Preprint*, 2020. URL `https://arxiv.org/abs/2006.08698`.

[111] Haohan Wang, Zexue He, and Eric P. Xing. Learning robust representations by projecting superficial statistics out. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJEjjoR9K7`.

[112] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rJxbJeHFPS`.

[113] Daniel LK Yamins and James J DiCarlo. Using goal-driven deep learning models to understand sensory cortex. *Nature neuroscience*, 19(3):356–365, 2016.

[114] Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Josh Tenenbaum. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. In *Advances in Neural Information Processing Systems*, pages 1031–1042, 2018.

[115] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*, pages 4800–4810, 2018.

[116] Anthony M Zador. A critique of pure learning and what artificial neural networks can learn from animal brains. *Nature communications*, 10(1):1–7, 2019.

[117] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.

[118] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535. IEEE, 2010.

[119] Rowan Zellers, Mark Yatskar, Sam Thomson, and Yejin Choi. Neural motifs: Scene graph parsing with global context. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5831–5840, 2018.

[120] Chi Zhang, Feng Gao, Baoxiong Jia, Yixin Zhu, and Song-Chun Zhu. Raven: A dataset for relational and analogical visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5317–5327, 2019.

[121] Chi Zhang, Baoxiong Jia, Feng Gao, Yixin Zhu, Hongjing Lu, and Song-Chun Zhu. Learning perceptual inference by contrasting. In *Advances in Neural Information Processing Systems*, pages 1073–1085, 2019.

[122] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

Appendix A

EulerNet

A.1 Architecture Configurations

In this section, I present detailed architecture configurations for reproducibility. Please refer to Figure 3.1 for an overview of the EulerNet architecture. Table A.1 shows the detailed configuration for Siamese CNN used in EulerNet, while Table A.2 shows the detailed configuration for the reasoning network.

Layer	Configuration
Conv-1	Kernel $3 \times 3 \times 64$; Batch-Norm; Activation: ReLU
Conv-2	Kernel $3 \times 3 \times 64$; Batch-Norm; Activation: ReLU
Pool-1	Max-Pooling 2×2
Conv-3	Kernel $3 \times 3 \times 64$; Batch-Norm; Activation: ReLU
Conv-4	Kernel $3 \times 3 \times 64$; Batch-Norm; Activation: ReLU
Pool-2	Max-Pooling 2×2
Conv-5	Kernel $3 \times 3 \times 32$; Batch-Norm; Activation: ReLU
Conv-6	Kernel $3 \times 3 \times 32$; Batch-Norm; Activation: ReLU
Pool-3	Max-Pooling 2×2
Conv-7	Kernel $3 \times 3 \times 32$; Batch-Norm; Activation: ReLU

Table A.1: Euler-Net: Siamese CNN configuration. Kernel confiugrations are in the format $Width \times Height \times Depth$

Layer	Configuration
FC-1	128 Hidden Units; BatchNorm; Activation: ReLU
Dropout-1	Dropout Rate:0.3
FC-2	64 Hidden Units; BatchNorm; Activation: ReLU
Dropout-2	Dropout Rate:0.3
FC-3	4 Output Units; Activation: Sigmoid

Table A.2: Euler-Net: Reasoning Network configuration. "FC" denotes fully connected layers.

I also present detailed configurations for the GAN network used for Euler diagram generation, as shown in Figure 3.2. Table A.3 shows the configuration of the generator network while Table A.4 shows the configuration of the discriminator network.

Layer	Configuration
ConvT-1	Kernel $4 \times 4 \times 256$; Stride 4; Batch-Norm; Activation: ReLU
ConvT-2	Kernel $4 \times 4 \times 256$; Stride 2; Batch-Norm; Activation: ReLU
ConvT-3	Kernel $4 \times 4 \times 128$; Stride 2; Batch-Norm; Activation: ReLU
ConvT-2	Kernel $4 \times 4 \times 64$; Stride 2; Batch-Norm; Activation: ReLU
ConvT-2	Kernel $4 \times 4 \times 3$; Stride 2; Activation: Sigmoid

Table A.3: Euler-Net: Generator network configuration. "ConvT" means transposed convolution.

Layer	Configuration
Conv-1	Kernel $4 \times 4 \times 32$; Stride 2; Batch-Norm; Activation: ReLU
Conv-2	Kernel $4 \times 4 \times 64$; Stride 2; Batch-Norm; Activation: ReLU
Conv-3	Kernel $4 \times 4 \times 128$; Stride 2; Batch-Norm; Activation: ReLU
Conv-4	Kernel $4 \times 4 \times 256$; Stride 2; Batch-Norm; Activation: ReLU
FC1	Kernel 512 Hidden Units; Batch-Norm; Activation: ReLU
FC2	Kernel 1 Output Units; Activation: Sigmoid

Table A.4: Euler-Net: Discriminator network configuration.

A.2 Hyper-Parameters

I use Adam Optimizer to train EulerNet with TensorFlow package. I set the learning rate as 0.001, β as $(0.9, 0.999)$ and batch size as 128. I also add L2 regularisation of $1e^{-5}$ to all layers. During training I additionally use the following data augmentations:

- Random vertical and horizontal flip
- Random vertical and horizontal shift equal to 10% of the image height and width.
- Random rotation in the range of $(-90, 90)$ degrees.

Appendix B

MXGNet

B.1 Architecture

In this section I present exact configurations of all model variants of MXGNet. Due to the complexity of architectures, I will describe each module in sequence. The object-level representation has two variations which are (o1) CNN features and (o2) Spatial Attention features. Furthermore, the models for PGM and RAVEN datasets differ in detail. Unless otherwise stated, in all layers I apply Batch Normalisation [48] and use Rectified Linear Unit as activation function.

B.1.1 Object-Level Representation Architecture

CNN features: The first approach applies a CNN on the input image and use each spatial location in the final CNN feature map as the object feature vector. This type of representation is used widely, such as in Relation Network [85] and VQ-VAE [100]. Formally, the output of a CNN is a feature map tensor of dimension $H \times W \times D$ where H, W and D are respectively height, width and depth of the feature map. At each H and W location, an object vector is extracted. This type of object representation is simple and fast, but does not guarantee that the receptive field at each feature map location fully covers objects in the image.

We use a residual module [39] with two residual blocks to extract CNN features, as shown in Figure B.1. This is because residual connections show better performance in experiments. The structure of a single Residual Convolution Block is shown in Figure B.2. Unless otherwise stated, the convolutional layers in residual blocks have a kernel size of 3×3. The output feature map processed by another residual block is treated as background encoding because we found that convolutional background encoding gives better results than feature vectors.

Spatial Attention features: The second approach is to use spatial attention to

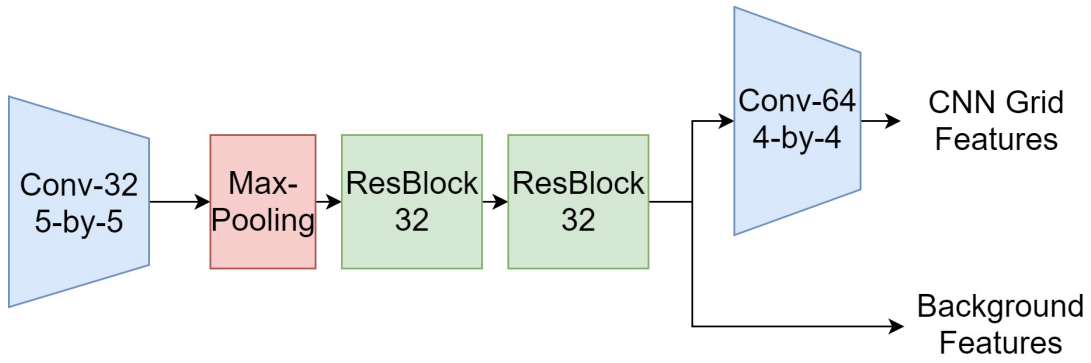


Figure B.1: CNN feature object-level representation module. 'Conv' is convolution layers, 'Max-Pooling' is max-pooling layer and 'ResConv Block' is Residual Convolutional Block.

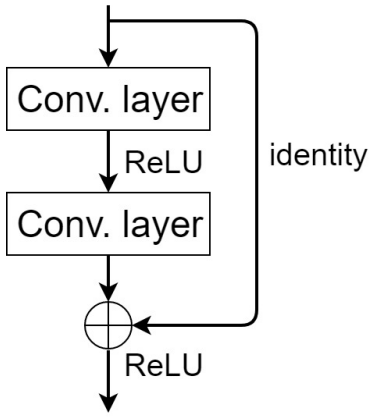


Figure B.2: Architecture of a single Residual Convolution Block.

attend to locations of objects, and extract representations for each object attended. This is similar to object detection models such as faster R-CNN [82], which use a Region Proposal Network to propose bounding boxes of objects in the input image. In practice, we use Spatial Transformer [49] as our spatial attention module. Figure B.3 shows the architecture used for extracting object-level representation using spatial attention. A CNN composed of 1 convolutional layer and 2 residual blocks is first applied to the input image, and the last layer feature map is extracted. This part is the same as in the CNN grid feature module. A spatial attention network composed of 2 convolutional layers then processes information at each spatial location on the feature map, and outputs k numbers of $z = (z^{pres}, z^{where})$, corresponding to k possible objects at each location. Here, z^{pres} is a binary value indicating if an object exists in this location, and z^{where} is an affine transformation matrix specifying a sampling region on the feature maps. z^{pres}, the binary variable, is sampled from Gumbel-Sigmoid distribution [70, 52], which approximates the Bernoulli distribution. We set Gumbel temperature to 0.7 throughout the experiments. For the PGM dataset, we restricted k to be 1 and z^{where} to be a translation and scaling

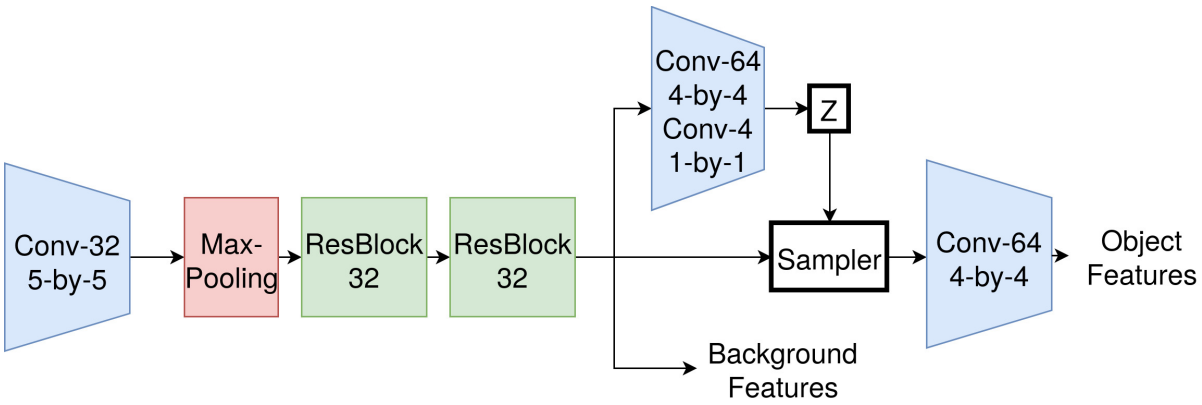


Figure B.3: Spatial attention based feature object-level representation module. 'Conv' is convolution layers, 'Max-Pooling' is max-pooling layer and 'ResConv Block' is Residual Convolutional Block. z is the spatial attention variable (z^{pres}, z^{where}). Sampler is a grid sampler which samples a grid of points from given feature maps.

matrix as 'shapes' objects do not overlap and do not have affine transformation attributes other than scaling and translation. For all $z_i; i \subset [1, H \times W]$, if z_i^{pres} is 1, an object encoder network samples a patch from location specified by z_i^{where} using a grid sampler with a fixed window size of 4×4 pixels. More details of the grid sampler can be found in [49]. The sampled patches are then processed by a conv-layer to generate object embeddings.

B.1.2 Graph networks

Multiplex Edge Embeddings: Figure 4.2 in Chapter 4 on page 58 shows an overview of the multiplex graph architecture. While the motivation and the overview of the architecture are explained in Section 4.1.2, in this section we provide exact configurations for each part of the model. Each sub-layer of the multiplex edge is embedded by a small MLP. For the PGM dataset, we use 6 parallel layers for each multiplex edge embedding, with each layer having 32 hidden units and 8 output units. For the RAVEN dataset, we use 4 layers with 16 hidden units and 8 output units because the RAVEN dataset contains fewer relations types than PGM dataset. The gating function is implemented as one Sigmoid fully connected layer with a hidden size equal to the length of concatenated aggregated embeddings. Gating variables are element-wise multiplied with concatenated embeddings for gating effects. Gated embeddings are then processed with a final fully connected layer with hidden size 64.

Graph Summarisation: This module summarises all node summary embeddings and background embeddings to produce a diagram subset embedding representing relations present in the set of diagrams. I experimented with various approaches and found that keeping embeddings as feature maps and processing them with residual blocks yields the best results. Background feature map embeddings are generated with one

additional residual block of 48 on top of lower layer feature-extracting resnet. For object representations obtained from CNN-grid features, node embeddings can be simply reshaped into a feature map, and processed with additional conv-nets to generate a feature map embedding of the same dimension to background feature map embeddings. For object representations with spatial attention, another Spatial Transformer can be used to write node summary embeddings to its corresponding locations on a canvas feature map. Finally, I concatenate node summary embeddings and background embeddings and process the concatenated embeddings with 2 residual blocks of size 64 to produce the relation embeddings.

B.2 Training details

The architecture is implemented in Pytorch framework. During training, I used RAdam optimizer [68] with learning rate 0.0001, $\beta_1 = 0.9$,$\beta_2 = 0.999$. I used batch size of 64, and distributed the training across 2 Nvidia Geforce Titan X GPUs. I early-stop training when validation accuracy stops increasing.

B.3 More details on search space reduction

In this section I provide a detailed architecture used for search space reduction, and present additional experimental results.

The node embeddings are generated by applying a Conv-Net of 4 convolutional layers (32 filters in each layer) of kernel size 3, and a fully connected layer mapping flattened final-layer feature maps to a feature vector of size 256. Edge embeddings are generated by a 3-layer MLP of $512 - 512 - 256$ hidden units. Subset embeddings are generated by a fully connected layer of 512 units. The subset embeddings are gated with the gating variables and summed into a feature vector, which is then fed into the reasoning net, a 3-layer MLP with $256 - 256 - 13$. The output layer contains 13 units. The first unit gives the probability of the currently combined answer choice being true. The remaining 12 units give meta-target prediction probabilities. This is the same as in [5]. The training loss function is:

$$\mathcal{L} = \mathcal{L}_{ans} + \beta \mathcal{L}_{meta-target} + \lambda \left\| \sum_{(i,j,k) \subset S} G_{i,j,k} \right\|_{L1} \tag{B.1}$$

In the experiment, I tested various values of λ, and found 0.01 to be the best. This model is trained with RAdam optimiser with a learning rate of 0.0001 and batch size of 64. After 10 epochs of training, only gating variables of subsets that are rows and columns are above the 0.5 threshold. The Gating variables for the three rows are 0.884, 0.812 and 0.832.

The gating variables for the three columns are 0.901, 0.845 and 0.854. All other gating variables are below 0.5. Among these, the one with the highest absolute value is 0.411. Table B.1 shows the top-16 ranked subsets, with each subset indexed by 2 connecting edges in the subset. Figure B.4 illustrates the indexing of the subset. For example, the first column with red inter-connecting arrows is indexed as 0-3-6. This indicates that there are two edges, one connecting diagrams 0 and 3, and the other connecting diagram 3 and 6. Similarly, the subset connected by blue arrows is indexed as 1-2-5. Note that 1-2-5 and 2-1-5 are different because the 1-2-5 contains edges 1-2 and 2-5 while 2-1-5 contains edges 2-1 and 1-5.

Rank	Diagram subsets	$\|GatingVariable\|$
1	0-3-6	0.901
2	0-1-2	0.884
3	2-5-8	0.854
4	1-4-7	0.845
5	6-7-8	0.832
6	3-4-5	0.812
7	1-2-5	0.411
8	2-1-5	0.384
9	3-6-7	0.381
10	3-7-4	0.364
11	6-3-7	0.360
12	1-5-4	0.357
13	0-4-6	0.285
14	3-4-7	0.282
15	1-3-4	0.273
16	1-4-5	0.271

Table B.1: All subsets ranked by the absolute value of their corresponding gating variables.

B.4 Ablation study

I performed ablation study experiments to test how much the multiplex edges affect performance. I tested two model variants, one without any graph modules, and the other model with graphs using vanilla edge embeddings produced by MLPs, on the PGM dataset. I found that without graph modules, the model only achieved 83.2% test accuracy. While this is lower than MXGNet's 89.6%, it is still higher than WReN's 76.9%. This is possibly because the search space reduction, by trimming away non-contributing subsets, allows the model to learn more efficiently. The graph model with vanilla edge embeddings achieves 88.3% accuracy, only slightly lower than MXGNet with multiplex edge embeddings. This shows that while a general graph neural network is a suitable model for capturing relations

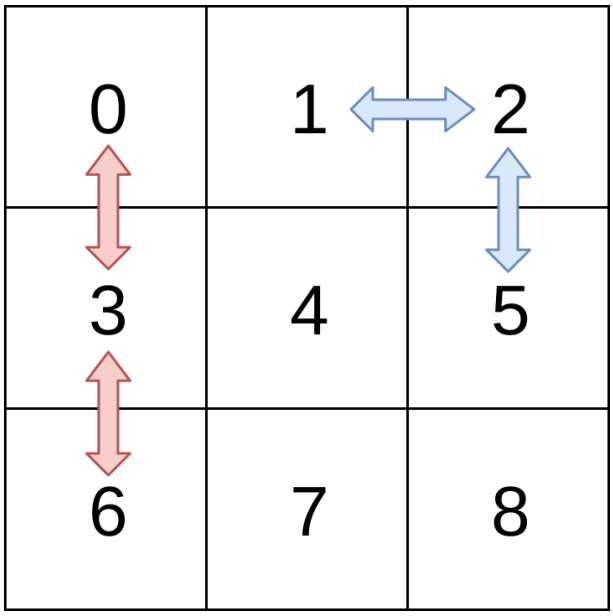


Figure B.4: Illustration of diagram ordering in the matrix and numbered representation of subsets.

between objects, the multiplex edge embedding does so more efficiently by allowing parallel relation multiplexing.

Appendix C

Discrete AIR

C.1 Details of architecture and training

C.1.1 Architecture

I use PyTorch package[1] for Python to build the neural network model. I describe below the details of each module in discrete-AIR.

Embedding: The embedding module embeds the difference image between the input and the previous canvas C^{t-1} into a feature vector which is input to the RNN module. For Multi-Sprites, I use a Convolutional Neural Network as the embedding module. The Conv-Net contains three convolutional layers with 16, 24, and 32 filters respectively. All layers have a kernel size of 5. Batch Normalisation is used for all three layers. After each Convolutional layer a max-pooling layer of pool size 2×2 is applied. For Multi-MNIST dataset, I did not use an embedding layer, as in the original AIR model, but feed the difference image directly to RNN.

RNN: RNN module takes embedded difference image, together with latent codes at the previous time step z^{t_1} as input, and generates spatial attention variables z^t_{where} and presence variable z^t_{pres} for the current step. The RNN is implemented as Long Short-Term Memory (LSTM) [42]. The LSTM module has 256 recurrent units. I perform gradient norm clipping to stabilise the training of RNN.

Read Attention: Read attention module takes the input image and the spatial attention parameters z_{where}. I use two consecutive Spatial Transformers [49], one for scaling and translation and the other for rotation and shearing combined. For the Multi-Sprites dataset, I did not include shearing for simplicity.

Encoder: The encoder takes the spatially attended image patch and encodes it into a set of latent codes. This is implemented as a CNN of 3 layers with 48, 64 and 96 number of filters of kernel size 5 respectively. Batch Normalisation is used for all three layers. I do

[1] `https://pytorch.org/`

not use a max-pooling layer, but instead set the stride of each convolutional layer to be 2.

Decoder: The decoder decodes latent codes into a reconstructed image patch. I use additive function to combine z_{cat} and z_{attr} as discussed in Section 5.2.2. The detailed architecture is shown in Table C.1. For the Multi-Sprites dataset, no attribute variable is used.

z_{cat}	z_{attr}
Fully Connected (128 units)	
Fully Connected (1024 units)	
Resize 1024 to $64 \times 4 \times 4$	
Transposed Conv (64 to 48) Batch-Norm	
Transposed Conv (48 to 32) Batch-Norm	
Transposed Conv (32 to 1)	

Table C.1: Decoder architecture.

Write Attention: The Write Attention Module takes generated objects and z_{where} and write the objects onto the canvas with two Spatial transformers with inverse transformation matrices of those in the Read Attention Module.

Canvas function: I used an additive canvas function $C^t = C^{t-1} + O^t \otimes z^t_{pres}$ where O^t, the generated object, is gated by the presence variable z_{pres}.

C.1.2 Training

Optimiser: ADAM
Learning rate: 0.0001
Batch size: 64
Temperature annealing scheme: $\tau = \max(0.5, e^{-rt})$ where t is the number of training iterations and r is the anneal rate. I set r to be 0.005 for both experiments.
Training epochs: I trained Multi-Sprites for 300 epochs and Multi-MNIST for 420 epochs. This is determined when the reconstruction loss is not improving for 10 consecutive epochs.

C.2 Building Multi-Sprites dataset

I built the Multi-Sprites dataset using the pseudo code in Algorithm C.1:

C.3 Analysis of failures

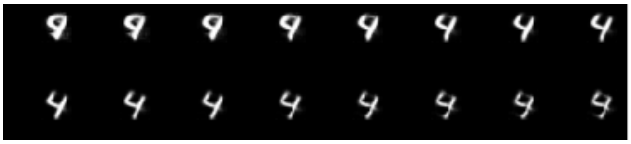
For the Multi-Sprites dataset, the model performs less well when objects are too small. This is because small objects only occupy the space of tens of pixels and are thus heavily

Algorithm C.1 Multi-Sprites Dataset Generation

Input: num to generate N, num max objects M
for $i = 1$ **to** N **do**
  n_{obj} = RandomInt(M)
  List of objects L initialised
  for $j = 1$ **to** n_{obj} **do**
    $Category$ = RandomInt(num of classes)
    x = RandomUniform(x_{min},x_{max})
    y = RandomUniform(y_{min},y_{max})
    $angle$ = RandomUniform(-π,π)
    O = generate($Category, x, y, angle$)
    while Overlap(O,L) > threshold **do**
      Repeat generation
    end while
    Append O in L
  end for
  Image = paint(L)
end for

sub-sampled in the rasterisation process. This causes the difference between different shapes to be indistinguishable.

For the Multi-MNIST dataset, the model occasionally learns to use one category for two similar-looking digits by encoding the difference in the attribute variable, thereby causing a considerable drop in the correspondence rate. Figure C.1 illustrates the case between digits 4 and 9 and digits 3 and 8.



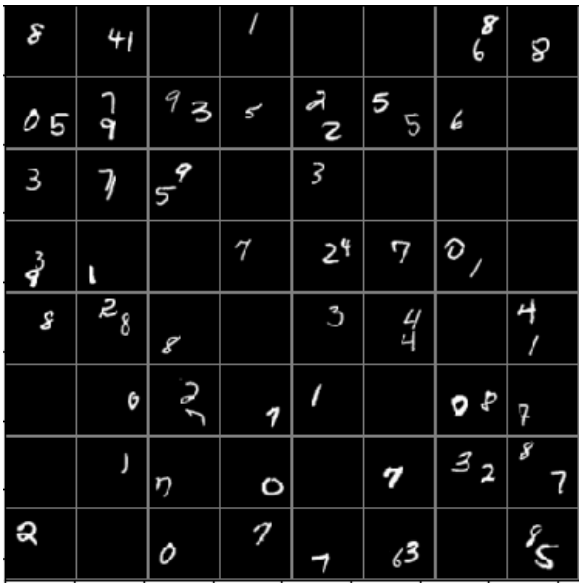
(a) 4 and 9

(b) 3 and 8

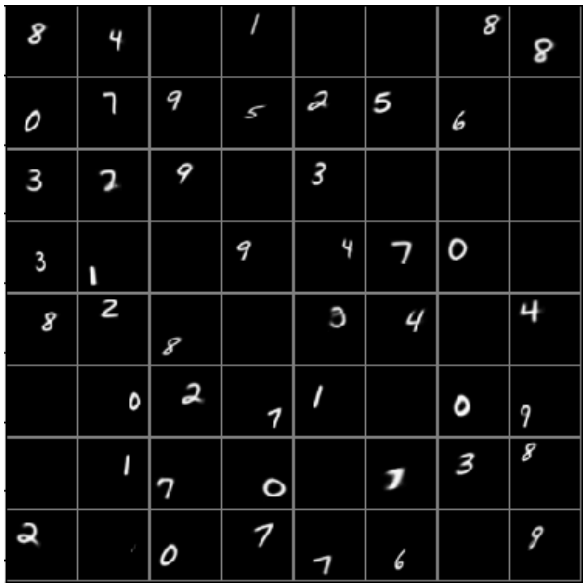
Figure C.1: Illustration of Discrete-AIR model occasionally squeeze two digits into one category.

C.4 Some more reconstructions

Figure C.2 shows additional samples of step-by-step reconstruction for the Multi-MNIST dataset. Figure C.3 shows this for the Multi-Sprites dataset.



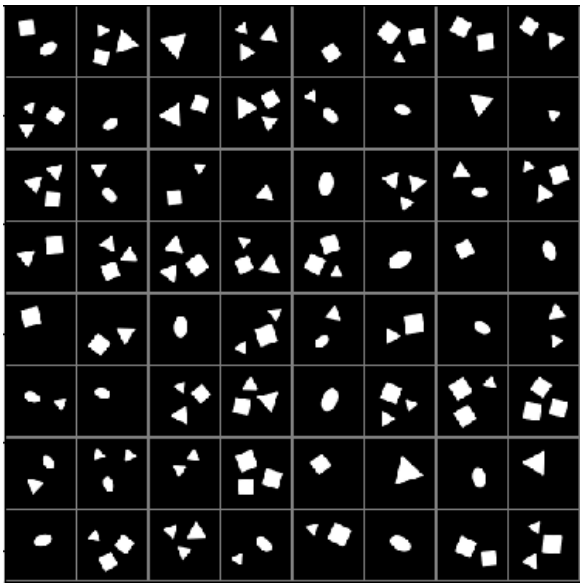
(a) original input



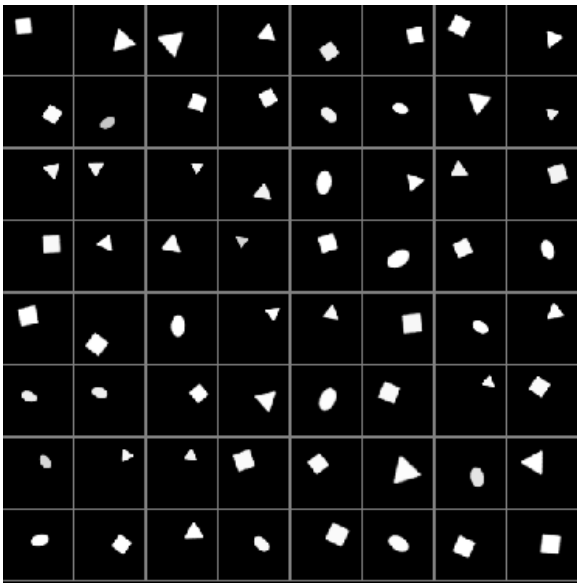
(b) 1st step recon

(c) 2nd step recon

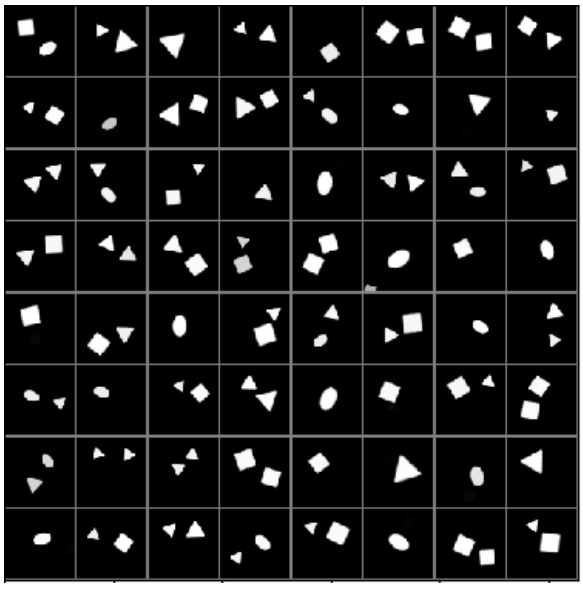
Figure C.2: Illustration of reconstruction for Multi-MNIST.



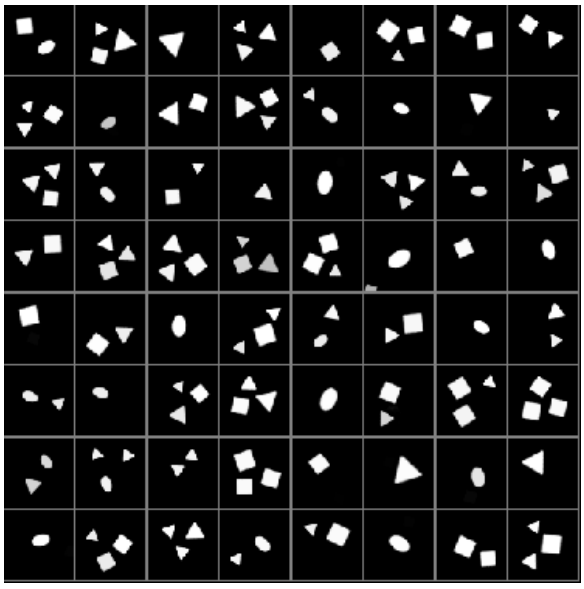
(a) original input



(b) 1st step recon



(c) 2nd step recon



(d) 3rd step recon

Figure C.3: Illustration of the reconstruction for Multi-Sprites.

Appendix D

Generalisable Relational Reasoning

D.1 Maximum of a set: architecture configurations

The architecture for the maximum of a set task has three sub-modules, namely a comparator $f(x_i, x_j)$, a comparison summariser $e_i = MLP(\sum_{j=1}^{N} f(x_i, x_j))$ and a pooling function $\sum_{i=1}^{N} a(e_i)p(x_i)$. For comparator f, I set K, the number of parallel comparisons (Equation 6.1 on page 88) to be 1 because the scalar valued real numbers do not have multiple parallel attributes. I implement the projection function p as a single feed forward layer. I choose 1-dimensional comparison space as this gives the best result. The comparison function c takes the projected difference $p(x_i) - p(x_j)$ as input, and is implemented as a single feed forward layer with 1 output unit. The MLP in the comparison summariser is implemented as a 2-layer MLP of hidden-size $16 - 1$. In the pooling function, the attention function a is implemented as a softmax layer which normalises e_i across $i \in 1 \dots N$.

D.2 Visual object comparison: dataset generation

I now describe details of the visual object comparison dataset. I sample images from the dSprites dataset [41] and generate comparison labels (categories include smaller than, equal to, greater than) from ground truth latent values provided in the dataset. For each image in the dSprites dataset, 5 ground truth attribute values are provided, which are shape "category", "size", "rotation angle", "horizontal position" and "vertical position". I add "colour" as the 6th attribute by randomly generating colour intensity value in the range [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0] for each image. I multiply image pixel values with the colour intensity, and add the colour intensity value to the ground truth latent values.

Algorithm D.1 shows the pseudo-code for generating the visual object comparison dataset. compare_attr indicates the object attribute to be compared for the task. I pick three different attributes for experiments, which are "size", "horizontal position" and "colour

intensity". I set the training attribute range to be the lower 60% while the test attribute range to be the upper 40%.

Algorithm D.1 Visual Object Comparison Dataset Generation

```
Input: train_size, test_size, compare_attr
train_data = EmptyList()
test_data = EmptyList()
for split in {train,test} do
  for i = 1 to train_size do
    attr_range, attr_idx = attr_stats(compare_attr)
    if split == train then
      LO, HI = 0, 0.6
    else
      LO, HI = 0.6, 1.0
    end if
    sample_range = Truncate(attr_range, LO, HI)
    latent_values_A = SampleLatent(attr_idx, sample_range)
    latent_values_B = SampleLatent(attr_idx, sample_range)
    image_A = SampleImage(latent_values_A)
    image_B = SampleImage(latent_values_B)
    less_than = latent_values_A < latent_values_B
    equal = latent_values_A == latent_values_B
    greater_than = latent_values_A > latent_values_B
    label = Concat(less_than, equal, greater_than)
    if split == train then
      train_data ← (image_A,image_B,label)
    else
      test_data ← (image_A,image_B,label)
    end if
  end for
end for
```

D.3 Visual object comparison: architecture configurations

In this section, I list a detailed configuration of the architecture with low-dim comparators and the baseline architecture. The architecture with low-dim comparator is illustrated in Figure 6.3a on page 91. The CNN module is a 4-layer CNN of with 32, 32, 64 and 64

number of filters respectively, followed by a 2-layer MLP of hidden size $256 - 10$. Each CNN layer has a stride value of 2 and a padding value of 1. The output of each CNN is the object representation o_i of raw image input x_i. The projection module p is a single feed-forward layer projecting to a space with dimension d. I use $d = 1$ for reporting results except for the manifold analysis experiments in Section 6.3.4. The comparator takes the projected vector difference $p(o_i) - p(o_j)$ as input, and is implemented as a 2-layer MLP of size $h - 3$, where h is the hidden size and 3 is the output size (corresponding to 3 different output categories). I found that varying h in the range from 4 to 32 has little effect on the performance. Hence I report performance with $h = 4$ to reduce computational costs.

The baseline architecture uses the same CNN module as the architecture with low-dim comparators. The baseline model concatenates the CNN output o_i and o_j and feeds the concatenated vector into an MLP. I performed a hyper-parameter search of the MLP with the number of layers ranging from 1 to 4, and with hidden unit sizes from 32 to 64. I found that $64 - 64 - 3$ is the best performing architecture.

D.4 PGM architecture configurations

In this section, I describe the detailed configuration of the PGM architecture with low-dim comparators, and the baseline model MLRN-P, which is an augmented version of MLRN [50]. The descriptions here is supplementary to descriptions in Section 6.2.4 and Figure 6.3b.

The CNN module is a 4-layer CNN with 32, 32, 64 and 64 number of filters respectively, followed by a 2-layer MLP of hidden size $256 - 128$. Each CNN layer has a stride value of 2 and a padding value of 1. The output of each CNN is the object representation o_i of raw image input x_i. Following [5], I attach to o_i a position tag to indicate its position in the diagram matrix. The tagged object representation is then projected onto $K = 512$ 1-dimensional manifolds for parallel attribute comparison. Next I describe the comparator module f (Equation 6.1). As shown in Figure 6.3b on page 91, there are two hierarchical projection comparisons. For the first level, I found that implementing c as a simple vector difference module achieves best results, which means $c = p(o_i) - p(o_j)$. This reflects the fact that the comparison between diagrams is directional, such as an increase in the number of objects from one diagram to the other. I implement g in equation 6.1 as a residual MLP of 4 layers with hidden size $2048 - 2048 - 2048 - 796$. The output from each pairwise comparison of diagrams in a row/column is concatenated to form the relation embedding for that row/column. For the second level I implement c as the absolute difference, which means $c = |p(o_i) - p(o_j)|$. This works better because relation comparison is less directional. For example, the difference between the relation of an increasing number of objects and the relation of increasing sizes of objects should be the same when the

compared diagrams are swapped. For the second level, I implement g as a 3 layer MLP of hidden sizes $1024 - 512 - 1$, which directly outputs the predicted similarity score between two rows/columns. For predicting the correct answer candidate, I follow [5] by applying a Softmax function to scores produced by comparing each answer row/column with context rows/columns to produce scores for each answer candidate. For meta-target prediction, I sum all context row/column embeddings and process it with a 3-layer MLP of hidden size $1024 - 512 - 12$, where 12 is the meta-target label size.

The baseline model MLRN-P is modified from MLRN [50]. I have performed three architecture modifications for a fair comparison with my model. Firstly, I inject the prior knowledge of relations existing only in rows/columns into the model. The first level of Relation Networks compares diagrams within rows/columns and the second level of Relation Networks compare row/column embeddings. Secondly, I swap MLP in the original MLRN with residual MLP, which is shown to improve performance slightly in my model. Thirdly I pretrained the CNN module with Beta-VAE [41] for a fair comparison with my model.

D.5 Training details

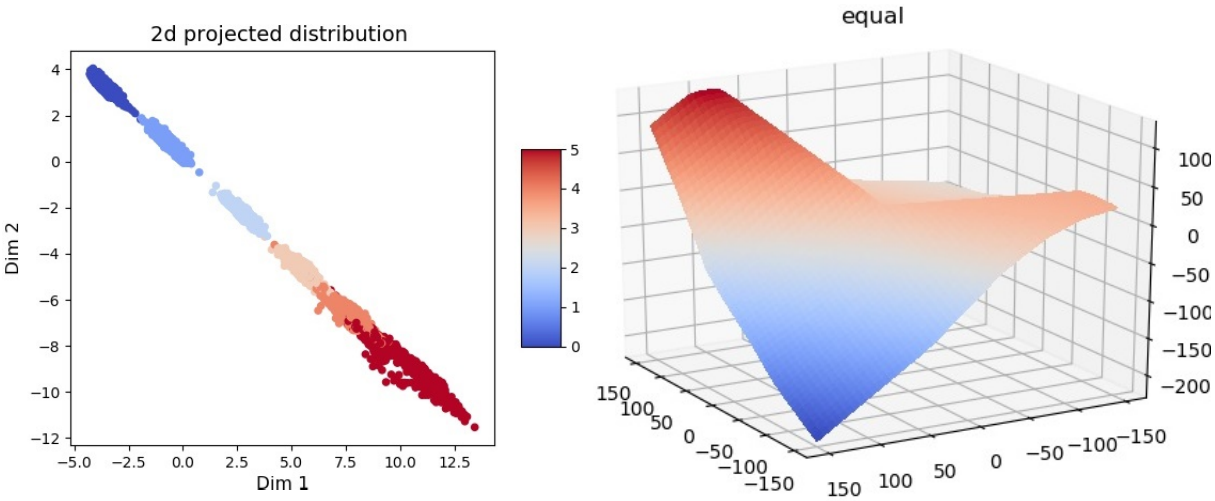
In this section, I describe the training details for all three experiments. I use PyTorch[1] for implementation. For gradient descent optimiser, I use RAdam [68], an improved version of the Adam optimiser. For all 3 experiments, I use the learning rate of 0.001 and the betas (0.9,0.999). We used 2 Nvidia Geforce Titan Xp GPUs for training all models. For "maximum of a set" and "visual object comparison", I set the batch size to be 64. For PGM I found a larger batch size of 512 slightly improves the results. For maximum of the set and visual object comparison, I set training epochs to be 20. For PGM I train for 50 epochs. For visual object comparison and RPM tasks I pre-train CNN as the encoder of a Beta-VAE [41]. I follow standard training procedures of Beta-VAE, and set the β value to be 1.

D.6 Additional plots

In section 6.3.4 I show plots of the projected distribution and the comparator's function landscape for the position comparison task. Here I show the same plots for comparison tasks for other attributes, which are sizes and colour intensity. Figure D.1 shows the plot for size comparison while Figure D.2 shows the plot for colour intensity comparison. The training range is the lower 60% of the colour bar while the test range is the upper 40%. The observations stated in Section 6.3.4 also holds true for these attributes. For size and

[1] `https://pytorch.org/`

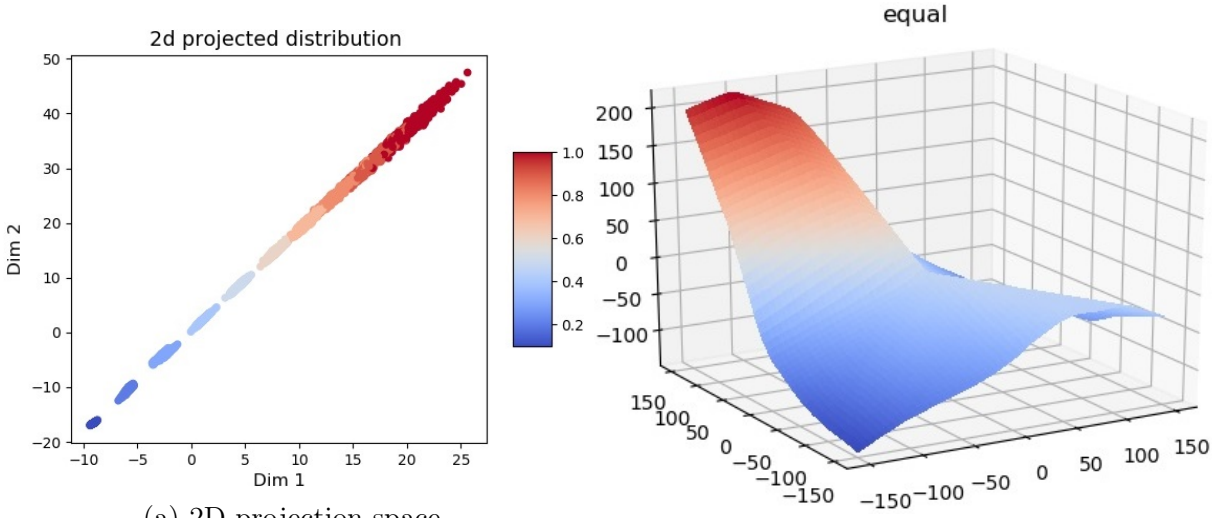
colour intensity comparison tasks, the projected distribution plots show that the test data is more clustered than that of spatial position comparison. This shows that the size and colour intensity attributes can be learnt better with a bespoke CNN perception module than with the spatial position attributes. This is supported by the fact that models trained for these two attributes achieved higher **o.o.d** test accuracies.



(a) 2D projection space

(b) 2D comparator function

Figure D.1: (a) shows a scatter plot of the 2D projected distribution of objects in the task of comparing sizes of objects in the image. X-axis and Y-axis are 2 dimensions of the projection manifold. The latent size variable (possible values are [0,1,2,3,4,5]) is indicated by colour. (b) plots the comparator's function landscape for "equal" output unit (before softmax) in the space of vector differences between 2D projections of objects.



(a) 2D projection space

(b) 2D comparator function

Figure D.2: (a) shows a scatter plot of the 2D projected distribution of objects in the task of comparing colour intensity of objects in the image. X-axis and Y-axis are 2 dimensions of the projection manifold. The latent colour intensity variable (possible values are [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]) is indicated by colour. (b) plots the comparator's function landscape for "equal" output unit (before softmax) in the space of vector differences between 2D projections of objects.