Deep learning with graph-structured representations

Kipf, T.N.

ISBN 978-94-6375-851-2

Deep Learning with Graph-Structured Representations

Thomas Kipf

Deep Learning with Graph-Structured Representations

Thomas Kipf

Deep Learning with Graph-Structured Representations

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. K.I.J. Maex
ten overstaan van een door het College voor Promoties
ingestelde commissie,
in het openbaar te verdedigen
op donderdag 23 april 2020, te 14.00 uur

door

Thomas Norbert Kipf

geboren te Roth

PROMOTIECOMMISSIE

Promotor:

prof. dr. M. Welling	Universiteit van Amsterdam

Copromotor:

dr. I.A. Titov	University of Edinburgh

Overige leden:

prof. dr. M.M. Bronstein	Imperial College London
prof. dr. F.A.H. van Harmelen	Vrije Universiteit Amsterdam
prof. dr. M. de Rijke	Universiteit van Amsterdam
dr. P.W. Battaglia	DeepMind Technologies Ltd
dr. H.C. van Hoof	Universiteit van Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

The work described in this thesis has been primarily carried out at the Amsterdam Machine Learning Lab (AMLab) of the University of Amsterdam and in part during an internship at DeepMind Technologies Ltd, London, UK. The research carried out at the University of Amsterdam was funded by SAP SE.

Printed by Ridderprint, The Netherlands.

ISBN: 978-94-6375-851-2

SUMMARY

In this thesis, *Deep Learning with Graph-Structured Representations*, we propose novel approaches to machine learning with structured data. Our proposed methods are largely based on the theme of structuring the representations and computations of neural network-based models in the form of a graph, which allows for improved generalization when learning from data with both *explicit* and *implicit* modular structure.

Our contributions are as follows:

- We propose ***graph convolutional networks*** (GCNs) (Kipf and Welling, 2017; Chapter 3) for semi-supervised classification of nodes in graph-structured data. GCNs are a form of graph neural network that perform parameterized message-passing operations in a graph, modeled as a first-order approximation to spectral graph convolutions. GCNs achieved state-of-the-art performance in node-level classification tasks in a number of undirected graph datasets at the time of publication.
- We propose ***graph auto-encoders*** (GAEs) (Kipf and Welling, 2016; Chapter 4) for unsupervised learning and link prediction in graph-structured data. GAEs utilize an encoder based on graph neural networks and a decoder that reconstructs links in a graph based on a pairwise scoring function. We further propose a model variant framed as a probabilistic generative model that is trained using variational inference, which we name *variational GAE*. GAEs and variational GAEs are particularly suited for representation learning on graphs in the absence of node labels.
- We propose ***relational GCNs*** (Schlichtkrull and Kipf et al., 2018; Chapter 5) that extend the GCN model to directed, relational graphs with multiple edge types. Relational GCNs are well-suited for modeling relational data and we demonstrate an application to semi-supervised entity classification in knowledge bases.

- We propose ***neural relational inference*** (NRI) (Kipf and Fetaya et al., 2018; Chapter 6) for discovery of latent relational structure in interacting systems. NRI combines graph neural networks with a probabilistic latent variable model over edge types in a graph. We apply NRI to model interacting dynamical systems, such as multi-particle systems in physics.

- We propose ***compositional imitation learning and execution*** (CompILE) (Kipf et al., 2019; Chapter 7), a model for structure discovery in sequential behavioral data. CompILE uses a novel differentiable sequence segmentation mechanism to discover and auto-encode meaningful sub-sequences or sub-programs of behavior in the context of imitation learning. Latent codes can be executed and re-composed to produce novel behavior.

- We propose ***contrastively-trained structured world models*** (C-SWMs) (Kipf et al., 2020; Chapter 8) for learning object-factorized models of environments from raw pixel observations without supervision. C-SWMs use graph neural networks to structure the representation of an environment in the form of a graph, where nodes represent objects and edges represent pairwise relations or interactions under the influence of an action. C-SWMs are trained using contrastive learning without pixel-based losses and are well-suited for learning models of environments with compositional structure.

LIST OF PUBLICATIONS

This thesis is based on the following publications:

- **Thomas N. Kipf** and Max Welling (2017). "Semi-Supervised Classification with Graph Convolutional Networks." In: *International Conference on Learning Representations (ICLR)*.
- **Thomas N. Kipf** and Max Welling (2016). "Variational Graph Auto-Encoders." In: *NeurIPS Workshop on Bayesian Deep Learning*.
- Michael Schlichtkrull*, **Thomas N. Kipf***, Peter Bloem, Rianne van den Berg, Ivan Titov and Max Welling (2018). "Modeling Relational Data with Graph Convolutional Networks." In: *European Semantic Web Conference (ESWC)*.
- **Thomas Kipf***, Ethan Fetaya*, Kuan-Chieh Wang, Max Welling and Richard Zemel (2018). "Neural Relational Inference for Interacting Systems." In: *International Conference on Machine Learning (ICML)*.
- **Thomas Kipf**, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Alvaro Sanchez-Gonzalez, Edward Grefenstette, Pushmeet Kohli and Peter Battaglia (2019). "CompILE: Compositional Imitation Learning and Execution." In: *International Conference on Machine Learning (ICML)*.
- **Thomas Kipf**, Elise van der Pol and Max Welling (2020). "Contrastive Learning of Structured World Models." In: *International Conference on Learning Representations (ICLR)*.

Ideas, text, figures, and experiments originate in majority from the first author with the exception of the papers on *modeling relational data* and *neural relational inference*, where the first two authors (denoted by *) contributed equally to all material. All other authors had important advisory roles, helped with running experiments and/or directly contributed in writing to a small number of individual sections of the above listed papers.

The author has further contributed to the following publications:

- Tim R. Davidson, Luca Falorsi, Nicola De Cao, **Thomas Kipf**, Jakub M. Tomczak (2018). "Hyperspherical Variational Auto-Encoders." In: *Conference on Uncertainty in Artificial Intelligence (UAI)*.
- Rianne van den Berg, **Thomas Kipf**, Max Welling (2018). "Graph Convolutional Matrix Completion." In: *KDD Deep Learning Day*.
- Nicola De Cao, **Thomas Kipf** (2018). "MolGAN: An Implicit Generative Model for Small Molecular Graphs." In: *ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models*.
- Raghavendra Selvan, **Thomas Kipf**, Max Welling, Jesper H. Pedersen, Jens Petersen, Marleen de Bruijne (2018). "Extraction of Airways using Graph Neural Networks." In: *International Conference on Medical Imaging with Deep Learning (MIDL), Abstract Track*.
- Cătălina Cangea, Petar Veličković, Nikola Jovanović, **Thomas Kipf**, Pietro Liò (2018). "Towards Sparse Hierarchical Graph Classifiers." In: *NeurIPS Workshop on Relational Representation Learning*.
- Andreas Kipf, **Thomas Kipf**, Bernhard Radke, Viktor Leis, Peter Boncz, Alfons Kemper (2019). "Learned Cardinalities: Estimating Correlated Joins with Deep Learning." In: *Conference on Innovative Data Systems Research (CIDR)*.
- Andreas Kipf, Dimitri Vorona, Jonas Müller, **Thomas Kipf**, Bernhard Radke, Viktor Leis, Peter Boncz, Thomas Neumann, Alfons Kemper (2019). "Estimating Cardinalities with Deep Sketches." In: *ACM SIGMOD International Conference on Management of Data (SIGMOD), Demo Track*.
- Davide Belli, **Thomas Kipf** (2019). "Image-Conditioned Graph Generation for Road Network Extraction." In: *NeurIPS Workshop on Graph Representation Learning*.
- Elise van der Pol, **Thomas Kipf**, Frans A. Oliehoek, Max Welling (2020). "Plannable Approximations to MDP Homomorphisms: Equivariance under Actions." In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

CONTENTS

II Learning with Implicit Structure

1

INTRODUCTION

1.1 STRUCTURE AND HUMAN COGNITION

Structure is all around us: from fundamental physical interactions to emergent structure such as atoms, molecules, living organisms, societal networks, ecosystems, planetary systems, and many more examples across a wide range of scales in the universe.

Millions[1] of years of evolution in this structured and complex environment have shaped what has become the human mind: a highly optimized system that is adept at navigating this world, at achieving goals in it, and at adapting to the most unforeseeable changes this environment has to offer — in other words, humans are *intelligent agents*.

Not only is the world around us rich in structure, but also our own understanding of the world — our *world model* — is highly structured: we often think, reason, and communicate in terms of concepts, abstractions, and relations. We can identify objects from raw perceptual input and understand visual scenes in terms of their composition as objects and relations. We can take two concepts and relate them to each other. Take the example of an *elephant with wings*: we can easily imagine this combined concept, even though we have likely never encountered such a creature.

How can we formalize this structure? A natural way to represent information in structured form is as a *graph*. A graph is a data structure describing a collection of entities, represented as *nodes*, and their pairwise relationships, represented as *edges*.

1 Or billions, when considering the earliest forms of life on Earth.

With modern technology and the rise of the internet, graphs are everywhere: social networks, the world wide web, street maps, knowledge bases used in search engines, and even chemical molecules are frequently represented as a set of entities and relations between them. The ubiquity of such a graph-structured description of our world calls for the development of effective methods that make use of and learn to understand information represented in this structural form — and while we get there, the lessons learned along the way might help us develop better intelligent agents that share a similar structured understanding of the world as we humans do.

1.2 ARTIFICIAL INTELLIGENCE AND DEEP LEARNING

The desire to understand human cognition has spawned a variety of scientific disciplines. Cognitive science, neuroscience, and the study of machine learning and artificial intelligence (AI) are the most prominent examples of scientific fields that study the human mind (cognitive science), its physical neural substrate (neuroscience), and the replication of some of its behavior and core algorithms in artificial machines (machine learning and AI).

The work in this thesis is situated in the field of machine learning, which is one of the most widely pursued branches in AI research. Machine learning deals with the question of how we can build systems and design algorithms that *learn* from data and experience (e.g., by interacting with an environment), which is in contrast to the traditional approach in computer science where systems are explicitly *programmed* to follow a precisely outlined sequence of instructions. The problem of learning is commonly approached by fitting a *model* to data with the goal that this learned model will generalize to new data or experiences.

Traditionally, many machine learning models are built on top of sets of features, extracted using a pre-defined procedure from the raw data format. For example, such features could be word occurrence statistics in natural language sentences or pixel statistics in image data. The process of developing sophisticated feature extractors is often referred to as *feature engineering*, culminating in the development of popular feature detection algorithms such as SIFT (Lowe, 1999).

Deep learning, an approach that has enjoyed immense popularity in the past decade and in recent years, instead addresses the learning problem by jointly learning representations of the raw input data and a predictive model for the task at hand. This is usually achieved by stacking multiple 'layers' of differentiable non-linear transformations and by training such a model in an *end-to-end* fashion using gradient descent techniques. The resulting class of models is often referred to as *deep neural networks*.

Despite recent successes of deep learning in many areas such as computer vision (e.g., LeCun et al., 1998; Krizhevsky et al., 2012), natural language processing (e.g., Bengio et al., 2003; Vaswani et al., 2017), game playing (e.g., Mnih et al., 2013; Silver et al., 2016), and in the natural sciences (e.g., Dahl et al., 2014; Louppe et al., 2019), deep neural networks still fall short of a general ability for relational and causal reasoning, for conceptual abstraction, and for many other human abilities.

A core problem in machine learning is that of *inductive bias* (Mitchell, 1980): how can we build models that learn the right representations, abstractions, and skills that allow them to generalize to novel and unforeseeable circumstances? Inductive bias in deep neural networks can come in many forms: the choice of model architecture, the training objective, the optimization procedure and even the way in which training data is presented to the model (e.g., using so-called data augmentation) can have a profound impact on generalization to unseen data. The rise in popularity of deep learning was partially enabled by the design of a particular architectural inductive bias: that of convolutional neural networks (CNNs) (LeCun et al., 1998; Krizhevsky et al., 2012).

CNNs utilize a specialized neural network architecture in which model parameters are tied or shared across image locations, thereby exploiting the fact that image feature statistics are often *translation invariant*. This form of parameter sharing equips the model with a useful inductive bias: 'filters' in the CNN model only have to learn about local features and will therefore generalize well across locations in the image. A related inductive bias can be found in recurrent neural networks (RNNs), which share parameters over time steps, and hence generalize more favorably on stationary time series and other sequential data.

In this thesis, we argue for the introduction and design of further structural and compositional[2] inductive biases in deep learning models, to reflect the rich structure in the data and environments that these models often face. One way to achieve this is by structuring the representations and computations in a deep neural network in the form of a graph, leading to a class of models named *graph neural networks* (Gori et al., 2005; Scarselli et al., 2009; Li et al., 2016; Kipf and Welling, 2017; Gilmer et al., 2017; Battaglia et al., 2018), which will be of central importance in this thesis.

1.3 SCOPE AND RESEARCH QUESTIONS

This thesis is structured in two parts: Part 1 will introduce deep neural network models for a variety of learning tasks with explicitly graph-structured data, i.e., datasets that are given to us in the form of entities and their relations. Part 2 will explore the topic of learning with implicit structure in the form of structural and compositional inductive biases, applied to tasks such as sequence modeling, imitation learning, scene understanding, world model learning, and intuitive physics. In this part, we are usually not given an explicitly structured dataset, but we develop models that infer or make use of hidden structure in the data with the goal of achieving improved generalization compared to using unstructured deep models.

The contributions of this thesis are guided by the following research questions:

Research Question 1: *Can we develop and efficiently implement deep neural network-based models for large-scale node classification tasks in graph-structured datasets?*

Our main contribution to address this question is a novel graph neural network model that we call the *graph convolutional network* (GCN). GCNs are introduced in Chapter 3 and in Kipf and Welling (2017). They improve upon earlier work in the community on so-called *spectral graph convolutions*. We introduce a simple yet effective semi-supervised training scheme for GCNs and demonstrate

2 Compositionality here describes that certain data examples can be composed of multiple parts or modules that can be sensibly combined and re-arranged in a number of ways, such as visual scenes containing multiple objects.

significant advantages over earlier state-of-the-art methods both in terms of efficiency and predictive accuracy. We present a relational extension to the GCN model, termed *relational GCN* (R-GCN), in Chapter 5 and in Schlichtkrull and Kipf et al. (2018) to support graphs with different edge types. We demonstrate an application of this model on a knowledge graph with millions of nodes and edges.

Research Question 2: *Can graph neural networks be utilized for link prediction and unsupervised node representation learning?*

We introduce two extensions to the GCN model to address this question in Chapter 4 and in Kipf and Welling (2016): the *graph auto-encoder* (GAE) and a model variant termed the *variational GAE* (VGAE). Both models can be trained on graphs in the absence of node labels, a setting often referred to as *unsupervised node representation learning*. We demonstrate applications of this model class to the task of link prediction in Chapter 4.

Having introduced neural network architectures for explicitly graph-structured data in Part 1 of this thesis (Chapters 3–5), Part 2 will investigate how models with structural and compositional inductive biases — such as graph neural networks — can be developed and applied to problems with implicit or *hidden* structure.

Research Question 3: *Can deep neural networks infer hidden relations and interactions between entities, such as forces in physical systems?*

We introduce the *neural relational inference* (NRI) model in Chapter 6 and in Kipf and Fetaya et al. (2018). NRI is a latent variable model based on a graph neural network with multiple interaction functions. Each pair of nodes is assigned a latent variable which determines the type of interaction between them, and hence the model can be trained to identify hidden interactions or relations. We demonstrate this capability on interacting physical systems and on motion capture data, where NRI can identify hidden interactions and accurately predict future dynamics of the system.

Research Question 4: *How can we improve upon neural network-based models that infer event structure and latent program descriptions in sequential data?*

To address this question, we introduce the CompILE model in Chapter 7 and in Kipf et al. (2019). CompILE stands for *compositional imitation learning and execution* and describes an unsupervised model for discovering task segmentations and hidden task representations in program execution data. CompILE is a latent variable model with a sequence of latent variables, each describing one segment of the input sequence. Latent variables are coupled to their respective segments using a learned 'soft' segmentation mechanism.

Research Question 5: *Can deep neural networks learn to discover and build effective representations of objects, their relations, and effects of actions by interacting with an environment?*

This question aims at the core of what it means to learn a *structured* model of the world by interacting with it. In Chapter 8 and in Kipf et al. (2020) we introduce the *contrastively-trained structured world model* (C-SWM), a deep neural network utilizing a graph neural network component that is capable of discovering object representations and learning about physical interactions between objects in an unsupervised way. The training algorithm is based on *contrastive learning*, which relates to the way we train GAE models for unsupervised node representation learning, but adapted to the case of learning from a dataset of *experiences* from environment interactions. We show that the inductive bias enabled by a graph neural network greatly improves generalization to unseen environment configurations.

Aside from the above listed main contributions of this dissertation, we include an introduction to several background topics in Chapter 2, which will serve as a basis for the material presented in the rest of this thesis. After presenting the main body of our work in Parts 1 and 2, we will conclude and outline interesting directions for future work in Chapter 9.

2

BACKGROUND

In this chapter, we will provide a brief introduction to several background topics and notation that will be extensively used throughout this thesis. Additional background will be introduced where necessary in later chapters. In what follows, we will give an introduction to *deep neural networks* in Section 2.2, to *graph neural networks* in Section 2.3, to *latent variable models* in the context of deep learning in Section 2.4, and lastly to *contrastive learning* in Section 2.5.

2.1 NOTATION

This section provides a reference for the most commonly used notation across this thesis. Individual chapters introduce additional notation where necessary.

Example	**Explanation**
x, y, z	A lowercase italic letter typically denotes a scalar or scalar-valued random variable.
$\mathbf{x}, \mathbf{y}, \mathbf{z}$	A lowercase bold letter typically denotes a vector or vector-valued random variable.
$\mathbf{X}, \mathbf{Y}, \mathbf{Z}$	An uppercase bold letter typically denotes a matrix or matrix-valued random variable.
$\mathcal{X}, \mathcal{Y}, \mathcal{Z}$	Calligraphic letters typically denote sets. An exception is $\mathcal{L}$, which we use to denote scalar-valued objective functions.

Example	Explanation
$\mathbf{I}_N$	The $N \times N$ identity matrix.
$\text{diag}(\mathbf{x})$	A diagonal square matrix with entries on the diagonal populated by the elements of the vector $\mathbf{x}$.
$\mathbb{R}^N$	The N-dimensional real space.
x_i	The i-th element of vector x.
$X_{i,j}$	The i, j-th element of matrix X.
$f_{\boldsymbol{\theta}}(.)$, $f(.;\boldsymbol{\theta})$	Parameter dependency of functions is typically made explicit (unless clear from context) with a greek letter $\boldsymbol{\theta}$ or ϕ.
$f(\mathbf{x}), f(\mathbf{X})$	If $f(.)$ is a function defined on scalars, then $f(\mathbf{x})$ and $f(\mathbf{X})$ are to be understood as an element-wise application of $f(.)$ on the elements of the vector $\mathbf{x}$ or matrix $\mathbf{X}$.
$p(.), q(.)$	Probability density functions (PDFs) or in other words *distributions* are denoted by the lower-case letters $p(.)$ and $q(.)$, where $q(.)$ is typically reserved for *variational* distributions. We use the same notation for probability mass functions (PMFs) of discrete random variables. The same letter will be used for marginals, joint distributions, and conditionals of the same probabilistic model.
$\mathcal{N}(.;\boldsymbol{\mu}, \boldsymbol{\Sigma})$	A multivariate normal (or Gaussian) distribution with a vector of means $\boldsymbol{\mu}$ and a covariance matrix $\boldsymbol{\Sigma}$.

2.2 DEEP NEURAL NETWORKS

For our purposes, we can think of a deep neural network (NN) in the simplest case as a composition of parameterized (affine) linear functions $f^l(.)$, indexed by $l \in \{1, \ldots, L\}$, each followed by an element-wise non-linear function $\sigma(.)$:

$$\text{NN}_{\boldsymbol{\theta}} = f^L \circ \sigma \circ f^{L-1} \circ \cdots \circ \sigma \circ f^2 \circ \sigma \circ f^1 \,, \tag{2.1}$$

where $\circ$ denotes function composition. We will use NNs as function approximators with typically a large number of learnable parameters $\boldsymbol{\theta}$. Deep learning is a vast field and many NN variants have been proposed, differing mostly in the *architecture* of how certain elementary building blocks are combined in a computational graph and how these building blocks are defined in the first place.

Throughout this thesis, we will often make use of the multi-layer perceptron (MLP) (Rosenblatt, 1961), where we will use L to refer to the number of *layers*. In an MLP, we define $f^l(.)$ as follows:

$$f^l(\mathbf{h}) = \mathbf{W}_l\mathbf{h} + \mathbf{b}_l\,, \tag{2.2}$$

where $\mathbf{W}_l$ is the so-called weight matrix of the l-th layer and $\mathbf{b}_l$ is the *bias* vector. Both constitute the set of learnable parameters for a layer. We use $\mathbf{h}$ to denote feature vectors (or hidden representations or embeddings) in a NN. For $\sigma(\mathbf{h})$, we will typically make use of the $\mathrm{ReLU}(\mathbf{h}) = \max(0, \mathbf{h})$ activation function in MLPs.

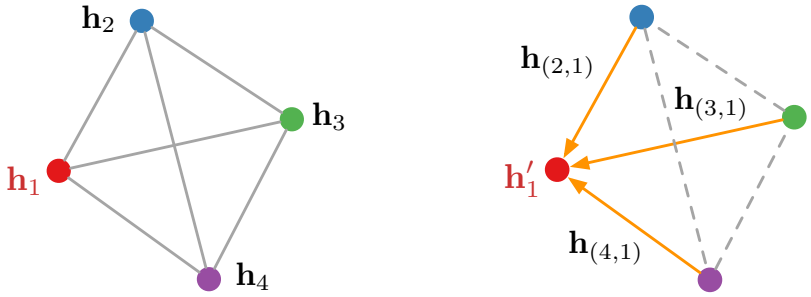
To train NNs, we will make use of backpropagation (Werbos, 1982) and variants of the stochastic gradient descent algorithm such as the Adam optimizer (Kingma and Ba, 2014) to minimize an objective function $\mathcal{L}$.

For details on other common NN variants and building blocks, especially recurrent neural networks and convolutional neural networks, we recommend the 'Deep Learning' book by Goodfellow et al. (2016).

2.3 GRAPH NEURAL NETWORKS

Graph neural networks (GNNs) are a class of NN models suitable for processing graph-structured data, and are of central importance to the topics covered in this thesis. Modern GNNs, as we will introduce them in this section, have been developed after (or concurrently to) our works on which Part 1 of this thesis is based on. They can be seen as a generalization of the model architectures proposed in Part 1, and much of Part 2 will make use of the definition of GNNs that we introduce in this section.

The architecture of a GNN is structured according to a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$. In our notation, nodes are identified by a unique index $i \in \mathcal{V}$ ranging from 1 to $|\mathcal{V}|$, and directed edges $i \to j$ are



(a) Node representations. **(b)** Message passing step.

Figure 2.1: Illustration of the message passing update in a graph neural network (GNN) on a fully-connected graph with four nodes. Each node is assigned a node representation $\mathbf{h}_i$ (left). In the message passing step (right), intermediate edge representations $\mathbf{h}_{(i,j)}$ are obtained from neighboring node representations $\mathbf{h}_i$, their initial node features $\mathbf{x}_i$ and edge features (if present) $\mathbf{x}_{(i,j)}$. Edge representations for incoming edges are aggregated to obtain updated node representations $\mathbf{h}'_i$.

represented by an ordered pair of nodes $(i, j) \in \mathcal{V} \times \mathcal{V}$. For undirected graphs, we assume that both (i, j) and (j, i) are in $\mathcal{E}$ if nodes i and j are connected.

A GNN takes as input an instance of a graph $\mathcal{G}$ (e.g., a sample from a dataset of many graphs), where nodes are associated with feature vectors $\mathbf{x}_i$ and edges can, too, be associated with feature vectors $\mathbf{x}_{(i,j)}$. We denote hidden representations in the neural network for nodes and edges with $\mathbf{h}_i$ and $\mathbf{h}_{(i,j)}$, respectively. We can set $\mathbf{h}_i = \mathbf{x}_i$ as an initial node representation. The structure of the graph $\mathcal{G}$ then determines the following message passing updates, which are executed in sequence to obtain updated node representations $\mathbf{h}'_i$ and edge representations $\mathbf{h}_{(i,j)}$:

$$\mathbf{h}_{(i,j)} = f_{\text{edge}}(\mathbf{h}_i, \mathbf{h}_j, \mathbf{x}_{(i,j)}) \,, \tag{2.3}$$

$$\mathbf{h}'_i = f_{\text{node}}(\mathbf{h}_i, \textstyle\sum_{j \in \mathcal{N}_i} \mathbf{h}_{(j,i)}, \mathbf{x}_i) \,. \tag{2.4}$$

$\mathcal{N}_i$ is the set of neighbors with an incoming edge to node i. f_{edge} and f_{node} typically are small MLPs with two or three layers which take a concatenation of the function arguments as input, but other choices are possible. Multiple message passing updates can be chained by setting $\mathbf{h}_i \leftarrow \mathbf{h}'_i$ after each node update given by Eq. 2.4. The parameters of f_{edge} and f_{node} need not be shared between message passing updates. See Figure 2.1 for an illustration of this message passing update.

This form of GNN was introduced by Gilmer et al. (2017) under the name *message passing neural network*, in an effort to generalize and unify earlier mod-

els, such as the *graph convolutional network* (GCN) (Kipf and Welling, 2017) or the *interaction network* (Battaglia et al., 2016). We can utilize this GNN as a function approximator on graph-based tasks trained with backpropagation, e.g., in the context of graph classification by aggregating the final outputs of the GNN into a global representation $\mathbf{h}_g = \sum_{i \in \mathcal{V}} \mathbf{h}_i$. For a recent study of the expressive power of this class of models in the context of function approximation, see Chen et al. (2019).

The first GNN model is typically attributed to Gori et al. (2005), who coined the term *graph neural network*. Their model contains many of the core ideas found in the GNN definition above, but was formulated as a recurrent neural network, trained by a version of backpropagation through time (Werbos, 1990) that demanded that message passing updates of the GNN model are a *contraction mapping*. This form of GNN further did not learn an explicit edge representation $\mathbf{h}_{(i,j)}$ and the update function for a node i was conditioned on neighboring states $\mathbf{h}_j$ with $j \in \mathcal{N}_i$ only (in addition to initial node feature vectors $\mathbf{x}_i$). Scarselli et al. (2009) extended this formulation by additionally conditioning the message passing update on initial edge features $\mathbf{x}_{(i,j)}$.

The GNN definition in Eqs. 2.3–2.4 is not all-encompassing, but covers the models considered in this thesis. Recent extensions include *graph networks* (Battaglia et al., 2018), which include a *global* state and update function, and *graph G-invariant networks* (Maron et al., 2019; Chen et al., 2019). Other recent related models and GNN variants can be cast as a special case of the message passing definition above, such as the *transformer* architecture (Vaswani et al., 2017) — see Battaglia et al. (2018) for details — and the *graph attention network* (Veličković et al., 2018b). Lastly, there exists a class of *spectral* methods for learning on graphs (Bruna et al., 2014; Henaff et al., 2015; Defferrard et al., 2016), which we will review in Chapter 3.

We will review additional prior and concurrent work on GNNs related to our model contributions in Part 1 of this thesis. For an overview of recent model variants and applications of GNNs, we recommend the review articles on *geometric deep learning* by Bronstein et al. (2017) and on *graph representation learning* by Hamilton et al. (2017b), and the position paper on *relational inductive biases and graph networks* by Battaglia et al. (2018).

2.4 LATENT VARIABLE MODELS

Several chapters in this thesis will utilize NNs in the context of *latent variable models*, for which we will provide a brief introduction here. We can understand a latent variable model as a probabilistic model that explains a set of observed variables $\mathbf{x}$ with a set of *latent variables* $\mathbf{z}$:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})\,d\mathbf{z}\,, \tag{2.5}$$

where $p_{\boldsymbol{\theta}}(\mathbf{x})$ is a model of the data distribution.

For our purposes, the goal of learning is to estimate the parameters of the conditional model $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ (typically called the *generative model*), such that $p_{\boldsymbol{\theta}}(\mathbf{x})$ is maximized for observed data $\mathbf{x} \in \mathcal{D}$ in a dataset $\mathcal{D}$, given a particular choice of prior $p(\mathbf{z})$.

We would often like to use NN-based generative models, e.g., by using a Gaussian output distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; f_{\boldsymbol{\theta}}(\mathbf{z}), \sigma^2 * \mathbf{I})$ with means provided by a NN $f_{\boldsymbol{\theta}}(.)$ and a fixed diagonal covariance with some scalar σ. This choice, however, typically renders the integral in Eq. 2.5 intractable. Variational inference, and in particular the *variational auto-encoder* (VAE) (Kingma and Welling, 2013; Rezende et al., 2014), addresses this issue by finding a lower bound to Eq. 2.5 (or equivalently, to the natural logarithm of this expression) and thus replacing the integration problem with an optimization problem.

We can obtain this *evidence lower bound* (ELBO) objective by introducing an approximate posterior distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and by using Jensen's inequality as follows:

$$\begin{aligned} \log p_{\boldsymbol{\theta}}(\mathbf{x}) &= \log \int p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})\,d\mathbf{z} \\ &= \log \int \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})\,d\mathbf{z} \geq \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\,d\mathbf{z}\,. \end{aligned} \tag{2.6}$$

The terms in the RHS of Eq. 2.6 can be re-arranged to arrive at a more commonly used expression for the ELBO:

$$\text{ELBO} = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] - D_{\text{KL}}[q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \,\|\, p(\mathbf{z})]\,, \tag{2.7}$$

where $\mathbb{E}_{q(.)}[\,.\,]$ denotes the expectation under a distribution $q(.)$ and $D_{\text{KL}}[\,.\,\|\,.\,]$ denotes the Kullback-Leibler divergence.

Similar to $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, the VAE model uses a NN-based *inference model* $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ with parameters $\boldsymbol{\phi}$. The prior $p(\mathbf{z})$ is often chosen to be a Gaussian distribution

with zero mean and unit covariance. Both the generative model and the inference model can jointly be optimized by stochastic gradient *ascent* with respect to the ELBO objective, using mini-batches of samples $\mathbf{x} \in \mathcal{D}$ from a dataset $\mathcal{D}$. Gradients of the term $\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})]$ with respect to the inference model parameters $\boldsymbol{\phi}$ can be obtained with the help of a Monte Carlo approximation of the expectation and by using the *reparameterization trick* (Kingma and Welling, 2013) for supported distributions $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$. Samples $\mathbf{z} \sim \mathcal{N}(\mathbf{z}\,; f_{\boldsymbol{\phi}}(\mathbf{x}), \sigma^2 * \mathbf{I})$ from a Gaussian distribution can be 'reparameterized' as follows:

$$\mathbf{z} = f_{\boldsymbol{\phi}}(\mathbf{x}) + \sigma\boldsymbol{\epsilon}\,, \text{ with } \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}\,; \mathbf{0}, \mathbf{I})\,. \tag{2.8}$$

After training, the inference model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ can be used to infer latent variables $\mathbf{z}$ of unseen test data $\mathbf{x}$. The trained generative model $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ can be used to generate data given a latent variable $\mathbf{z}$, for example obtained from the prior distribution $p(\mathbf{z})$.

We will use VAE-based latent variable models in conjunction with GNNs in Chapters 4 and 6, and in connection with sequential data in Chapter 7.

2.5 CONTRASTIVE LEARNING

Contrastive learning describes a class of methods for learning representations by contrasting pairs of related data examples against pairs of unrelated data examples. This approach naturally fits graph-structured data, as relations are given by the edges in the graph. We can cast this problem in the context of energy-based learning (LeCun et al., 2006), where we associate a scalar energy $E(f_{\boldsymbol{\theta}}(\mathbf{x}), f_{\boldsymbol{\theta}}(\mathbf{y}))$ for pairs of data points $(\mathbf{x}, \mathbf{y}) \in \mathcal{D} \times \mathcal{D}$ from a dataset $\mathcal{D}$, where $f_{\boldsymbol{\theta}}(.)$ is an encoder function that maps an observed data point $\mathbf{x}$ to its hidden representation $\mathbf{h}_{\mathbf{x}}$. We will use NNs for $f_{\boldsymbol{\theta}}(.)$ in practice. Training is carried out by optimizing an objective that encourages low energies for positive (related) pairs and higher energies for negative (unrelated) pairs.

Variants of this approach include *noise contrastive estimation* (NCE) (Gutmann and Hyvärinen, 2010; Mnih and Teh, 2012), *negative sampling* (Mikolov et al., 2013), and *deep metric learning* (Chopra et al., 2005; Hadsell et al., 2006). In

NCE and negative sampling, the objective is a binary cross-entropy loss, which specifically for negative sampling takes the following form:

$$\mathcal{L} = -\frac{1}{|\mathcal{T}|} \sum_{(\mathbf{x},\mathbf{y},c)\in\mathcal{T}} c \log l\big(s(\mathbf{x},\mathbf{y})\big) + (1-c) \log\Big(1 - l\big(s(\mathbf{x},\mathbf{y})\big)\Big) \,, \tag{2.9}$$

where we define the *score* for a pair as $s(\mathbf{x},\mathbf{y}) = -E(f_{\boldsymbol{\theta}}(\mathbf{x}), f_{\boldsymbol{\theta}}(\mathbf{y}))$. $\mathcal{T}$ is a set that contains all positive pairs and a number of negative pairs — usually k negative samples per positive pair, where k is a hyperparameter. $l(\mathbf{x}) = 1/(1+e^{-\mathbf{x}})$ is the logistic sigmoid function and c is an indicator variable that is 1 for positive pairs and 0 for negative samples. A common technique for obtaining negative samples is by *corrupting* a positive pair, e.g., by replacing one data example in the pair with a random other data example. In this context, the energy function is often chosen to be the negative inner product between hidden representations $E(\mathbf{h}_\mathbf{x}, \mathbf{h}_\mathbf{y}) = -\mathbf{h}_\mathbf{x}^\top \mathbf{h}_\mathbf{y}$, but other choices are possible.

The related NCE objective differs slightly from Eq. 2.9, and can be used to learn an (asymptotically) unbiased model of the underlying data distribution — see Dyer (2014) for details.

In *deep metric learning*, the goal is to learn representations $\mathbf{h}_\mathbf{x}$ of data examples $\mathbf{x} \in \mathcal{D}$ such that similar or positive pairs are assigned a small *distance* and dissimilar or negative pairs are assigned a larger distance in the representation space. The distance function $d(\mathbf{h}_\mathbf{x}, \mathbf{h}_\mathbf{y})$ can itself have parameters which are to be learned. A typical objective used in this setting is the following hinge loss:

$$\mathcal{L} = \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{x},\mathbf{y},c)\in\mathcal{T}} c\, d^2(\mathbf{h}_\mathbf{x}, \mathbf{h}_\mathbf{y}) + (1-c) \max\Big(0, \gamma - d^2(\mathbf{h}_\mathbf{x}, \mathbf{h}_\mathbf{y})\Big) \,, \tag{2.10}$$

with $\mathbf{h}_\mathbf{x} = f_{\boldsymbol{\theta}}(\mathbf{x})$ and $\mathbf{h}_\mathbf{y} = f_{\boldsymbol{\theta}}(\mathbf{y})$. γ is a hyperparameter. In this context, we can define the energy function to be the squared distance $E(f_{\boldsymbol{\theta}}(\mathbf{x}), f_{\boldsymbol{\theta}}(\mathbf{y})) = d^2(f_{\boldsymbol{\theta}}(\mathbf{x}), f_{\boldsymbol{\theta}}(\mathbf{y}))$. A common choice for $d(.,.)$ is the Euclidean distance.

We will make use of contrastive learning for unsupervised representation learning on graphs and link prediction in Chapter 4 using an objective based on Eq. 2.9. We will further use an objective similar to the hinge loss in Eq. 2.10 for state representation learning in Chapter 8.

Part I

Learning with Explicit Structure

MOTIVATION AND SUMMARY

A plethora of structured data comes in the form of graphs or networks: from social networks, the World Wide Web, and knowledge bases that serve as a foundation of most of our 'online' experience today, over street networks and power grids for infrastructure and city planning, to biological networks such as protein-interaction networks or even molecules and drugs, which can be represented as graphs themselves. Modeling this type of data using machine learning algorithms is an important and active area of research.

This part of the thesis explores how we can build neural network-based models for graph-structured data, i.e., data that is given to us in the explicit form of a graph or network.

In Chapter 3, we introduce the *graph convolutional network* (GCN), a simple yet effective architecture for representation learning on graphs. We demonstrate how GCNs can be used for the task of node classification in academic citation networks.

Chapter 4 extends the GCN model for unsupervised learning on graphs and link prediction, resulting in two model architectures that we call the *graph auto-encoder* (GAE) and the *variational GAE*.

The final chapter in this part of the thesis, Chapter 5, introduces the *relational GCN*, which extends the GCN and GAE models to work on multi-relational graph data. We apply relational GCNs to the task of entity classification in knowledge graphs.

3

GRAPH CONVOLUTIONAL NETWORKS FOR SEMI-SUPERVISED CLASSIFICATION

3.1 INTRODUCTION

Graphs are ubiquitous and used across many domains to represent structured and relational data, such as social networks, biological networks or knowledge bases. Accurate predictive models for graph-structured data thus have a wide range of applications which can be found across scientific disciplines and in industry.

Graphs come in many forms, sometimes allowing for multiple edge types and different types of nodes. Some graphs are directed, others are undirected, and there exist many other special cases depending on the particular application and use case. In this chapter, we focus on undirected graphs without edge attributes, which one could consider as the simplest type of graph representation used in practice. Social networks or academic co-authorship and citation networks can be represented in this way, among many other examples.

In this chapter and in Kipf and Welling (2017) we introduce the *graph convolutional network* (GCN), a specialized neural network architecture for graph-structured data[1]. We apply the GCN model for node classification in undirected graphs where nodes are allowed to have attributes, such as a featurized description of a document.

Our contributions are two-fold. Firstly, we simplify prior work on spectral graph convolutions (Hammond et al., 2011; Bruna et al., 2014; Defferrard et al.,

1 This chapter is based on our ICLR 2017 publication (Kipf and Welling, 2017). An earlier version of this paper appeared as arXiv preprint `arXiv:1609.02907`.

2016) by means of a first-order approximation. The resulting model, which we term GCN, can be understood as a neural network with integrated message passing operations, wherein messages are passed among direct neighbors in the graph. Secondly, we demonstrate how this form of a graph-based neural network model can be used for semi-supervised classification of nodes in a graph. Experiments on a number of datasets demonstrate that our model compares favorably both in classification accuracy and efficiency (measured in wall-clock time) against earlier state-of-the-art methods for semi-supervised learning.

3.2 BACKGROUND

3.2.1 Graph-Based Semi-Supervised Learning

We consider a setting in which labels are only available for a small subset of nodes. This problem can be framed as graph-based semi-supervised learning.

In graph-based semi-supervised learning, the graph structure is utilized in addition to both unlabeled and labeled data points, which take the role of nodes in the graph. A typical method to address this setting is to place a regularization loss $\mathcal{L}_{\text{reg}}$ on the model during training that takes into account the graph structure (Zhu et al., 2003; Zhou et al., 2004; Belkin et al., 2006; Weston et al., 2012), while the original supervised loss $\mathcal{L}_{\text{sup}}$ only considers individual labeled nodes in isolation:

$$\mathcal{L} = \mathcal{L}_{\text{sup}} + \lambda \mathcal{L}_{\text{reg}} \,. \tag{3.1}$$

A weighing factor λ (a hyperparameter) is used to weigh the contribution of the regularizer.

A common choice for the graph-based regularizer is based on a soft similarity constraint that encourages similar representations or predictions for neighboring nodes:

$$\mathcal{L}_{\text{reg}} = \sum_{i,j} A_{i,j} \| f(\mathbf{x}_i) - f(\mathbf{x}_j) \|^2 \propto f(\mathbf{X})^\top \mathbf{\Delta} f(\mathbf{X}) \,, \tag{3.2}$$

where $\mathbf{\Delta} = \mathbf{D} - \mathbf{A}$ denotes the unnormalized graph Laplacian of an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N nodes $i \in \mathcal{V}$, edges $(i, j) \in \mathcal{E}$, a binary adjacency

matrix $\mathbf{A} \in \{0,1\}^{N \times N}$ and a diagonal degree matrix $D_{i,i} = \sum_j A_{i,j}$. $f(\cdot)$ is a classifier (for our purposes a differentiable neural network) and $\mathbf{X} \in \mathbb{R}^{N \times d_{\text{in}}}$ is a matrix of d_{in}-dimensional node feature vectors $\mathbf{x}_i$. $f(\mathbf{X})$ in this context is to be understood as applying the classifier on all rows $\mathbf{x}_i$ of $\mathbf{X}$. The formulation of Eq. 3.2 relies on the assumption that connected nodes in the graph are likely to share the same label.

In this chapter, we will see that we can train an effective semi-supervised classifier *without* relying on an explicit regularization term. We will achieve this by informing the classifier itself with the structure of the graph, by conditioning it on the adjacency matrix: $f(\mathbf{X}, \mathbf{A})$. The resulting neural network model will perform message passing operations informed by the structure of the graph to distribute encoded feature information among connected nodes. At the same time, this will allow the model to distribute gradient information from the supervised loss $\mathcal{L}_{\text{sup}}$ and will enable it to learn representations of nodes both with and without labels.

3.2.2 Spectral Graph Convolutions

Our starting point for building a graph-based neural network classifier is the notion of a *spectral graph convolution*. A spectral convolution on a graph can be understood as a parameterized filtering operation that takes into account both node features (in this setting often described as *signal*) and the structure of a graph.

We consider spectral convolutions on graphs defined as the multiplication of a signal $\mathbf{x} \in \mathbb{R}^N$ (a scalar for every node) with a filter $g_{\boldsymbol{\theta}} = \text{diag}(\boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^N$ in the Fourier domain, i.e.:

$$g_{\boldsymbol{\theta}} \star x = \mathbf{U} g_{\boldsymbol{\theta}} \mathbf{U}^\top \mathbf{x}\,, \tag{3.3}$$

where $\mathbf{U}$ is the matrix of eigenvectors of the normalized graph Laplacian $\mathbf{L} = \mathbf{I}_N - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top$, with a diagonal matrix of its eigenvalues $\boldsymbol{\Lambda}$ and $\mathbf{U}^\top\mathbf{x}$ being the *graph Fourier transform* of $\mathbf{x}$. We can understand $g_{\boldsymbol{\theta}}$ as a function of the eigenvalues of $\mathbf{L}$, i.e., $g_{\boldsymbol{\theta}}(\boldsymbol{\Lambda})$. Evaluating Eq. 3.3 is computationally expensive, as multiplication with the eigenvector matrix $\mathbf{U}$ is $\mathcal{O}(N^2)$. Furthermore, computing the eigendecomposition of $\mathbf{L}$ in the first place might be prohibitively expensive for large graphs.

To circumvent this problem one can approximate $g_\theta(\mathbf{\Lambda})$ by a truncated polynomial expansion, e.g., using a monomial basis or, as proposed in Hammond et al. (2011), in terms of Chebyshev polynomials $T_k(x)$ up to K-th order:

$$g_{\theta'}(\mathbf{\Lambda}) \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\mathbf{\Lambda}}) \,, \tag{3.4}$$

with a rescaled $\tilde{\mathbf{\Lambda}} = \frac{2}{\lambda_{\max}}\mathbf{\Lambda} - \mathbf{I}_N$. $\lambda_{\max}$ denotes the largest eigenvalue of $\mathbf{L}$. $\boldsymbol{\theta}' \in \mathbb{R}^K$ is now a vector of Chebyshev coefficients. The Chebyshev polynomials are recursively defined as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$. The reader is referred to Hammond et al. (2011) and Defferrard et al. (2016) for an in-depth discussion of this approximation.

Going back to our definition of a convolution of a signal $\mathbf{x}$ with a filter $g_{\theta'}$, we now have:

$$g_{\theta'} \star \mathbf{x} \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\mathbf{L}})\mathbf{x} \,, \tag{3.5}$$

with $\tilde{\mathbf{L}} = \frac{2}{\lambda_{\max}}\mathbf{L} - \mathbf{I}_N$; as can easily be verified by noticing that $(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top)^k = \mathbf{U}\mathbf{\Lambda}^k\mathbf{U}^\top$. Note that this expression is now K-localized since it is a K-th order polynomial in the Laplacian, i.e., it depends only on nodes that are at maximum K steps away from the central node (K-th order neighborhood), and hence it can be seen as a *spatial* graph filter. The complexity of evaluating Eq. 3.5 is $\mathcal{O}(|\mathcal{E}|)$, i.e., linear in the number of edges. Defferrard et al. (2016) use this K-localized convolution to define a convolutional neural network on graphs.

3.3 METHODS

3.3.1 Graph Convolutional Networks

In this section, we introduce the *graph convolutional network* (GCN). The GCN is a graph-based neural network model $f(\mathbf{X}, \mathbf{A})$ with message passing operations that can be motivated as a first-order (i.e., linear) approximation to spectral graph convolutions, followed by a non-linear activation function.

Let $\mathbf{h}_i^{(l)} \in \mathbb{R}^{d_l}$ be the hidden representation vector of node $i \in \mathcal{V}$ with dimensionality d_l after the l-th message passing step (or 'layer'), then a single message passing step in the GCN model takes the following form:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)^\top}\mathbf{h}_i^{(l)} + \sum_{j\in\mathcal{N}_i} c_{i,j}\mathbf{W}_1^{(l)^\top}\mathbf{h}_j^{(l)}\right), \tag{3.6}$$

where σ is a pointwise non-linearity such as the ReLU activation function. $\mathbf{W}_0^{(l)}$ and $\mathbf{W}_1^{(l)}$ are learnable $d_l \times d_{l+1}$ parameter matrices and $\mathcal{N}_i$ is the set of neighbors of node i. $c_{i,j} = 1/\sqrt{D_{i,i}D_{j,j}}$ is a normalization constant, where $D_{i,i}$ is the degree of node i. We will later see that this constant originates from the symmetric normalization of the adjacency matrix used in the definition of the spectral graph convolution in Eq. 3.3. It is further possible to include a learnable, additive bias vector $\mathbf{b}$ in the update of Eq. 3.6, which we will omit for simplicity.

First-Order Model

We can write the GCN message passing update more compactly in matrix form:

$$\mathbf{H}^{(l+1)} = \sigma\Big(\mathbf{H}^{(l)}\mathbf{W}_0^{(l)} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}_1^{(l)}\Big), \tag{3.7}$$

where the normalized sum over neighboring nodes is replaced by a multiplication with the normalized adjacency matrix $\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$. $\mathbf{H}^{(l)} \in \mathbb{R}^{N\times d_l}$ is the matrix of activations in the l-th layer with $\mathbf{H}^{(0)} = \mathbf{X}$, i.e., the matrix of input node features.

The connection between the GCN message passing step and the definition of an approximate spectral graph convolution in Eq. 3.5 becomes evident if we set $K = 1$, i.e., if we take a first-order approximation to the spectral convolution, and further approximate $\lambda_{\max} \approx 2$:

$$g_{\theta'} \star \mathbf{x} \approx \theta_0'\mathbf{x} + \theta_1'\left(\mathbf{L} - \mathbf{I}_N\right)\mathbf{x} = \theta_0'\mathbf{x} - \theta_1'\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{x}. \tag{3.8}$$

Restricting the (linear) filtering operation to the first-order neighborhood in this way can reduce overfitting by using fewer free parameters per filtering operation, and hence might prove useful in contexts where only few labels are available or where a model is required to generalize in an inductive setting to unseen parts of a graph.

This definition can be generalized to a signal $\mathbf{X} \in \mathbb{R}^{N \times d_{\text{in}}}$ with d_{in} input channels (i.e., a d_{in}-dimensional feature vector for every node) and d_{out} filters or feature maps as follows:

$$\mathbf{Z} = \mathbf{X}\boldsymbol{\Theta}_0 - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{X}\boldsymbol{\Theta}_1 \,, \tag{3.9}$$

where $\boldsymbol{\Theta}_0$ and $\boldsymbol{\Theta}_1 \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ are now matrices of filter parameters and $\mathbf{Z} \in \mathbb{R}^{N \times d_{\text{out}}}$ is the convolved signal matrix[2]. We obtain the GCN message passing update in Eq. 3.7 by identifying $\mathbf{W}_0$ with $\boldsymbol{\Theta}_0$ and $\mathbf{W}_1$ with $-\boldsymbol{\Theta}_1$.

Single Parameter Model

In semi-supervised learning, overfitting to a small set of labeled nodes can often be an issue. This can be addressed by only using a single parameter matrix $\mathbf{W}^{(l)}$ per layer:

$$\mathbf{H}^{(l+1)} = \sigma\Big((\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{H}^{(l)}\mathbf{W}^{(l)}\Big) \,. \tag{3.10}$$

The operator $\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ has eigenvalues in the range $[0, 2]$, which can affect training stability when training deep neural network models with repeated application of this operator (e.g., by stacking multiple layers). While the model parameters could in principle adapt to this change in scaling, we find that it can have positive impact on training performance to renormalize the adjacency matrix with added self-connections as $\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} \rightarrow \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$, with $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ and $\tilde{D}_{i,i} = \sum_j \tilde{A}_{i,j}$. This results in the following single-parameter variant of the GCN model:

$$\mathbf{H}^{(l+1)} = \sigma\Big(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\Big) \,. \tag{3.11}$$

3.3.2 Semi-Supervised Node Classification

We consider a two-layer GCN for semi-supervised node classification on a graph with a symmetric binary adjacency matrix $\mathbf{A}$. We utilize the renormalized single-parameter model as outlined in Section 3.3.1. The renormalized

2 One could similarly arrive at this approximation by noting that the graph Laplacian $\mathbf{L}$ and the normalized adjacency matrix $\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ have the same eigenvectors (but different eigenvalues) and hence they can be exchanged in the definition of the graph Fourier transform. Using a polynomial filter up to first order recovers Eq. 3.8 up to a sign.

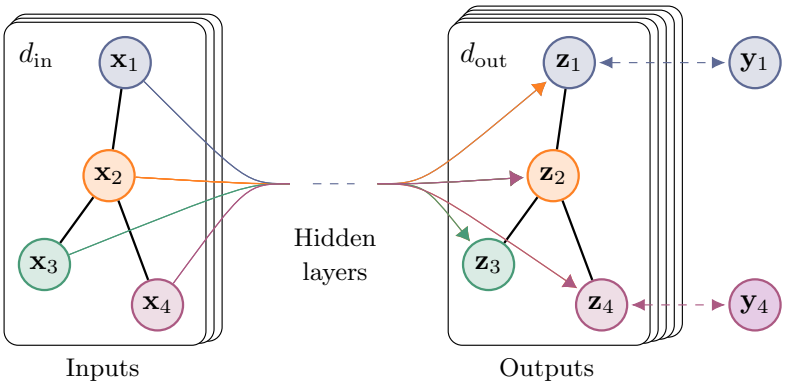


Figure 3.1: Schematic depiction of a multi-layer GCN for semi-supervised classification with d_{in} input channels and d_{out} feature maps in the output layer. The graph structure (edges are shown as black lines) is shared over layers. $\mathbf{x}_i$ are input features, $\mathbf{z}_i$ are node-wise predictions, and node labels are denoted by $\mathbf{y}_i$.

adjacency matrix $\widehat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ is calculated in a pre-processing step. Our forward model then takes the simple form:

$$\mathbf{Z} = f(\mathbf{X}, \mathbf{A}) = \text{softmax}\Big(\widehat{\mathbf{A}}\ \text{ReLU}\Big(\widehat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(0)}\Big)\ \mathbf{W}^{(1)}\Big)\ . \tag{3.12}$$

Here, $\mathbf{W}^{(0)} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{hid}}}$ is a input-to-hidden weight matrix for a hidden layer with H feature maps. $\mathbf{W}^{(1)} \in \mathbb{R}^{d_{\text{hid}} \times d_{\text{out}}}$ is a hidden-to-output weight matrix. The softmax activation function, defined as $\text{softmax}(x_i) = \frac{1}{\mathcal{Z}}\exp(x_i)$ with $\mathcal{Z} = \sum_i \exp(x_i)$, is applied row-wise.

We optimize the GCN for the task of semi-supervised node classification using the following cross-entropy loss on all *labeled* nodes:

$$\mathcal{L}_{\text{sup}} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{d_{\text{out}}} Y_{l,f} \ln Z_{l,f}\ , \tag{3.13}$$

where $\mathcal{Y}_L$ is the set of node indices that have labels and $Y_{l,f}$ is an indicator variable that is 1 if node l has label f and 0 otherwise.

The neural network weights $\mathbf{W}^{(0)}$ and $\mathbf{W}^{(1)}$ are trained using gradient descent. We perform batch gradient descent using the full dataset for every training iteration, which is a viable option as long as datasets fit in memory. An application of mini-batch gradient descent is non-trivial as individual data examples (nodes) depend on their neighborhoods, and is left for future work. We use a sparse representation for $\mathbf{A}$, hence space complexity is $\mathcal{O}(|\mathcal{E}|)$, i.e., linear in the number of edges. Stochasticity in the training process is introduced via dropout (Srivastava et al., 2014).

In the experiments of this chapter, we make use of TensorFlow (Abadi et al., 2016) for an efficient GPU-based implementation of Eq. 3.12 using sparse-dense matrix multiplications. Our implementation is available under `https://github.com/tkipf/gcn`.

3.4 RELATED PRIOR WORK

In this section, we discuss related work that was published prior to our work from this chapter. Our model draws inspiration both from the field of graph-based semi-supervised learning and from work on neural networks that operate on graphs. In what follows, we provide a brief overview on prior related work in both fields.

Graph-Based Semi-Supervised Learning

A large number of approaches for semi-supervised learning using graph representations have been proposed in the recent years, most of which fall into two broad categories: methods that use some form of explicit graph Laplacian regularization and graph embedding-based approaches.

Prominent examples for graph Laplacian regularization include label propagation (Zhu et al., 2003), manifold regularization (Belkin et al., 2006) and deep semi-supervised embedding (Weston et al., 2012). These approaches have been extended with ideas from spectral graph theory (Shuman et al., 2011; Ekambaram et al., 2013).

A separate branch of related models is based on graph embeddings with methods inspired by the skip-gram model (Mikolov et al., 2013). DeepWalk (Perozzi et al., 2014) and node2vec (Grover and Leskovec, 2016) learn embeddings via the prediction of the local neighborhood of nodes, sampled from random walks on the graph. LINE (Tang et al., 2015) similarly learns embeddings via neighborhood prediction, but without using random walks. For all these methods, however, a multi-step pipeline including embedding learning and semi-supervised training is required where each step has to be optimized separately. Planetoid (Yang et al., 2016) alleviates this shortcoming by injecting label information in the process of learning embeddings, but still relies on random walk generation.

Neural Networks on Graphs

Neural networks that operate on graphs had prior to our work been introduced in Gori et al. (2005) and Scarselli et al. (2009) as a form of recurrent neural network. Their framework requires the repeated application of contraction maps as propagation functions until node representations reach a stable fixed point. This restriction was later alleviated in Li et al. (2016) by introducing modern practices for recurrent neural network training to the original graph neural network framework.

Duvenaud et al. (2015) introduce a convolution-like propagation rule on graphs and methods for graph-level classification. Their approach requires to learn node degree-specific weight matrices which does not scale to large graphs with wide node degree distributions.

A related approach to semi-supervised node classification with a graph-based neural network is introduced in Atwood and Towsley (2016). Their model differs in that they integrate local graph information (up to a pre-chosen neighborhood size) in a single graph convolution-like layer, followed by fully-connected neural network layers.

A related framework for convolutional neural networks on graphs is introduced in Niepert et al. (2016). Their approach converts graphs locally into sequences that are fed into a conventional 1D convolutional neural network, which requires defining a canonical node ordering in a pre-processing step.

Our method is related to spectral graph convolutional neural networks, introduced in Bruna et al. (2014) and later extended by Defferrard et al. (2016) with fast localized convolutions. The latter of which can also be interpreted as a spatial method that performs message passing on local neighborhoods in the graph.

3.5 EXPERIMENTS

We test the proposed GCN model on semi-supervised document classification in three different citation networks. We further perform an evaluation of various graph propagation models and a run-time analysis on random graphs.

3.5.1 Datasets

We closely follow the experimental setup in Yang et al. (2016). Dataset statistics are summarized in Table 3.1. In the citation network datasets — Citeseer, Cora and Pubmed (Sen et al., 2008) — nodes are documents and edges are citation links. Label rate denotes the number of labeled nodes that are used for training divided by the total number of nodes in each dataset.

Table 3.1: Dataset statistics, as reported in Yang et al. (2016).

Dataset	Nodes	Edges	Classes	Features	Label rate
Citeseer	3,327	4,732	6	3,703	0.036
Cora	2,708	5,429	7	1,433	0.052
Pubmed	19,717	44,338	3	500	0.003

Citation Networks

We consider three citation network datasets: Citeseer, Cora, and Pubmed (Sen et al., 2008). The datasets contain sparse bag-of-words feature vectors for each document and a list of citation links between documents. We treat the citation links as (symmetric) edges and construct a binary, symmetric adjacency matrix $\mathbf{A}$. Each document has a class label. For training, we only use 20 labels per class, but all feature vectors.

Random Graphs

We simulate random graph datasets of various sizes for experiments where we measure training time per epoch. For a dataset with N nodes we create a random graph assigning $2N$ edges uniformly at random. We take the identity matrix $\mathbf{I}_N$ as input feature matrix $\mathbf{X}$, thereby implicitly taking a featureless approach where the model is only informed about the identity of each node, specified by a unique one-hot vector. In these experiments, we omit regularization (i.e., no dropout and no L2 regularization on the weights) and create dummy labels $Y_i = 1$ for each node. In each training epoch, we perform a forward pass on the full dataset, evaluate the cross-entropy error between the model prediction and the label *for every node* and update weights using Adam (Kingma and Ba, 2014). We measure and report the average wall-clock time in

seconds per epoch for 100 training epochs. We compare results on a GPU and on a CPU-only implementation in TensorFlow (Abadi et al., 2016)[3].

3.5.2 Experimental Setup

Unless otherwise noted, we train a two-layer GCN as described in Section 3.3.2 and evaluate prediction accuracy on a test set of 1000 labeled examples. We choose the same dataset splits as in Yang et al. (2016) with an additional validation set of 500 labeled examples for hyperparameter optimization (dropout rate for all layers, L2 regularization factor for the first GCN layer, and number of hidden units). We do not use the validation set labels for training.

We optimize hyperparameters on Cora only and use the same set of parameters for Citeseer and Pubmed. We train all models for a maximum of 200 epochs (training iterations) using Adam (Kingma and Ba, 2014) with a learning rate of 0.01 and early stopping with a window size of 10, i.e., we stop training if the validation loss does not decrease for 10 consecutive epochs. We initialize weights using the initialization described in Glorot and Bengio (2010) and accordingly (row-)normalize input feature vectors.

3.5.3 Baselines

We compare against the same baseline methods as in Yang et al. (2016): label propagation (LP) (Zhu et al., 2003), semi-supervised embedding (SemiEmb) (Weston et al., 2012), manifold regularization (ManiReg) (Belkin et al., 2006) and skip-gram based graph embeddings (DeepWalk) (Perozzi et al., 2014). We further compare against Planetoid (Yang et al., 2016), where we always choose their best-performing model variant (transductive vs. inductive) as a baseline.

3 Hardware used in experiments: 16-core Intel® Xeon® CPU E5-2640 v3 @ 2.60GHz, GeForce® GTX TITAN X

3.6 RESULTS

3.6.1 Semi-Supervised Node Classification

Results are summarized in Table 3.2.

Table 3.2: Summary of results in terms of classification accuracy in percent. See text for details.

Method	**Citeseer**	**Cora**	**Pubmed**
ManiReg (Belkin et al., 2006)	60.1	59.5	70.7
SemiEmb (Weston et al., 2012)	59.6	59.0	71.1
LP (Zhu et al., 2003)	45.3	68.0	63.0
DeepWalk (Perozzi et al., 2014)	43.2	67.2	65.3
Planetoid (Yang et al., 2016)	64.7 (26s)	75.7 (13s)	77.2 (25s)
GCN (Our method)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)
GCN (Random splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7

Reported numbers denote mean classification accuracy in percent. Results for baseline methods are taken from the Planetoid paper (Yang et al., 2016). Planetoid denotes the best model for the respective dataset out of the variants presented in their paper.

We further report wall-clock training time in seconds (s) until convergence for our method (incl. evaluation of validation error) and for Planetoid. For the latter, we used an implementation provided by the authors[4] and trained on the same hardware (with GPU) as our GCN model. We trained and tested our model on the same dataset splits as in (Yang et al., 2016) and report mean accuracy of 100 runs with random weight initializations. We used the following set of hyperparameters: 0.5 (dropout rate), $5 \cdot 10^{-4}$ (L2 regularization) and 16 (number of hidden units).

In addition, we report performance of our model on 10 randomly drawn dataset splits of the same size as in Yang et al. (2016), denoted by GCN (rand. splits). Here, we report mean and standard error of prediction accuracy on the test set split in percent.

4 https://github.com/kimiyoung/planetoid

3.6.2 Evaluation of Propagation Model

We compare different variants of our proposed per-layer propagation model on the citation network datasets. We follow the experimental set-up described in the previous section. Results are summarized in Table 3.3. The propagation model of the GCN model used in the experiments in Table 3.2 is denoted by *renormalized* (in bold). In all other cases, the propagation model of both neural network layers is replaced with the model specified under *propagation model*.

Table 3.3: Propagation model evaluation. See text for details.

Description	**Propagation model**	**Citeseer**	**Cora**	**Pubmed**
Chebyshev ($K = 3$)	$\sum_{k=0}^{K} T_k(\tilde{\mathbf{L}})\mathbf{X}\mathbf{W}_k$	69.8	79.5	74.4
Chebyshev ($K = 2$)		69.6	81.2	73.8
1^{st}-order	$\mathbf{X}\mathbf{W}_0 + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{X}\mathbf{W}_1$	68.3	80.0	77.5
Single parameter	$(\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{X}\mathbf{W}$	69.3	79.2	77.4
Renormalized	$\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{W}$	**70.3**	**81.5**	**79.0**
1^{st}-order term only	$\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{X}\mathbf{W}$	68.7	80.5	77.8
MLP	$\mathbf{X}\mathbf{W}$	46.5	55.1	71.4

Reported numbers denote mean classification accuracy for 100 repeated runs with random weight matrix initializations. In case of multiple variables $\mathbf{W}_i$ per layer, we impose L2 regularization on all weight matrices of the first layer. The models denoted as 1^{st}-*order term only* and *multi-layer perceptron* (MLP) are included for comparison; they represent the 1^{st}- and 0^{th}-order terms in the original 1^{st}-*order model*, respectively.

3.6.3 Training Time per Epoch

Here, we report results for the mean training time per epoch (forward pass, cross-entropy calculation, backward pass) on simulated random graphs, measured in seconds wall-clock time. The experimental set-up follows the description from Section 3.5.1. Figure 3.2 summarizes the results.

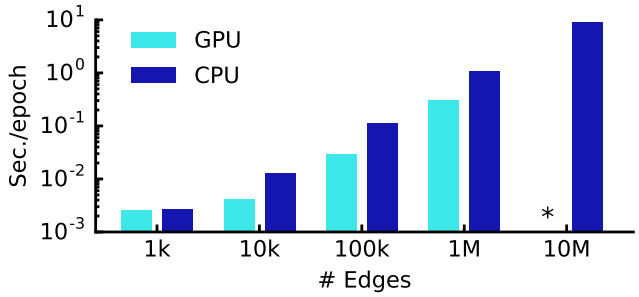


Figure 3.2: Experimental measurements of wall-clock time per epoch for simulated random graphs for a GPU and a CPU-only implementation. The GPU ran out of memory for the graph with 10 million edges.

3.7 DISCUSSION

3.7.1 Semi-Supervised Model

In the experiments demonstrated here, our proposed GCN model for semi-supervised node classification outperforms prior related methods by a significant margin. Methods based on graph-Laplacian regularization (Zhu et al., 2003; Belkin et al., 2006; Weston et al., 2012) are likely limited due to their assumption that edges solely encode similarity of class labels. Skip-gram based methods on the other hand are limited by the fact that they are based on a multi-step pipeline which is more difficult to optimize. Our proposed model can overcome both limitations, while still comparing favorably in terms of efficiency (measured in wall-clock time) to related methods.

We should emphasize that we used a single set of hyperparameters for all datasets. Other methods, such as Planetoid (Yang et al., 2016), typically do not generalize in such a way and require separate fine-tuning of hyperparameters.

We have further demonstrated that the proposed renormalized propagation model (Eq. 3.9) offers both improved efficiency (fewer parameters and operations, such as multiplication or addition) and better predictive performance compared to the naïve 1^{st}-order graph convolutional model (Eq. 3.8).

3.7.2 Limitations and Future Work

Here, we describe several limitations of our current model and outline how these might be overcome in future work.

Memory Requirement

In the current setup with full-batch gradient descent, memory requirement grows linearly in the size of the dataset. We have shown that for large graphs that do not fit in GPU memory, training on CPU can still be a viable option. Mini-batch stochastic gradient descent can alleviate this issue. The procedure of generating mini-batches, however, should take into account the number of layers in the GCN model, as the K-th order neighborhood for a GCN with K layers has to be stored in memory for an exact procedure. For very large and densely connected graph datasets, further approximations or sampling techniques might be necessary.

Fixed Weighing of Neighboring Nodes

The GCN propagation model resembles a center-surround filter, i.e., all neighboring node features are transformed using the same weight matrix followed by a weighted aggregation with fixed weights $c_{i,j}$. In many real-world scenarios, it might be beneficial to adaptively assign different weights to neighbors depending on their importance for a particular update. This could be achieved, for example, by making $c_{i,j}$ a learned function of neighboring node features, which is left for future work.

Directed Edges and Edge Features

The framework proposed in this chapter does not naturally support edge features and is limited to undirected graphs. This limitation will be addressed in Chapter 5 with the introduction of the *relational GCN* model.

3.8 CONCLUSION

We have introduced an approach for semi-supervised node classification using a neural network model for graph-structure data. Our model, termed GCN, uses an efficient layer-wise propagation rule that is based on a first-order approximation of spectral convolutions on graphs. Experiments on a number of network datasets suggest that the proposed GCN model is capable of encoding both graph structure and node features in a way useful for semi-supervised

classification. In this setting, our model outperforms several previously proposed methods by a significant margin, while being computationally efficient.

Since our original publication (Kipf and Welling, 2017), GCNs have been extended and applied in various settings and domains outside of semi-supervised document classification. Notable model extensions include attention-based propagation models such as in MoNet (Monti et al., 2017) or *graph attention networks* (Veličković et al., 2018b), and model variants that utilize node sampling for efficient and scalable mini-batch training (Hamilton et al., 2017a; Chen et al., 2018). Recent work by Wu et al. (2019) has found that our propagation-based approach to semi-supervised node classification can perform competitively on some tasks even in the absence of non-linear activation functions, i.e., using purely linear feature propagation.

GCNs and their model variants have since found application in, e.g., traffic prediction (Yu et al., 2018), object instance segmentation (Ling et al., 2019), large-scale recommender systems (Ying et al., 2018), machine translation (Bastings et al., 2017), and drug discovery (You et al., 2018), among many others. We will explore further extensions to the GCN model for unsupervised learning and link prediction, and for modeling relational data in the following chapters.

4

LINK PREDICTION WITH GRAPH AUTO-ENCODERS

4.1 INTRODUCTION

Having explored the problem of node classification in graph-structured datasets in the previous chapter, we now turn to a similarly important task in graph representation learning: the task of link prediction, i.e., the task of predicting whether two nodes should be connected by an edge. Link prediction has important applications in, e.g., knowledge base completion (Nickel et al., 2015), recommender systems (Chen et al., 2005), and in biological networks (such as protein-protein interaction networks).

With only a slight modification, we can turn the GCN model (Kipf and Welling, 2017) into an effective model for predicting missing edges in a graph[1]. This model, which we term *graph auto-encoder* (GAE), is based on an *encoder-decoder* architecture. The encoder module is a graph-based neural network which takes in a set of node features and outputs an updated set of node representations. We will use the GCN-based encoder introduced in Chapter 3 for this module. The decoder will *reconstruct* the connectivity structure of the graph, i.e., its adjacency matrix from node representations only. In practice, we will use a pairwise *scoring function*, such as an inner product, for the decoder module to determine whether two nodes should be connected or not.

We will further provide two alternative perspectives on the GAE framework: 1) a probabilistic version framed as a latent variable model, which we term the

1 This chapter is based on our NeurIPS 2016 workshop paper (Kipf and Welling, 2016). An earlier version of this paper appeared as arXiv preprint `arXiv:1611.07308`.

variational GAE (VGAE), and 2) a contrastive training procedure using negative sampling (Mikolov et al., 2013) for improved efficiency and scalability.

On a number of citation network datasets, we demonstrate that the proposed GAE framework achieves competitive results in the task of link prediction. In contrast to most earlier models for unsupervised learning on graph-structured data and link prediction (Tang and Liu, 2011; Perozzi et al., 2014; Tang et al., 2015; Grover and Leskovec, 2016), our model can naturally incorporate node features, which significantly improves predictive performance on a number of benchmark datasets.

4.2 METHODS

4.2.1 Graph Auto-Encoder

We are given an undirected, unweighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $N = |\mathcal{V}|$ nodes. We further introduce a binary $N \times N$ adjacency matrix $\mathbf{A}$ of $\mathcal{G}$ (with added self-connections, i.e., ones on the diagonal) and its diagonal degree matrix $\mathbf{D}$ with $D_{i,i} = \sum_j A_{i,j}$.

We address the task of link prediction by introducing a *scoring function* $s(\mathbf{z}_i, \mathbf{z}_j)$ that is tasked to assign high scores for pairs of nodes i and $j \in \mathcal{V}$ that are or should be connected and a low score otherwise. $\mathbf{z}_i$ and $\mathbf{z}_j \in \mathbb{R}^{d_{\text{emb}}}$ are *embedding vectors* or hidden representations for nodes i and j, respectively.

The GAE follows an encoder-decoder architecture. The scoring function acts as a decoder $s(\mathbf{z}_i, \mathbf{z}_j)$, which is tasked to reconstruct the adjacency matrix of the graph from hidden representations $\mathbf{z}_i$. The encoder, on the other hand, takes as input the adjacency matrix of the graph $\mathbf{A}$ and a set of node feature vectors $\{\mathbf{x}_i\}_{i \in \mathcal{V}}$, and produces hidden node representations $\mathbf{z}_i$.

Encoder

The GAE encoder model uses a GNN to process an initial set of node features $\mathbf{x}_i \in \mathbb{R}^{d_{\text{in}}}$ jointly with the structure of the graph to produce a set of hidden representations $\mathbf{z}_i$. Using a GNN-based encoder model is appealing as it can jointly capture feature descriptions of nodes and the structure of the graph, which allows the model to be used in inductive settings where predictions are to be made on unseen parts of the graph for which embeddings $\mathbf{z}_i$ are

not available. This is in contrast to embedding-based approaches, such as DeepWalk (Perozzi et al., 2014) or node2vec (Grover and Leskovec, 2016), that directly optimize embeddings $\mathbf{z}_i$ using a scoring function, without utilizing an encoder module, and hence cannot be applied inductively without re-training.

For simplicity, we use the GCN architecture introduced in Chapter 3 as the GNN component, but other choices are possible. Specifically, we use a two-layer GCN of the following form as encoder model:

$$\mathbf{Z} = \mathrm{GCN}(\mathbf{X}, \mathbf{A}) = \widehat{\mathbf{A}}\,\mathrm{ReLU}\big(\widehat{\mathbf{A}}\mathbf{X}\mathbf{W}_0\big)\mathbf{W}_1\,, \tag{4.1}$$

where we have summarized node features in a $N \times d_{\text{in}}$ matrix $\mathbf{X}$ and hidden representations in a $N \times d_{\text{emb}}$ matrix $\mathbf{Z}$. $\mathbf{W}_0 \in \mathbb{R}^{d_{\text{in}} \times d_{\text{hid}}}$ and $\mathbf{W}_1 \in \mathbb{R}^{d_{\text{hid}} \times d_{\text{emb}}}$ are trainable parameter matrices, initialized to small random values. $\widehat{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ is the symmetrically normalized adjacency matrix (with added self-connections). Alternatively, one could normalize the adjacency matrix via $\mathbf{D}^{-1}\mathbf{A}$, which we do not consider here to stay in line with the notation in Chapter 3.

Decoder

The task of the decoder is to reconstruct the adjacency matrix $\mathbf{A}$ (with added self-connections) from $\mathbf{Z}$. We obtain a *soft* reconstruction $\mathbf{A}'$ representing a weighted graph with edge weights in the interval $(0, 1)$ using the following decoder:

$$\mathbf{A}' = l\big(\mathbf{Z}\mathbf{Z}^\top\big)\,, \tag{4.2}$$

where $l(.)$ is the logistic sigmoid function. In other words, elements of the reconstructed adjacency matrix are obtained via $A'_{i,j} = l\big(s(\mathbf{z}_i, \mathbf{z}_j)\big)$ with an inner-product scoring function $s(\mathbf{z}_i, \mathbf{z}_j) = \mathbf{z}_i^\top \mathbf{z}_j$.

Training Objective

As $\mathbf{A}$ is binary, we can train the GAE model with a cross-entropy loss of the following form:

$$\mathcal{L} = -\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} A_{i,j}\log l\big(s(\mathbf{z}_i, \mathbf{z}_j)\big) + (1 - A_{i,j})\log\Big(1 - l\big(s(\mathbf{z}_i, \mathbf{z}_j)\big)\Big)\,. \tag{4.3}$$

For very sparse graphs it can be beneficial to use a weighted cross-entropy loss:

$$\mathcal{L} = -\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{\text{pos}} A_{i,j}\log A'_{i,j} + w_{\text{neg}}(1 - A_{i,j})\log(1 - A'_{i,j})\,, \tag{4.4}$$

with class weights w_{pos} and w_{neg} to account for class imbalance. We found the following heuristic choice to be effective for balancing the contribution of postive and negative terms in the loss function: $w_{\text{pos}} = N^2/(2N_{\text{pos}})$ and $w_{\text{neg}} = N^2/(2N_{\text{neg}})$, where $N_{\text{pos}} = |\mathcal{E}|$ is the number of edges in the graph and $N_{\text{neg}} = N^2 - N_{\text{pos}}$ is the number of zeros in $\mathbf{A}$.

4.2.2 Variational GAE

In this section, we introduce a probabilistic perspective on the GAE framework in the context of latent variable models, resulting in a model that we term the *variational GAE* (VGAE). See Chapter 2 for an introduction to latent variable models.

We start by modeling the *conditional* distribution $p_{\boldsymbol{\theta}}(\mathbf{A}|\mathbf{X})$, i.e., the likelihood of the graph $\mathcal{G}$ with adjacency matrix $\mathbf{A}$ conditioned on a matrix of node feature vectors $\mathbf{X}$, via the following latent variable model:

$$p_{\boldsymbol{\theta}}(\mathbf{A}|\mathbf{X}) = \int p_{\boldsymbol{\theta}}(\mathbf{A}|\mathbf{Z},\mathbf{X})p(\mathbf{Z}|\mathbf{X})\,d\mathbf{Z}\,, \tag{4.5}$$

with a fixed prior over the latent variables $p(\mathbf{Z}|\mathbf{X}) = \prod_{i=1}^{N} p(\mathbf{z}_i)$, independent of $\mathbf{X}$ and factorized over nodes $i \in \mathcal{V}$. In practice, we use a zero-mean, unit-covariance Gaussian prior $p(\mathbf{z}_i) = \mathcal{N}(\mathbf{z}_i; \mathbf{0}, \mathbf{I})$. The goal of learning is to find parameters $\boldsymbol{\theta}$ such that $p_{\boldsymbol{\theta}}(\mathbf{A}|\mathbf{X})$ is maximized for a single observed graph $\mathcal{G}$ with adjacency matrix $\mathbf{A}$ and node features $\mathbf{X}$, or for a dataset of multiple graphs.

Inference Model

We follow the VAE framework (Kingma and Welling, 2013; Rezende et al., 2014) and introduce an inference model $q_{\boldsymbol{\phi}}(\mathbf{Z}|\mathbf{X},\mathbf{A})$ for which we assume the following factorization (mean field approximation):

$$q_{\boldsymbol{\phi}}(\mathbf{Z}|\mathbf{X},\mathbf{A}) = \prod_{i=1}^{N} q_{\boldsymbol{\phi}}(\mathbf{z}_i|\mathbf{X},\mathbf{A})\,, \quad \text{with } q_{\boldsymbol{\phi}}(\mathbf{z}_i|\mathbf{X},\mathbf{A}) = \mathcal{N}(\mathbf{z}_i; \boldsymbol{\mu}_i, \operatorname{diag}(\sigma_i^2))\,. \tag{4.6}$$

The inference model is parameterized by two-layer GCNs as follows: $\boldsymbol{\mu}_i = [\text{GCN}^{(1)}(\mathbf{X},\mathbf{A})]_i$ and $\log \boldsymbol{\sigma}_i = [\text{GCN}^{(2)}(\mathbf{X},\mathbf{A})]_i$. The two GCN models are defined as in Eq. 4.1 with shared parameters $\mathbf{W}_0$ in the first layer and separate parameters $\mathbf{W}_1^{(1)}$ and $\mathbf{W}_1^{(2)}$ in the second (last) layer. Here, $[\mathbf{M}]_i$ denotes extraction the i-th row of a matrix $\mathbf{M}$ into a column vector.

Generative Model

We assume the following generative model, which factorizes over edges and is independent of the initial node feature matrix $\mathbf{X}$ for simplicity:

$$p_{\boldsymbol{\theta}}(\mathbf{A}|\mathbf{Z},\mathbf{X}) = \prod_{i=1}^{N}\prod_{j=1}^{N} p_{\boldsymbol{\theta}}(A_{i,j}|\mathbf{z}_i,\mathbf{z}_j)\,. \tag{4.7}$$

We model $p_{\boldsymbol{\theta}}(A_{i,j}|\mathbf{z}_i,\mathbf{z}_j)$ as a Bernoulli distribution with probabilities

$$p_{\boldsymbol{\theta}}(A_{i,j}=1|\mathbf{z}_i,\mathbf{z}_j) = l\big(s(\mathbf{z}_i,\mathbf{z}_j)\big)\,, \tag{4.8}$$

where $l(.)$ is the logistic sigmoid function and $s(.,.)$ is a scoring function for which we will use the inner product: $s(\mathbf{z}_i,\mathbf{z}_j) = \mathbf{z}_i^\top \mathbf{z}_j$. Note that one could also use a parameterized scoring function, such as a bilinear product $\mathbf{z}_i^\top \mathbf{W}\mathbf{z}_j$ with parameters $\mathbf{W}$.

Learning

We optimize the variational lower bound ELBO with respect to the parameters $\boldsymbol{\phi}$ of the inference model and, when using parameterized scoring functions, the parameters of the generative model $\boldsymbol{\theta}$:

$$\text{ELBO} = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{Z}|\mathbf{X},\mathbf{A})}\big[\log p_{\boldsymbol{\theta}}(\mathbf{A}|\mathbf{Z},\mathbf{X})\big] - \text{KL}\big[q_{\boldsymbol{\phi}}(\mathbf{Z}|\mathbf{X},\mathbf{A})\,||\,p(\mathbf{Z})\big]\,, \tag{4.9}$$

where $\text{KL}[q(.)||p(.)]$ is the Kullback-Leibler divergence between $q(.)$ and $p(.)$. We optimize via full-batch gradient ascent (i.e., using all data points at once) and make use of the *reparameterization trick* (Kingma and Welling, 2013) for training.

4.2.3 Contrastive Training

We can view the reconstruction objective of both the GAE and the Variational GAE models in the lens of contrastive learning, as introduced in Chapter 2. We can utilize *negative sampling* (Mikolov et al., 2013) to address the problem of class imbalance in the reconstruction loss, as an alternative to the re-weighing heuristic proposed in Eq. 4.4. Negative sampling will further avoid the expensive evaluation of $\mathcal{O}(N^2)$ negative terms ($A_{i,j} = 0$) in the loss for every

training iteration, and hence can speed up training. Using negative sampling, the reconstruction objective[2] takes the following form:

$$\mathcal{L} = -\frac{1}{|\mathcal{T}|} \sum_{(i,j,c)\in\mathcal{T}} c \log l\big(s(\mathbf{z}_i, \mathbf{z}_j)\big) + (1-c) \log\Big(1 - l\big(s(\mathbf{z}_i, \mathbf{z}_j)\big)\Big)\,, \qquad (4.10)$$

where $\mathcal{T}$ is a set that contains tuples (i, j, c): i and j indicate edge indices $(i, j) \in \mathcal{V} \times \mathcal{V}$ and $c \in \{0, 1\}$ indicates whether (i, j) is a *positive example* ($c = 1$), i.e., whenever the edge (i, j) exists in the graph ($A_{i,j} = 1$), or a *negative example* ($c = 0$) otherwise. Negative sampling of this form is used for learning graph embeddings, e.g., in node2vec (Grover and Leskovec, 2016) for undirected graphs and DistMult (Yang et al., 2014) for directed, relational graphs. A common choice is to use the same number of positive and negative examples and to sample negatives uniformly at random from the set of negative terms with $A_{i,j} = 0$.

Under this perspective, we can view learning under an energy-based framework (LeCun et al., 2006) as outlined in Chapter 2. This suggests extensions of the GAE model with different decoder variants, e.g., with the energy of a pair of nodes defined as the Euclidean distance between their embeddings, and other training objectives such as the hinge loss in Eq. 2.10.

4.3 RELATED PRIOR WORK

Graph Embeddings

Our approach is closely related to embedding-based approaches for link prediction and node classification (Tang and Liu, 2011; Perozzi et al., 2014; Tang et al., 2015; Grover and Leskovec, 2016; Cao et al., 2016; Wang et al., 2016). For earlier approaches based on feature engineering, see Liben-Nowell and Kleinberg (2007) for an overview. In Tang and Liu (2011), node embeddings are obtained via spectral decomposition of the normalized graph Laplacian, using the first d_{emb} eigenvectors with the smallest eigenvalues. This is related to *Laplacian eigenmaps* (Belkin and Niyogi, 2003), with the difference that the adjacency matrix is provided externally and not obtained from the node features. In DeepWalk (Perozzi et al., 2014), node embeddings are obtained in

2 This corresponds to the first term in the ELBO (Eq. 4.9) for VGAE, up to a constant.

a two-stage procedure: first, the graph is serialized into a number of short random walks, and secondly, the SkipGram model (Mikolov et al., 2013) is applied to the random walk-based node sequences. An extension of DeepWalk that includes node features was proposed by Yang et al. (2015). Grover and Leskovec (2016) propose a variant of DeepWalk with biased random walks, termed node2vec. LINE (Tang et al., 2015) directly optimizes a score function similar to ours with negative sampling for first-order neighbors (and optionally also for second-order neighbors), but does not use an encoder. Lastly, Cao et al. (2016) and Wang et al. (2016) use auto-encoders applied on node features to obtain node embeddings. Cao et al. (2016) use the adjacency vector of a node (a row in the adjacency matrix) as initial node features, whereas Wang et al. (2016) use rows of a node-node co-occurrence matrix in short random walk sequences as node features.

Generative Models of Graphs

The VGAE is a probabilistic model of graph generation, a class of models which dates back to the random graph model by Erdös and Rényi (1959), in which edges are modeled by a fixed probability. The *stochastic block model* (Holland et al., 1983) generalizes this notion to a graph with two (or multiple) communities, where edges within communities and between communities are modeled with different probabilities. Other well-known fixed-probability generative models for graphs include the preferential attachment model by Barabási and Albert (1999) and the small-world model by Watts and Strogatz (1998). In our approach, edge probabilities are instead provided by a neural network-based model. Concurrent to our work, two other generative models for graphs based on deep neural networks have been proposed: Johnson (2017) proposes a generative model for graphs that utilizes GNNs to obtain intermediate node representations and predicts one or multiple nodes and edges to be added to the graph per generation step, in a recurrent generative process. Gómez-Bombarelli et al. (2016) introduce a VAE-based generative model for molecular graphs, which operate on a sequence-based representation for molecules using recurrent neural networks.

4.4 EXPERIMENTS

We demonstrate the ability of the VGAE and GAE models to learn representations useful for a link prediction task on undirected graphs.

4.4.1 Datasets

As in the previous chapter, we consider three citation network datasets: Citeseer, Cora, and Pubmed (Sen et al., 2008). Nodes are documents described by sparse bag-of-words feature vectors, and edges are (undirected) citation links between documents. Dataset statistics are summarized in Table 4.1.

Table 4.1: Datasets used for link prediction.

Dataset	Nodes	Edges	Features
Citeseer	3,327	4,732	3,703
Cora	2,708	5,429	1,433
Pubmed	19,717	44,338	500

4.4.2 Experimental Setup

The models are trained on an incomplete version of these datasets where parts of the citation links (edges) have been removed, while all nodes (and their feature descriptions) are kept. We form validation and test sets from previously removed edges and the same number of randomly sampled pairs of unconnected nodes (non-edges).

We compare models based on their ability to correctly classify edges and non-edges. The validation and test sets contain 5% and 10% of citation links, respectively. The validation set is used for optimization of hyperparameters. We compare against two popular baselines: *spectral embedding* (SE) (Tang and Liu, 2011) and *DeepWalk* (DW) (Perozzi et al., 2014). Both SE and DW provide node embeddings $\mathbf{Z}$. We use Eq. 4.2, i.e., an inner product followed by a sigmoid activation function, to calculate scores for elements of the reconstructed adjacency matrix. Both SE and DW do not support node features.

For VGAE and GAE, we initialize weights as described in Glorot and Bengio (2010). We train for 200 iterations using Adam (Kingma and Ba, 2014) with a learning rate of 0.01. We use a hidden layer of dimensionality $d_{\text{hid}} = 32$ and latent variables of dimensionality $d_{\text{emb}} = 16$ in all experiments. We utilize the weighted cross entropy from Eq. 4.4 for the reconstruction loss in VGAE and GAE. In initial experiments we found that the contrastive loss from Eq. 4.10 resulted in comparable predictive performance, but required more training epochs for convergence due to negative sampling. For SE, we use the SpectralEmbedding implementation from Pedregosa et al. (2011) with an embedding dimension of 128. For DW, we use the implementation provided by Grover and Leskovec (2016) with standard settings used in their paper, i.e., embedding dimension of 128, 10 random walks of length 80 per node and a context size of 10, trained for a single epoch.

For a model variant of VGAE/GAE without feature conditioning, we simply drop the dependence on $\mathbf{X}$ in the inference model (in case of VGAE) and replace $\mathbf{X}$ with the identity matrix in the GCN. Our implementation in TensorFlow (Abadi et al., 2016) is available under `https://github.com/tkipf/gae`.

4.4.3 Results

Results for the link prediction task in citation networks are summarized in Table 4.2. GAE* and VGAE* denote experiments with using node features, all other models do not use node features. We report *area under the ROC curve* (AUC) and *average precision* (AP) scores for each model on the test set. Numbers show mean results and standard error for 10 runs with random initializations on fixed dataset splits.

Both VGAE and GAE achieve competitive results on the featureless task. Adding input features significantly improves predictive performance across datasets. We did not perform an explicit comparison to baselines with support for node features in our original experimental study in Kipf and Welling (2016), on which this chapter is based. Such a comparison, however, was carried out in later work by Bojchevski and Günnemann (2017), where GAE was found to significantly outperform earlier representatives of this model class — TADW (Yang et al., 2015) and TRIDNR (Pan et al., 2016) — on similar datasets as considered here.

Table 4.2: Average precision (AP) and area under the ROC curve (AUC) scores for link prediction. * denotes models that make use of node features. Highest mean scores highlighted in bold.

Method	Cora		Citeseer		Pubmed	
	AUC	AP	AUC	AP	AUC	AP
SE	86.8 ± 0.02	90.3 ± 0.01	78.5 ± 0.04	82.8 ± 0.02	84.7 ± 0.02	88.2 ± 0.01
DW	84.7 ± 0.01	86.7 ± 0.01	80.6 ± 0.02	82.9 ± 0.02	84.3 ± 0.00	84.3 ± 0.00
GAE	86.1 ± 0.02	89.2 ± 0.01	78.4 ± 0.02	83.6 ± 0.02	81.8 ± 0.01	87.3 ± 0.00
VGAE	86.3 ± 0.01	89.2 ± 0.01	78.1 ± 0.01	83.3 ± 0.01	82.6 ± 0.01	87.5 ± 0.01
GAE*	92.3 ± 0.02	92.6 ± 0.02	89.1 ± 0.04	89.3 ± 0.05	$\mathbf{95.9} \pm 0.00$	$\mathbf{96.1} \pm 0.00$
VGAE*	$\mathbf{92.8} \pm 0.01$	$\mathbf{93.2} \pm 0.01$	$\mathbf{90.3} \pm 0.02$	$\mathbf{91.8} \pm 0.02$	94.4 ± 0.01	94.7 ± 0.01

4.5 LIMITATIONS

Feature Conditioning and Decoding

Our GAE model is framed as a conditional auto-encoder (or VAE), where both the encoder and the decoder are conditioned on the node feature matrix $\mathbf{X}$, i.e., we are modeling $p_{\boldsymbol{\theta}}(\mathbf{A}|\mathbf{Z}, \mathbf{X})$. Note that we drop the conditioning on $\mathbf{X}$ in the decoder in practice to make for a simpler model: this typically does not affect link prediction performance since the hidden representation $\mathbf{Z}$ provided by the encoder can capture information stored in $\mathbf{X}$. If the GAE model is to be used in a purely generative way without an encoder, then it might be beneficial to introduce an explicit dependency on $\mathbf{X}$ in the decoder. Lastly, one could frame the GAE model as a non-conditional generative model $p_{\boldsymbol{\theta}}(\mathbf{A}, \mathbf{X}|\mathbf{Z})$ where both $\mathbf{A}$ and $\mathbf{X}$ are reconstructed. This would only require a minor change in the GAE decoder, for example by adding a separate output head (a small MLP) that produces $\mathbf{X}$ from $\mathbf{Z}$.

Graph Generation

The GAE model does not learn a global latent representation for an instance of a graph, but rather assigns one latent variable per node in a graph. It further does not model the *number of nodes* in a graph, but assumes that this is provided to the model externally via the size of the provided adjacency matrix. The GAE model is hence less suited for modeling a distribution of many small graphs of different size, e.g., in the context of drug discovery. Better-suited models

for such a task typically assign a global latent variable per graph and allow for generation of graphs of different size, such as GraphRNN (You et al., 2018), GraphVAE (Simonovsky and Komodakis, 2018), and MolGAN (De Cao and Kipf, 2018).

Scalability of Encoder

The scalability of the GAE model is mainly limited by its encoder: using a GNN requires processing of not only the nodes in a mini-batch itself, but also their respective neighbors and potentially higher-order neighbors (when using multiple message passing steps). Hence, even when using negative sampling and a sparse implementation of the message passing operations to reduce the overall complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(|\mathcal{E}|)$, training the GAE model on very large graphs can still be challenging. To overcome this limitation, neighborhood sampling techniques such as in Hamilton et al. (2017a) and Ying et al. (2018) can be employed in the GAE encoder.

4.6 CONCLUSION

Motivated by our research question, whether GNNs can be utilized for link prediction and unsupervised learning, we have proposed the *graph auto-encoder* (GAE) and *variational GAE* (VGAE) models. These models serve as a first demonstration that GNNs are promising candidates for link prediction problems. GAEs and VGAEs learn node representations in an unsupervised way, which could prove useful for other downstream tasks. Using representations learned by GAE for node classification is studied in Bojchevski and Günnemann (2017), who demonstrate strong performance gains over earlier related methods. An extension of GAE for semi-supervised classification and with an iterative decoding scheme was later proposed in Graphite (Grover et al., 2018). Follow-up work by Davidson et al. (2018) explores utilizing a hyperspherical latent space in the VGAE model which improves link prediction performance on some datasets.

One major challenge in utilizing GNNs for link prediction lies in scalability. Negative sampling can alleviate this issue in terms of evaluation of the decoder module in GAE/VGAE. The GNN-based encoder, however, requires keeping large parts of the graph in memory for message passing. One approach to

overcome this limitation is based on subsampling neighborhoods, which is utilized in PinSAGE (Ying et al., 2018) and allows the authors to train a GAE-based model on a recommendation task with 18 billion edges.

GAEs, as proposed in this chapter, are limited to undirected graphs without edge features. We will address this limitation in the following chapter with the introduction of the *relational GCN* (R-GCN) model.

5

MODELING RELATIONAL DATA WITH GRAPH CONVOLUTIONAL NETWORKS

5.1 INTRODUCTION

In Chapters 3 and 4 we have introduced the *graph convolutional network* (GCN) and explored applications in node classification and link prediction on undirected graphs with node attributes. In this chapter and in Schlichtkrull and Kipf et al. (2018), we introduce an extension to the GCN model that extends its modeling capabilities to multi-relational data. We call this model the *relational GCN* (R-GCN)[1]. Multi-relational data can be represented as a graph with multiple edge types (i.e., *labeled edges*), and potentially multiple edges between two nodes (also called a *multi-graph*).

An important example of multi-relational data is a *knowledge base*. Knowledge bases organize and store factual knowledge, enabling a range of applications including question answering (Yao and Van Durme, 2014; Bao et al., 2014; Seyler et al., 2015; Hixon et al., 2015; Bordes et al., 2015; Dong et al., 2015) and information retrieval (Kotov and Zhai, 2012; Dalton et al., 2014; Xiong and Callan, 2015b; Xiong and Callan, 2015a).

Developing machine learning models for knowledge bases, and for multi-relational data in general, is important, as even the largest knowledge bases (e.g., DBPedia, Wikidata or Yago) are incomplete and potentially noisy, despite enormous efforts spent in populating and maintaining them. The resulting

1 This chapter is based on our ESWC 2018 publication (Schlichtkrull and Kipf et al., 2018). The first two authors contributed equally to this publication. An earlier version of this paper appeared as arXiv preprint `arXiv:1703.06103`. Permission was given by the co-authors for reproduction in this thesis.

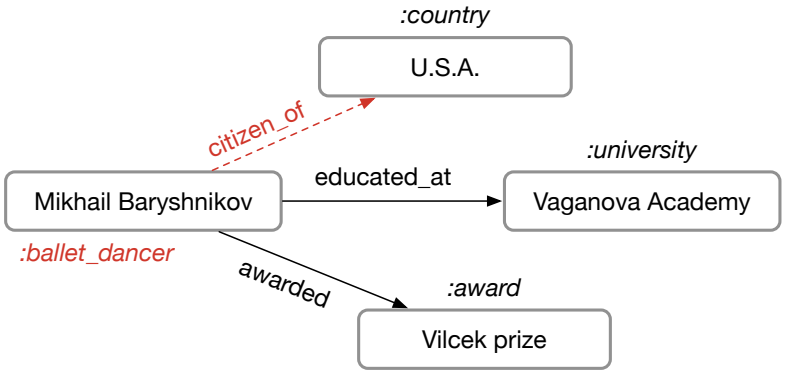


Figure 5.1: A knowledge base fragment: The nodes are entities, the edges are relations labeled with their types, the nodes are labeled with entity types (e.g., *university*). The edge and the node label shown in red are the missing information to be inferred.

gaps in coverage can significantly harm downstream applications. Predicting missing information in knowledge bases via machine learning techniques is a typical task in *statistical relational learning* (SRL), a subfield of machine learning that addresses learning with relational structure.

As commonly done in SRL, we represent a knowledge base as a collection of triples of the form (subject, predicate, object). Consider, for example, the triple (*Mikhail Baryshnikov*, *educated_at*, *Vaganova Academy*), where we will refer to *Baryshnikov* and *Vaganova Academy* as entities and to *educated_at* as a relation. Additionally, we assume that entities are labeled with types (e.g., *Vaganova Academy* is marked as a *university*). This collection of triples can be understood as describing a directed labeled multi-graph with entities corresponding to nodes (subjects and objects) and triples encoded by labeled edges (see Figure 5.1).

In this chapter, we focus on the SRL task of entity classification (assigning types or categorical properties to entities). Many missing pieces of information can be expected to reside within the graph encoded through the neighborhood structure and hence we utilize a GNN-based encoder model for entities in the relational graph. In the example in Figure 5.1, we can see that knowing that *Mikhail Baryshnikov* was educated at the *Vaganova Academy* implies both that *Mikhail Baryshnikov* should have the label *person*, and that the triple (*Mikhail Baryshnikov*, *lived_in*, *Russia*) must belong to the knowledge graph. The latter is an example of *relational link prediction*, which is another important SRL task, and we refer the reader to Schlichtkrull and Kipf et al. (2018) for specific details on how our proposed R-GCN model can be adapted for this task.

Our entity classification model uses softmax classifiers at each node in the graph, similarly to Kipf and Welling (2017) as described in Chapter 3. The classifiers take node representations supplied by an R-GCN graph encoder and predict the labels. The model, including R-GCN parameters, is trained by optimizing the cross-entropy loss.

Our main contributions in this chapter are as follows: we extend the GCN framework to support multi-relational data, specifically for entity classification tasks, and we introduce regularization techniques utilizing parameter sharing, that allow us to effectively apply R-GCNs to graphs with a large number of relations. We validate the proposed model on semi-supervised entity classification tasks in multi-relational data with sizes up to approx. 6M triples.

5.2 METHODS

We introduce the following notation: we denote directed and labeled multi-graphs as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ with nodes (entities) $i \in \mathcal{V}$ and labeled edges (relations) $(i, r, j) \in \mathcal{E}$, where $r \in \mathcal{R}$ is a relation type[2] and i and $j \in \mathcal{V}$ are nodes.

5.2.1 Relational GCN

Our model is primarily motivated as an extension of the GCN model, introduced in Chapter 3 and in Kipf and Welling (2017), to large-scale relational data. Recall that we defined the message-passing step in the GCN model as follows:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i} \frac{1}{c_{i,j}} \mathbf{W}^{(l)} \mathbf{h}_j^{(l)} + \mathbf{W}_0^{(l)} \mathbf{h}_i^{(l)}\right), \tag{5.1}$$

where $\mathbf{h}_i^{(l)} \in \mathbb{R}^{d^{(l)}}$ is the hidden state of node $i \in \mathcal{V}$ in the l-th layer of the neural network, with $d^{(l)}$ being the dimensionality of this layer's representations. Incoming messages of the form $\mathbf{W}^{(l)} \mathbf{h}_j^{(l)}$ are accumulated, normalized using a normalization constant $c_{i,j}$, and passed through an element-wise activation function $\sigma(\cdot)$, such as the $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$.

2 We assume that $\mathcal{R}$ contains relations both in canonical direction (e.g., *born_in*) and in inverse direction (e.g., *born_in_inv*).

We can extend this update rule to support multiple edge types by using a set of relation-specific weight matrices $\mathbf{W}_r^{(l)}$, where $r \in \mathcal{R}$ denotes the relation type, as opposed to a single weight matrix $\mathbf{W}^{(l)}$ that is used across all edges irrespective of their type:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}\mathbf{W}_r^{(l)}\mathbf{h}_j^{(l)} + \mathbf{W}_0^{(l)}\mathbf{h}_i^{(l)}\right), \tag{5.2}$$

where $\mathcal{N}_i^r$ denotes the set of neighbor node indices that are connected to node i with an incoming edge of relation type $r \in \mathcal{R}$. $c_{i,r}$ is a normalization constant.

We can recover two interesting special cases for this update rule. Firstly, the R-GCN update rule is identical to that of a CNN with grid-structured filters (e.g., 3×3 filters for image processing), where nodes are locations in a regular 2D grid and relation types are spatial relations such as *upper-right neighbor*, and when setting the normalization constant $c_{i,r}$ to 1 and $|\mathcal{N}_i^r| = 1$, i.e., only one neighbor per relation type. Secondly, by setting $c_{i,r} = 1$, dropping the self-connection $\mathbf{W}_0^{(l)}\mathbf{h}_i^{(l)}$ and by coupling weight matrices over layers, we recover a variant of the Gated Graph Neural Network (Li et al., 2016) model without gated updates, which similarly uses a relation-specific update function.

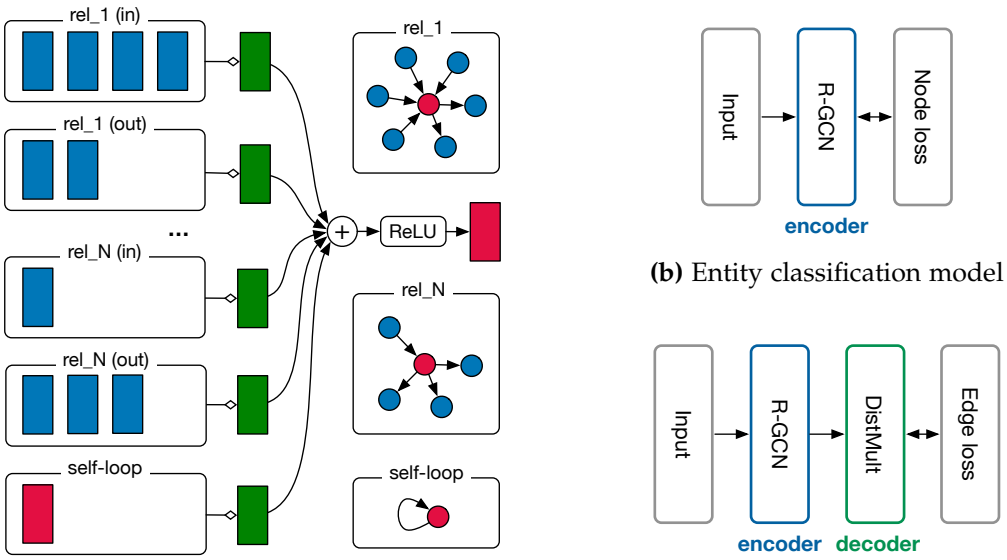
A neural network layer update consists of evaluating the message passing update (Eq. 5.2) in parallel for every node $i \in \mathcal{V}$ in the graph. Multiple layers can be stacked to allow for dependencies across several relational steps. We refer to this graph encoder model as a *relational GCN* (R-GCN). The computation graph for a single node update in the R-GCN model is depicted in Figure 5.2.

5.2.2 Regularization

A central issue with applying the message passing update in Eq. 5.2 to highly multi-relational data is the rapid growth in number of parameters with the number of relations in the graph. In practice this can easily lead to overfitting on rare relations and to models of very large size. An intuitive strategy to address such issues is to share parameters between weight matrices to limit the total number of parameters.

Corresponding to this strategy, we introduce a *basis decomposition*, where we decompose $\mathbf{W}_r^{(l)}$ as follows:

$$\mathbf{W}_r^{(l)} = \sum_{b=1}^{B} a_{r,b}^{(l)}\mathbf{V}_b^{(l)}, \tag{5.3}$$



(a) Single R-GCN layer

(b) Entity classification model

(c) Link prediction model

Figure 5.2: Diagram for computing the update of a single graph node/entity (red) in the R-GCN model. Activations from neighboring nodes (dark blue) are gathered and then transformed for each relation type individually (for both in- and outgoing edges). The resulting representation (green) is accumulated in a (normalized) sum and passed through an activation function (such as the ReLU). This per-node update can be computed in parallel with shared parameters across the whole graph. (b) Depiction of an R-GCN model for entity classification with a per-node loss function. (c) The R-GCN model can also be used in an encoder-decoder framework for link prediction tasks, similar to the GAE model (Kipf and Welling, 2016) presented in Chapter 4, here shown using a DistMult (Yang et al., 2014) decoder. We refer to Schlichtkrull and Kipf et al. (2018) for details on this link prediction setting.

i.e., as a linear combination of basis transformations $\mathbf{V}_b^{(l)} \in \mathbb{R}^{d^{(l+1)} \times d^{(l)}}$ with learnable coefficients $a_{r,b}^{(l)}$ such that only the coefficients depend on r.

The basis decomposition (Eq. 5.3) can be seen as a form of effective weight sharing between different relation types, which reduces the number of parameters needed to be learned for highly multi-relational data (such as realistic knowledge bases).

The overall R-GCN model then takes the following form: We stack L layers as defined in Eq. 5.2 — the output of the previous layer being the input to the next layer. The input to the first layer can be chosen as a unique one-hot vector for each node in the graph if no other features are present. While in this work

we only consider the featureless approach, we note that GCN-type models can incorporate predefined feature vectors (Kipf and Welling, 2017).

5.2.3 Entity Classification

For (semi-)supervised classification of nodes (entities), we simply stack R-GCN layers as defined in Eq. 5.2, with a softmax$(\cdot)$ activation (per node) on the output of the last layer. We minimize the following cross-entropy loss on all labeled nodes (while ignoring unlabeled nodes):

$$\mathcal{L} = -\sum_{i \in \mathcal{Y}} \sum_{k=1}^{K} t_{i,k} \ln h_{i,k}^{(L)} , \tag{5.4}$$

where $\mathcal{Y}$ is the set of node indices that have labels and $h_{i,k}^{(L)}$ is the k-th entry of the network output for the i-th labeled node. $t_{i,k}$ denotes its respective ground truth label. In practice, we train the model using (full-batch) gradient descent techniques, similar to the way we trained the GCN model in Chapter 3. A schematic depiction of the model is given in Figure 5.2b.

5.3 RELATED PRIOR WORK

A wide range of techniques in SRL learn representations of entities and relations via direct optimization of a *scoring function* (Nickel et al., 2011; Bordes et al., 2013; Socher et al., 2013; Yang et al., 2014; Chang et al., 2014; Nickel et al., 2016; Trouillon et al., 2016; Dettmers et al., 2018). Many of these approaches can be regarded as modifications or special cases of classic tensor decomposition methods such as CP or Tucker; for an overview of tensor decomposition literature we refer to Kolda and Bader (2009). For an overview of the field of SRL for knowledge base completion, we recommend the review article by Nickel et al. (2015).

Incorporation of paths between entities in knowledge bases has recently received considerable attention. We can roughly classify previous work into 1) methods creating auxiliary triples, which are then added to the learning objective of a factorization model (Guu et al., 2015; García-Durán et al., 2015); 2) approaches using paths (or walks) as features when predicting edges (Lin et al., 2015); or 3) doing both at the same time (Neelakantan et al., 2015; Toutanova et

al., 2016). The first direction is largely orthogonal to ours, as we would also expect improvements from adding auxiliary edges or triples to our graphs. The second research line is more comparable to our approach with the main difference being that R-GCNs incorporate information from neighborhoods via message passing steps, integrated into the model architecture, as opposed to sampling walks or paths in a graph on which a machine learning model is applied.

A similar GNN architecture with support for different edge types was proposed in Li et al. (2016) in the form of a recurrent neural network. Their model does not normalize messages after aggregation and does not regularize or decompose message weights in the presence of a large number of relation types, both of which we found to be helpful additions for modeling large-scale knowledge bases.

5.4 EXPERIMENTS

Here, we consider the task of classifying entities in a knowledge base. In order to infer, for example, the type of an entity (e.g., person or company), a successful model needs to reason about the relations with other entities that this entity is involved in.

5.4.1 Datasets

We evaluate our model on four benchmark datasets introduced by Ristoski et al. (2016) in *resource description framework* (RDF) format: AIFB, MUTAG, BGS, and AM. Relations in these datasets need not necessarily encode directed subject-object relations, but are also used to encode the presence, or absence, of a specific feature for a given entity. In each dataset, the targets to be classified are properties of a group of entities represented as nodes. The exact statistics of the datasets can be found in Table 5.1. For a more detailed description of the datasets the reader is referred to Ristoski et al. (2016). We remove relations that were used to create entity labels: *employs* and *affiliation* for AIFB, *isMutagenic* for MUTAG, *hasLithogenesis* for BGS, and *objectCategory* and *material* for AM.

For the entity classification benchmarks described in this chapter, the evaluation process differs subtly between publications. To eliminate these differences,

we repeated the baseline experiments in a uniform manner, using the canonical test/train split from Ristoski et al. (2016). We performed hyperparameter optimization on only the training set, running a single evaluation on the test set after hyperparameters were chosen for each baseline. This explains why the numbers we report differ slightly from those in the original publications (where cross-validation accuracy was reported).

Table 5.1: Number of entities, relations, edges, and classes along with the number of labeled entities for each of the datasets. *Labeled* denotes the subset of entities that have labels and that are to be classified.

Dataset	AIFB	MUTAG	BGS	AM
Entities	8,285	23,644	333,845	1,666,764
Relations	45	23	103	133
Edges	29,043	74,227	916,199	5,988,321
Labeled	176	340	146	1,000
Classes	4	2	2	11

5.4.2 Baselines

As a baseline for our experiments, we compare against prior state-of-the-art classification results from RDF2Vec embeddings (Ristoski and Paulheim, 2016), Weisfeiler-Lehman kernels (WL) (Shervashidze et al., 2011; Vries and Rooij, 2015), and hand-designed feature extractors (Feat) (Paulheim and Fümkranz, 2012). Feat assembles a feature vector from the in- and out-degree (per relation) of every labeled entity. RDF2Vec extracts walks on labeled graphs which are then processed using the Skipgram (Mikolov et al., 2013) model to generate entity embeddings, used for subsequent classification. See Ristoski and Paulheim (2016) for an in-depth description and discussion of these baseline approaches. All entity classification experiments were run on CPU nodes with 64GB of memory.

For WL, we use the *tree* variant of the Weisfeiler-Lehman subtree kernel from the Mustard library.[3] For RDF2Vec, we use an implementation provided by Ristoski and Paulheim (2016) which builds on Mustard. In both cases, we extract explicit feature vectors for the instance nodes, which are classified by

3 https://github.com/Data2Semantics/mustard

a linear SVM. For the MUTAG task, our preprocessing differs from that used in Vries and Rooij (2015) and Ristoski and Paulheim (2016) where for a given target relation (s, r, o) all triples connecting s to o are removed. Since o is a boolean value in the MUTAG data, one can infer the label after processing from other boolean relations that are still present. This issue is now mentioned in the Mustard documentation. In our preprocessing, we remove only the specific triples encoding the target relation.

5.4.3 Results

All results in Table 5.2 are reported on the train/test benchmark splits from Ristoski et al. (2016). We further set aside 20% of the training set as a validation set for hyperparameter tuning. For R-GCN, we report performance of a 2-layer model with 16 hidden units (10 for AM to reduce the memory footprint), basis function decomposition (Eq. 5.3), and trained with Adam (Kingma and Ba, 2014) for 50 epochs using a learning rate of 0.01. The normalization constant is chosen as $c_{i,r} = |\mathcal{N}_i^r|$, i.e., we average all incoming messages from a particular relation type. We found this setting to perform better than averaging all messages across relation types or not normalizing at all.

Table 5.2: Entity classification results in accuracy (average and standard error over 10 runs) for a feature-based baseline (see main text for details), WL (Shervashidze et al., 2011; Vries and Rooij, 2015), RDF2Vec (Ristoski and Paulheim, 2016), and R-GCN (this work). Test performance is reported on the train/test set splits provided by Ristoski et al. (2016).

Model	AIFB	MUTAG	BGS	AM
Feat	55.55 ± 0.00	77.94 ± 0.00	72.41 ± 0.00	66.66 ± 0.00
WL	80.55 ± 0.00	$\mathbf{80.88} \pm 0.00$	86.20 ± 0.00	87.37 ± 0.00
RDF2Vec	88.88 ± 0.00	67.20 ± 1.24	$\mathbf{87.24} \pm 0.89$	88.33 ± 0.61
R-GCN (Ours)	$\mathbf{95.83} \pm 0.62$	73.23 ± 0.48	83.10 ± 0.80	$\mathbf{89.29} \pm 0.35$

Hyperparameters for baselines are chosen according to the best model performance in Ristoski and Paulheim (2016), i.e., WL: 2 (tree depth), 3 (number of iterations); RDF2Vec: 2 (WL tree depth), 4 (WL iterations), 500 (embedding size), 5 (window size), 10 (SkipGram iterations), 25 (number of negative samples). We choose the regularization constant $C \in \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$

of the SVM classifier used in the baselines based on performance on a 80/20 train/validation split (of the original training set).

For R-GCN, we choose an $l2$ penalty on first layer weights $C_{l2} \in \{0, 5 \cdot 10^{-4}\}$ and the number of basis functions $B \in \{0, 10, 20, 30, 40\}$ based on validation set performance, where $B = 0$ refers to no basis decomposition. We summarize the final hyperparameter choices used in our experiments for the R-GCN model in Table 5.3. Our implementation is available under `https://github.com/tkipf/relational-gcn`.

Table 5.3: Hyperparameter choices based on validation set performance for 2-layer R-GCN model.

R-GCN setting	AIFB	MUTAG	BGS	AM
$l2$ penalty	0	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$
# basis functions	0	30	40	40
# hidden units	16	16	16	10

The R-GCN model performs comparable to or better than the baselines on AIFB and AM. We further find that our proposed basis function decomposition improves predictive performance for all but the smallest dataset (AIFB) — see Table 5.3. We find that there is a gap in performance on MUTAG and BGS, which could be related to the nature of these datasets. MUTAG is a dataset of molecular graphs, where relations either indicate atomic bonds or the presence of a certain feature. BGS is a dataset of rock types with hierarchical feature descriptions, where relations encode the presence of a certain feature or feature hierarchy. These feature-encoding nodes are typically of very high degree as they connect to all nodes in the graph that are associated with this particular feature. Hence they will serve as a high-degree 'hub', where many incoming messages in the R-GCN model are averaged, which can potentially negatively affect model performance. A promising way to overcome this limitation is to introduce a self-attention mechanism (Vaswani et al., 2017), i.e., to replace the normalization constant $1/c_{i,r}$ with data-dependent attention weights $a_{i,j,r}$.

5.5 CONCLUSION

We have introduced relational graph convolutional networks (R-GCNs) and we have demonstrated their effectiveness at the example of entity classification in multi-relational data. Variants of R-GCNs have since found application in a wide range of domains, such as 1) in recommender systems (Berg et al., 2017), where an encoder based on an R-GCN is coupled to a decoder based on a bilinear scoring function (similar to the relational link prediction model in Schlichtkrull and Kipf et al., 2018), 2) in machine translation (Bastings et al., 2017), where the graph is given by syntactic relations, and 3) in molecular synthesis (De Cao and Kipf, 2018; You et al., 2018), where relations are given by the bond types between atoms in a molecule.

Given the promising performance of graph-based models of this class on tasks where the graph structure is explicitly provided by the data, a natural question to ask is whether the structural inductive bias of graph neural networks can also be utilized for tasks where graph structure is *not* provided by the data, but has to be inferred. We will explore this setting — and related questions — in the second part of this thesis.

Part II

Learning with Implicit Structure

MOTIVATION AND SUMMARY

The world around us and our understanding of it is rich in structure: we perceive everyday scenes in terms of objects and their relations, much of our understanding of physics relies on structuring the world into sub-components and their interactions, and we often find it useful to structure the flow of time in terms of events and episodes. These are examples of *implicit* structure, where the sensory observation or data (e.g., an image) might not contain an explicit description of the underlying structure (e.g., objects in a scene), but where this structure has to be inferred.

This part of the thesis explores how we can develop models that are capable of inferring hidden structure in the form of interactions between components, events in a sequence, or objects and their relations in a scene and how they are affected by actions. For problems involving interactions or relations between components, we will see that graph-structured neural architectures (i.e., GNNs) allow us to effectively model hidden structure, even in the absence of any explicit graph structure provided by the data. We will use similar insights to develop an effective architecture for inferring hidden temporal structure in sequential data.

In Chapter 6, we introduce the *neural relational inference* (NRI) model for inference of hidden relations in interacting systems, such as multi-particle physical systems.

Chapter 7 introduces the *compositional imitation learning and execution* (CompILE) model that is capable of inferring hidden structure in sequential data. We explore this model at the example of latent program inference in the context of imitation learning.

In the final chapter in this part of the thesis, Chapter 8, we introduce the *contrastively-trained structured world model* (C-SWM), a model for discovering objects and for modeling the effect of actions in multi-object environments from raw pixel observations.

6

NEURAL RELATIONAL INFERENCE FOR INTERACTING SYSTEMS

6.1 INTRODUCTION

Interacting systems are prevalent in nature and in society, from dynamical systems in physics and biology, to multi-agent systems such as city traffic or sports games, to complex societal dynamics. The dynamics of such systems are governed by individual components and their interactions, which can give rise to complex behavior in the overall system. Modeling these type of dynamics is challenging: often, we only have access to individual trajectories, without knowledge of the underlying interactions or dynamical model.

To address this problem, we introduce the *neural relational inference* (NRI) model in this chapter[1] and in Kipf and Fetaya et al. (2018). The NRI model addresses the problem of inferring hidden interaction structure (see Figure 6.1) while simultaneously learning the dynamical model of the interacting system in an unsupervised way.

NRI uses a graph neural network (GNN) (Scarselli et al., 2009; Li et al., 2016; Kipf and Welling, 2017; Gilmer et al., 2017; Battaglia et al., 2018) over a discrete *latent* graph with multiple edge types to model the dynamics of an interacting system. The goal of learning in NRI is to perform inference over these discrete latent variables, while learning a predictive model for the dynamical evolution of the system. Using a probabilistic latent variable model allows us to incor-

1 This chapter is based on our ICML 2018 publication (Kipf and Fetaya et al., 2018). The first two authors contributed equally to this publication. An earlier version of this paper appeared as arXiv preprint `arXiv:1802.04687`. Permission was given by the co-authors for reproduction in this thesis.

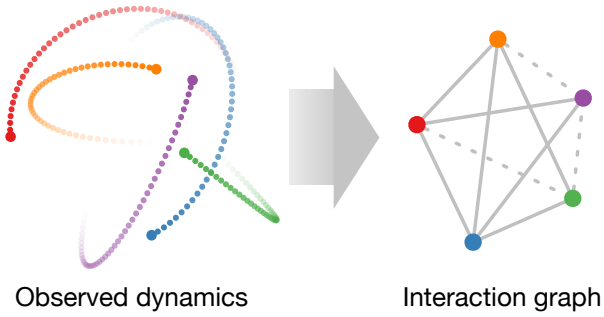


Figure 6.1: Physical simulation of 2D particles coupled by invisible springs (*left*) according to a latent interaction graph (*right*). In this example, solid lines between two particle nodes denote connections via springs whereas dashed lines denote the absence of a coupling. In general, multiple, directed edge types – each with a different associated relation – are possible.

porate prior beliefs about the graph structure, such as sparsity, in a principled manner.

In a range of experiments on physical simulations, we show that our NRI model possesses a favorable inductive bias that allows it to discover ground-truth physical interactions with high accuracy in a completely unsupervised way. We further show on real motion capture data that our model can learn a very small number of edge types that enable it to accurately predict the dynamics many time steps into the future.

6.2 METHODS

Our NRI model consists of two parts trained jointly: an encoder that predicts the interactions given the trajectories, and a decoder that learns the dynamical model given the interaction graph.

More formally, our input consists of trajectories of N objects. We denote by $\mathbf{x}_i^t$ the feature vector of object i at time t, e.g., location and velocity. We denote by $\mathbf{x}^t = (\mathbf{x}_1^t, ..., \mathbf{x}_N^t)$ the features of all N objects at time t, and we denote by $\mathbf{x}_i = (\mathbf{x}_i^1, ..., \mathbf{x}_i^T)$ the trajectory of object i, where T is the total number of time steps. Lastly, we mark the whole trajectories by $\mathbf{x} = (\mathbf{x}^1, ..., \mathbf{x}^T)$. We assume that the dynamics can be modeled by a GNN given an unknown graph $\mathbf{z}$ where $\mathbf{z}_{(i,j)}$ represents the discrete edge type between objects i and j. The task is to simultaneously learn to predict the edge types and learn the dynamical model

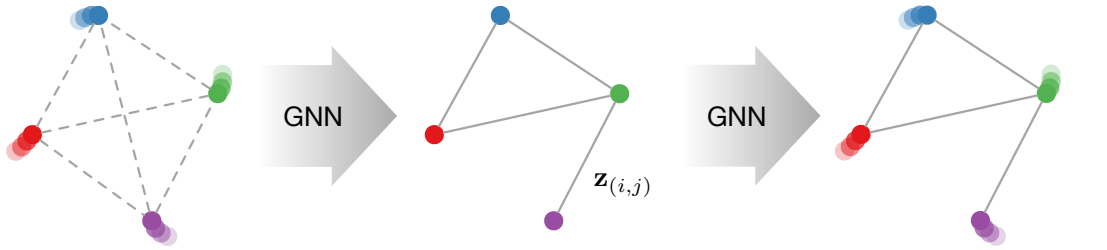


Figure 6.2: The NRI model consists of two jointly trained parts: An encoder that predicts a probability distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ over the latent interactions given input trajectories; and a decoder that generates trajectory predictions $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ conditioned on both the latent code of the encoder and the previous time step of the trajectory. The encoder takes the form of a GNN with multiple rounds of node-to-edge ($v \to e$) and edge-to-node ($e \to v$) message passing, whereas the decoder runs multiple GNNs in parallel, one for each edge type supplied by the latent code of the encoder $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$.

in an unsupervised way. Note that we use lower-case symbols for both node features $\mathbf{x}_i$ and edge types $\mathbf{z}_{(i,j)}$ in this chapter to simplify notation, even when they refer to collections $\mathbf{x}$ and $\mathbf{z}$ for all objects and interactions, respectively.

We formalize our model as a variational autoencoder (VAE) (Kingma and Welling, 2013; Rezende et al., 2014) that maximizes the ELBO:

$$\mathcal{L} = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})] - D_{\mathrm{KL}}[q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z})] \tag{6.1}$$

The encoder $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ returns a factorized distribution of $\mathbf{z}_{(i,j)}$, where $\mathbf{z}_{(i,j)}$ is a discrete categorical variable representing the edge type between objects i and j. We use a one-hot representation of the K edge types for $\mathbf{z}_{(i,j)}$.

The decoder

$$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\mathbf{x}^{t+1}|\mathbf{x}^t, ..., \mathbf{x}^1, \mathbf{z}) \tag{6.2}$$

models $p_{\boldsymbol{\theta}}(\mathbf{x}^{t+1}|\mathbf{x}^t, ..., \mathbf{x}^1, \mathbf{z})$ with a GNN given the latent graph structure $\mathbf{z}$.

The prior $p_{\boldsymbol{\theta}}(\mathbf{z}) = \prod_{i \neq j} p_{\boldsymbol{\theta}}(\mathbf{z}_{(i,j)})$ is a factorized uniform distribution over edge types. If one edge type is 'hard coded' to represent 'non-edge' (no messages being passed along this edge type), we can use an alternative prior with higher probability on the 'non-edge' label. This will encourage sparser graphs.

There are some notable differences between our model and the original formulation of the VAE (Kingma and Welling, 2013). First, in order to avoid the common issue in VAEs of the decoder ignoring the latent code $\mathbf{z}$ (Chen et al.,

2016), we train the decoder to predict multiple time steps and not a single step as the VAE formulation requires. This is necessary since interactions often only have a small effect in the time scale of a single time step. Second, the latent distribution is discrete, so we use a continuous relaxation (Jang et al., 2017; Maddison et al., 2017) in order to use the reparameterization trick (Kingma and Welling, 2013). Lastly, we note that we do not learn $p(\mathbf{x}^1|\mathbf{z})$ (i.e., for $t = 1$) as we are interested in the dynamics and interactions, and this does not have any effect on either (but would be easy to include if there was a need).

The overall model is schematically depicted in Figure 6.2. In the following, we describe the encoder and decoder components of the model in detail.

6.2.1 Encoder

At a high level, the goal of the encoder is to infer pairwise interaction types $\mathbf{z}_{(i,j)}$ given observed trajectories $\mathbf{x} = (\mathbf{x}^1, ..., \mathbf{x}^T)$. Since we do not know the underlying graph, we can use a GNN on the fully-connected graph to predict the latent graph structure. For an introduction to GNNs, see Chapter 2.

More formally, we model the encoder as $q_{\boldsymbol{\phi}}(\mathbf{z}_{(i,j)}|\mathbf{x}) = \text{softmax}(f_{\text{enc},\boldsymbol{\phi}}(\mathbf{x})_{i,j,1:K})$, where $f_{\text{enc},\boldsymbol{\phi}}(\mathbf{x})$ is a GNN acting on the fully-connected graph (without self-loops). Given input trajectories $\mathbf{x}_1, ..., \mathbf{x}_N$ our encoder computes the following message passing operations:

$$\mathbf{h}_j^1 = f_{\text{emb}}(\mathbf{x}_j) \tag{6.3}$$

$$v \to e: \quad \mathbf{h}_{(i,j)}^1 = f_e^1([\mathbf{h}_i^1, \mathbf{h}_j^1]) \tag{6.4}$$

$$e \to v: \quad \mathbf{h}_j^2 = f_v^1\left(\sum_{i \neq j} \mathbf{h}_{(i,j)}^1\right) \tag{6.5}$$

$$v \to e: \quad \mathbf{h}_{(i,j)}^2 = f_e^2([\mathbf{h}_i^2, \mathbf{h}_j^2]) \tag{6.6}$$

Finally, we model the edge type posterior as $q_{\boldsymbol{\phi}}(\mathbf{z}_{(i,j)}|\mathbf{x}) = \text{softmax}(\mathbf{h}_{(i,j)}^2)$ where $\boldsymbol{\phi}$ summarizes the parameters of the neural networks in Eqs. 6.3–6.6. Note that in a single pass, Eqs. 6.3–6.4, the embedding $\mathbf{h}_{(i,j)}^1$ only depends on $\mathbf{x}_i$ and $\mathbf{x}_j$, ignoring interactions with other nodes, while $\mathbf{h}_j^2$ uses information from the whole graph.

The functions $f_{(...)}$ are neural networks that map between the respective representations. In our experiments we use either fully-connected networks (MLPs) or 1D convolutional networks (CNNs) with attentive pooling similar to Lin et al. (2017).

6.2.2 Sampling

It is straightforward to sample from $q_{\phi}(\mathbf{z}_{(i,j)}|\mathbf{x})$, however we cannot use the reparametrization trick to backpropagate though the sampling as our latent variables are discrete. A recently popular approach to handle this difficulty is to sample from a continuous approximation of the discrete distribution (Jang et al., 2017; Maddison et al., 2017) and use the repramatrization trick to get (biased) gradients from this approximation. We used the concrete distribution (Maddison et al., 2017) where samples are drawn as:

$$\mathbf{z}_{(i,j)} = \text{softmax}((\mathbf{h}^2_{(i,j)} + \mathbf{g})/\tau) \tag{6.7}$$

where $\mathbf{g} \in \mathbb{R}^K$ is a vector of i.i.d. samples drawn from a Gumbel$(0, 1)$ distribution and τ (softmax temperature) is a parameter that controls the 'smoothness' of the samples. This distribution converges to one-hot samples from our categorical distribution when $\tau \to 0$.

6.2.3 Decoder

The task of the decoder is to predict the future continuation of the interacting system's dynamics $p_{\theta}(\mathbf{x}^{t+1}|\mathbf{x}^t, ..., \mathbf{x}^1, \mathbf{z})$. Since the decoder is conditioned on the graph $\mathbf{z}$ we can in general use any GNN model as our decoder.

For physics simulations the dynamics are Markovian $p_{\theta}(\mathbf{x}^{t+1}|\mathbf{x}^t, ..., \mathbf{x}^1, \mathbf{z}) = p_{\theta}(\mathbf{x}^{t+1}|\mathbf{x}^t, \mathbf{z})$, if the state is location and velocity, and $\mathbf{z}$ is the ground-truth graph. For this reason we use a GNN similar to interaction networks (Battaglia et al., 2016); unlike interaction networks we have a separate neural network for each edge type. More formally:

$$v \to e: \quad \tilde{\mathbf{h}}^t_{(i,j)} = \sum_k z_{i,j,k} \tilde{f}^k_e([\mathbf{x}^t_i, \mathbf{x}^t_j]) \tag{6.8}$$

$$e \to v: \quad \boldsymbol{\mu}^{t+1}_j = \mathbf{x}^t_j + \tilde{f}_v(\textstyle\sum_{i \neq j} \tilde{\mathbf{h}}^t_{(i,j)}) \tag{6.9}$$

$$p(\mathbf{x}^{t+1}_j|\mathbf{x}^t, \mathbf{z}) = \mathcal{N}(\mathbf{x}^{t+1}_j; \boldsymbol{\mu}^{t+1}_j, \sigma^2 \mathbf{I}) \tag{6.10}$$

Note that $z_{i,j,k}$ denotes the k-th element of the vector $\mathbf{z}_{(i,j)}$ and σ^2 is a fixed variance. When $z_{i,j,k}$ is a discrete one-hot sample, the messages $\tilde{\mathbf{h}}^t_{(i,j)}$ are $\tilde{f}^k_e([\mathbf{x}^t_i, \mathbf{x}^t_j])$ for the selected edge type k, and for the continuous relaxation we get a weighted sum. Also note that since in Eq. 6.9 we add the present state $\mathbf{x}^t_j$, our model only learns the change in state $\Delta\mathbf{x}^t_j$.

6.2.4 Avoiding Degenerate Decoders

If we look at the ELBO, Eq. 6.1, the reconstruction loss term has the form $\sum_{t=1}^{T} \log[p_{\theta}(\mathbf{x}^t|\mathbf{x}^{t-1}, \mathbf{z})]$, which involves only single step predictions. One issue with optimizing this objective is that the interactions can have only a small effect on short-term dynamics. For example, in physics simulations a fixed velocity assumption can be a good approximation for a short time period. This leads to a sub-optimal decoder that ignores the latent edges completely and achieves only a marginally worse reconstruction loss.

We address this issue in two ways: First, we predict multiple steps into the future, where a 'degenerate' decoder (which ignores the latent edges) would perform much worse. Second, instead of having one neural network that computes the messages given $[\mathbf{x}_i^t, \mathbf{x}_j^t, \mathbf{z}_{(i,j)}]$, as was done in Battaglia et al. (2016), we have a separate MLP for each edge type. This makes the dependence on the edge type more explicit and harder to be ignored by the model.

Predicting multiple steps is implemented by replacing the correct input $\mathbf{x}^t$, with the predicted mean $\boldsymbol{\mu}^t$ for M steps (we used $M = 10$ in our experiments), then feed in the correct previous step and reiterate. More formally, if we denote our decoder as $\boldsymbol{\mu}_j^{t+1} = f_{\text{dec}}(\mathbf{x}_j^t)$ then we have:

$$
\begin{aligned}
\boldsymbol{\mu}_j^2 &= f_{\text{dec}}(\mathbf{x}_j^1) && \\
\boldsymbol{\mu}_j^{t+1} &= f_{\text{dec}}(\boldsymbol{\mu}_j^t) && t = 2, \ldots, M \\
\boldsymbol{\mu}_j^{M+2} &= f_{\text{dec}}(\mathbf{x}_j^{M+1}) && \\
\boldsymbol{\mu}_j^{t+1} &= f_{\text{dec}}(\boldsymbol{\mu}_j^t) && t = M+2, \ldots, 2M \\
&\ldots &&
\end{aligned}
$$

We are backpropagating through this whole process, and since the errors accumulate for M steps the degenerate decoder is now highly suboptimal.

6.2.5 Recurrent Decoder

In many applications the Markovian assumption used in Section 6.2.3 does not hold. To handle such applications we use a recurrent decoder that can model $p_{\theta}(\mathbf{x}^{t+1}|\mathbf{x}^t, ..., \mathbf{x}^1, \mathbf{z})$. Our recurrent decoder adds a GRU (Cho et al., 2014) unit to the GNN message passing operation. The GRU is a common architecture choice for implementing recurrent neural networks with gating operations that

alleviate the vanishing gradient problem (Hochreiter, 1991). With the GRU, our recurrent decoder takes the following form:

$$v \to e: \quad \tilde{\mathbf{h}}_{(i,j)}^t = \sum_k z_{i,j,k} \tilde{f}_e^k([\tilde{\mathbf{h}}_i^t, \tilde{\mathbf{h}}_j^t]) \tag{6.11}$$

$$e \to v: \quad \text{MSG}_j^t = \textstyle\sum_{i \neq j} \tilde{\mathbf{h}}_{(i,j)}^t \tag{6.12}$$

$$\tilde{\mathbf{h}}_j^{t+1} = \text{GRU}([\text{MSG}_j^t, \mathbf{x}_j^t], \tilde{\mathbf{h}}_j^t) \tag{6.13}$$

$$\boldsymbol{\mu}_j^{t+1} = \mathbf{x}_j^t + f_{\text{out}}(\tilde{\mathbf{h}}_j^{t+1}) \tag{6.14}$$

$$p(\mathbf{x}^{t+1}|\mathbf{x}^t, \mathbf{z}) = \mathcal{N}(\mathbf{x}^{t+1}; \boldsymbol{\mu}^{t+1}, \sigma^2 \mathbf{I}) \tag{6.15}$$

The input to the message passing operation is the recurrent hidden state at the previous time step. f_{out} denotes an output transformation, modeled by a small MLP. For each node i, the input to the GRU update is the concatenation of the aggregated messages MSG_j^{t+1}, the current input $\mathbf{x}_j^{t+1}$, and the previous hidden state $\tilde{\mathbf{h}}_j^t$.

If we wish to predict multiple time steps in the recurrent setting, the method suggested in Section 6.2.4 will be problematic. Feeding in the predicted (potentially incorrect) path and then periodically jumping back to the true path will generate artifacts in the learned trajectories. In order to avoid this issue we provide the correct input $\mathbf{x}_j^t$ in the first $(T - M)$ steps, and only utilize our predicted mean $\boldsymbol{\mu}_j^t$ as input at the last M time steps.

6.2.6 Training

Now that we have described all the elements, the training goes as follows: given training example $\mathbf{x}$, we first run the encoder and compute $q_{\boldsymbol{\phi}}(\mathbf{z}_{(i,j)}|\mathbf{x})$, then we sample $\mathbf{z}_{(i,j)}$ from the concrete reparameterizable approximation of $q_{\boldsymbol{\phi}}(\mathbf{z}_{(i,j)}|\mathbf{x})$. We then run the decoder to compute $\boldsymbol{\mu}^2, ..., \boldsymbol{\mu}^T$. The ELBO objective, Eq. 6.1, has two terms: the reconstruction error $\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})]$ and KL divergence $D_{\text{KL}}[q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z})]$. The reconstruction error is estimated by:

$$-\sum_j \sum_{t=2}^{T} \frac{||\mathbf{x}_j^t - \boldsymbol{\mu}_j^t||^2}{2\sigma^2} + \text{const}\,, \tag{6.16}$$

while the KL term for a uniform prior is just the sum of entropies (plus a constant):

$$\sum_{i \neq j} H(q_{\boldsymbol{\phi}}(\mathbf{z}_{(i,j)}|\mathbf{x})) + \text{const}\,. \tag{6.17}$$

As we use a reparameterizable approximation, we can compute gradients by backpropagation and optimize.

6.3 RELATED PRIOR WORK

Several prior works have studied the problem of learning the dynamics of a physical system from simulated trajectories (Battaglia et al., 2016; Guttenberg et al., 2016; Chang et al., 2017) and from generated video data (Watters et al., 2017; Steenkiste et al., 2018) with a GNN. Unlike our work they either assume a known graph structure or infer interactions implicitly.

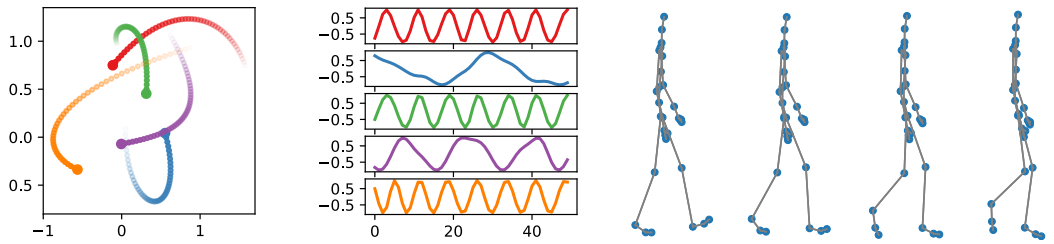
Related approaches using graph-based methods for human motion prediction include work by Alahi et al. (2016), where the graph is not learned but is based on proximity, and Le et al. (2017) tries to cluster agents into roles.

A number of related works (Monti et al., 2017; Duan et al., 2017; Hoshen, 2017; Veličković et al., 2018b; Garcia and Bruna, 2018; Steenkiste et al., 2018) parameterize messages in GNNs with a soft attention mechanism (Luong et al., 2015; Bahdanau et al., 2014). This equips these models with the ability to focus on specific interactions with neighbors when aggregating messages. Our work is different from this line of research, as we explicitly perform inference over the latent graph structure. This allows for the incorporation of prior beliefs (such as sparsity) and for an interpretable discrete structure with multiple relation types.

The problem of inferring interactions or latent graph structure has been investigated in other settings in different fields. For example, in causal reasoning Granger causality (Granger, 1969) infers causal relations. Another example from computational neuroscience is Linderman et al. (2016) and Linderman and Adams (2014) where they infer interactions between neural spike trains.

6.4 EXPERIMENTS

In this section, we evaluate our proposed NRI model on two simulated physical multi-particle systems and on a motion capture dataset. See Figure 6.3 for example observations in all datasets. Our encoder implementation uses two-layer fully-connected networks (MLPs) with 256 hidden units as our mes-



(a) Springs (2D) **(b)** Kuramoto (1D) **(c)** Motion capture data

Figure 6.3: Examples of trajectories used in our experiments from simulations of (a) particles connected by invisible springs and (b) phase-coupled oscillators according to the Kuramoto model (Kuramoto, 1975), and (c) example of a motion capture trajectory (human walking motion) from the CMU Motion Capture Database (CMU, 2003).

sage passing function. For our decoder we used fully-connected networks or alternatively a recurrent decoder. See Appendix 6.A for further implementation details on encoders and decoders. Optimization was performed using the Adam algorithm (Kingma and Ba, 2014). Our implementation uses PyTorch (Paszke et al., 2017) and is available under `https://github.com/tkipf/nri`. Additional experimental details are provided in Appendix 6.B.

6.4.1 Physics Simulations

We experimented with two simulated systems: particles connected by springs and phase-coupled oscillators (Kuramoto model; Kuramoto, 1975). The springs are modeled via Hooke's law $\mathbf{F}_{i,j} = -k(\mathbf{r}_i - \mathbf{r}_j)$ where $\mathbf{F}_{i,j}$ is the force applied to particle i by particle j, $k \in \{0, 0.1\}$ is the spring constant and $\mathbf{r}_i$ is the 2D location vector of particle i. All masses are $m_i = 1$. The initial location is sampled from a Gaussian $\mathcal{N}(\mathbf{r}_i; \mathbf{0}, \sigma\mathbf{I})$ with $\sigma = 0.5$, and the initial velocity is a random vector of norm 0.5. Given the initial locations and velocity we can simulate the trajectories by solving Newton's equations of motion. We do this by leapfrog integration using a step size of 0.001 and then subsample each 100 steps to get our training and testing trajectories.

To test whether our model can also learn interactions in non-linear settings, we simulate the Kuramoto model which is governed by the following differential equation:

$$\frac{d\phi_i}{dt} = \omega_i + \sum_{j \neq i} k_{i,j} \sin(\phi_i - \phi_j) \tag{6.18}$$

Table 6.1: Accuracy (in %) of unsupervised interaction recovery.

	Springs		**Kuramoto**	
Number of objects	5	10	5	10
Correlation (path)	52.4±0.0	50.4±0.0	62.8±0.0	59.3±0.0
Correlation (LSTM)	52.7±0.9	54.9±1.0	54.4±0.5	56.2±0.7
NRI (simulation decoder)	**99.8**±0.0	**98.2**±0.0	–	–
NRI (learned decoder)	**99.9**±0.0	**98.4**±0.0	**96.0**±0.1	**75.7**±0.3
Supervised	99.9±0.0	98.8±0.0	99.7±0.0	97.1±0.1

with phases ϕ_i, coupling constants $k_{i,j} \in \{0, 1\}$, and intrinsic frequencies ω_i. We simulate 1D trajectories by solving Eq. 6.18 with a fourth-order Runge-Kutta integrator with step size 0.01. We simulate phase-coupled oscillators in 1D with intrinsic frequencies ω_i and initial phases $\phi_i^{t=1}$ sampled uniformly from $[1, 10)$ and $[0, 2\pi)$, respectively. We obtain 1D trajectories by solving Eq. 6.18 with a fourth-order Runge-Kutta integrator. We create trajectories $\mathbf{x}_i$ by concatenating $\frac{d\phi_i}{dt}$, $\sin\phi_i$, and the intrinsic frequencies ω_i (copied for every time step as ω_i are static).

These settings allow us to attempt to learn the dynamics and interactions when the ground-truth interactions are known. These systems, controlled by simple rules, can exhibit complex dynamics. The objects do or do not interact with equal probability. We generate 50k training examples, and 10k validation and test examples for all tasks.

We note that the simulations are differentiable and so we can use it as a ground-truth decoder to train the encoder. We used an external code base (Laszuk, 2017) for stable integration of the differential equation governing the Kuramoto model and therefore do not have access to gradient information in this particular simulation.

Results

We evaluated our NRI model on both simulated physical systems and compared our performance, both in terms of future state prediction and in terms of accuracy of estimating the edge types in an unsupervised manner.

For edge prediction, we compare to the 'gold standard', i.e., training our encoder in a supervised way given the ground-truth labels. We also com-

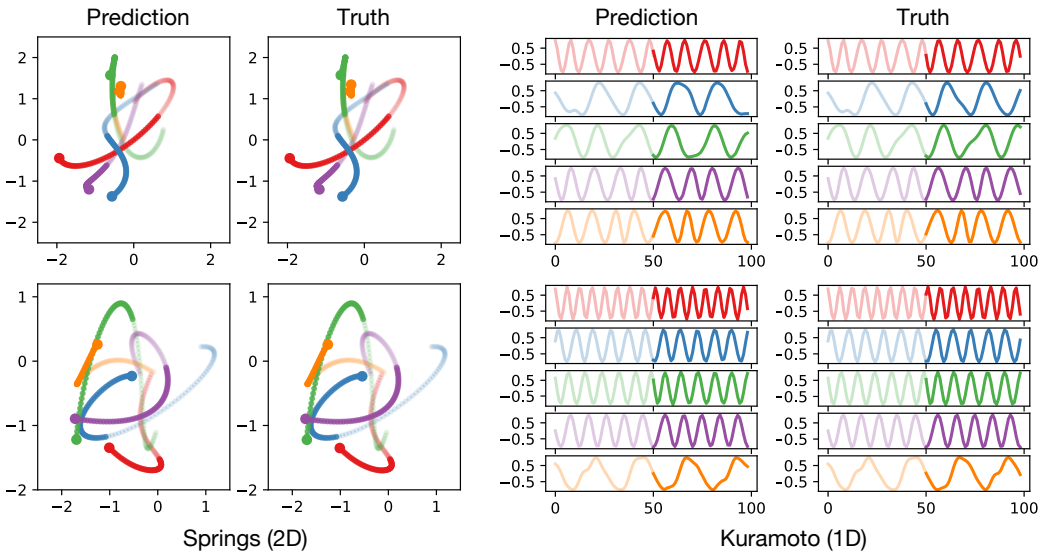


Figure 6.4: Trajectory predictions from a trained NRI model (unsupervised). Semi-transparent paths denote the first 49 time steps of ground-truth input to the model, from which the interaction graph is estimated. Solid paths denote self-conditioned model predictions.

Table 6.2: Mean squared error (MSE) in predicting future states for simulations with 5 interacting objects.

	Springs			**Kuramoto**		
Prediction steps	1	10	20	1	10	20
Static	7.93e-5	7.59e-3	2.82e-2	5.75e-2	3.79e-1	3.39e-1
LSTM (single)	2.27e-6	4.69e-4	4.90e-3	7.81e-4	3.80e-2	8.08e-2
LSTM (joint)	4.13e-8	2.19e-5	7.02e-4	**3.44e-4**	**1.29e-2**	4.74e-2
NRI (full graph)	1.66e-5	1.64e-3	6.31e-3	2.15e-2	5.19e-2	8.96e-2
NRI (learned)	**3.12e-8**	**3.29e-6**	**2.13e-5**	1.40e-2	2.01e-2	**3.26e-2**
NRI (true graph)	1.69e-11	1.32e-9	7.06e-6	1.35e-2	1.54e-2	2.19e-2

pare to the following baselines: our NRI model with the ground-truth simulation decoder, NRI (sim.), and two correlation-based baselines, Corr. (path) and Corr. (LSTM). Corr. (path) estimates the interaction graph by thresholding the matrix of correlations between trajectory feature vectors. Corr. (LSTM) trains an LSTM (Hochreiter and Schmidhuber, 1997) with shared parameters to model each trajectory individually and calculates correlations between the final hidden states to arrive at an interaction matrix after thresholding.

Results for the unsupervised interaction recovery task are summarized in Table 6.1 (average over 5 runs and standard error). As can be seen, the unsupervised NRI model, NRI (learned), greatly surpasses the baselines and recovers the ground-truth interaction graph with high accuracy on most tasks. For the springs model our unsupervised method is comparable to the supervised 'gold standard' benchmark. We note that our supervised baseline is similar to the work by Santoro et al. (2017), with the difference that we perform multiple rounds of message passing in the graph.

For future state prediction we compare to the static baseline, i.e., $\mathbf{x}^{t+1} = \mathbf{x}^t$, two LSTM baselines, and a full graph baseline. One LSTM baseline, marked as 'single', runs a separate LSTM (with shared weights) for each object. The second, marked as 'joint' concatenates all state vectors and feeds it into one LSTM that is trained to predict all future states simultaneously. Note that the latter will only be able to operate on a fixed number of objects (in contrast to the other models).

In the full graph baseline, we use our message passing decoder on the fully-connected graph without edge types, i.e., without inferring edges. This is similar to the model used in Watters et al. (2017). We also compare to the 'gold standard' model, denoted as NRI (true graph), which is training only a decoder using the ground-truth graph as input. The latter baseline is comparable to previous works such as interaction networks (Battaglia et al., 2016).

In order to have a fair comparison, we generate longer test trajectories and only evaluate on the last part unseen by the encoder. Specifically, we run the encoder on the first 49 time steps (same as in training and validation), then predict with our decoder the following 20 unseen time steps. For the LSTM baselines, we first have a 'burn-in' phase where we feed the LSTM the first 49 time steps, and then predict the next 20 time steps. This way both algorithms have access to the first 49 steps when predicting the next 20 steps. We show mean squared error (MSE) results in Table 6.2, and note that our results are better than using LSTM for long-term prediction. Example trajectories predicted by our NRI (learned) model for up to 50 time steps are shown in Figure 6.4.

For the Kuramoto model, we observe that the LSTM baselines excel at smoothly continuing the shape of the waveform for short time frames, but fail to model the long-term dynamics of the interacting system.

6.4.2 Motion Capture

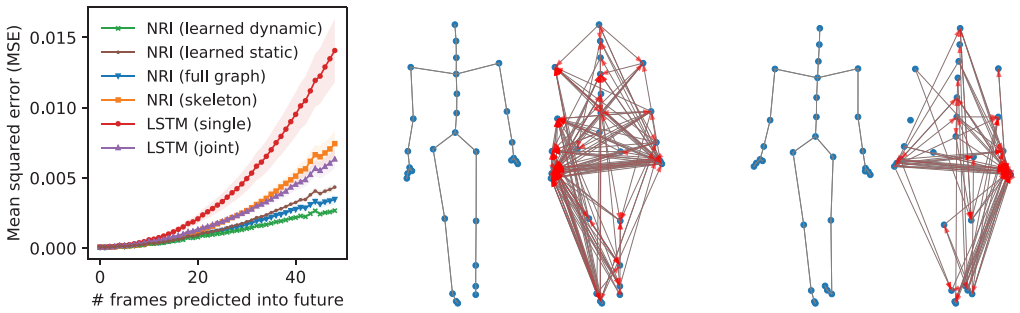
The CMU Motion Capture Database (CMU, 2003) is a large collection of motion capture recordings for various tasks (such as walking, running, and dancing) performed by human subjects. We use recorded walking motion data of a single subject (subject #35). The data is in the form of 31 3D trajectories, each tracking a single joint. We split the different walking trials into non-overlapping training (11 trials), validation (4 trials) and test sets (7 trials). We provide both position and velocity data. We train our NRI model with an MLP encoder and RNN decoder on this data using 2 or 4 edge types where one edge type is 'hard-coded' as non-edge, i.e., messages are only passed on the other edge types. We found that experiments with 2 and 4 edge types give almost identical results, with two edge types being comparable in capacity to the fully connected graph baseline while four edge types (with sparsity prior) are more interpretable and allow for easier visualization.

Dynamic Graph Re-Evaluation

We find that the learned graph depends on the particular phase of the motion (Figure 6.5), which indicates that the ideal underlying graph is dynamic. To account for this, we dynamically re-evaluate the NRI encoder for every time step during testing, effectively resulting in a dynamically changing latent graph that the decoder can utilize for more accurate predictions.

Results

The quantitative results for our method and the same baselines used in Section 6.4.1 can be seen in Figure 6.5. As one can see, we outperform the fully-connected graph setting in long-term predictions, and both models outperform the LSTM baselines. Dynamic graph re-evaluation significantly improves predictive performance for this dataset compared to a static baseline. One interesting observation is that the skeleton graph is quite suboptimal, which is surprising as the skeleton is the 'natural' graph. When examining the edges found by our model (trained with 4 edge types and a sparsity prior) we see an edge type that mostly connects a hand to other extremities, especially the opposite hand, as seen in Figure 6.5. This can seem counter-intuitive as one might assume that the important connections are local, however we note that



(a) Test MSE comparison **(b)** Latent graph (left step) **(c)** Latent graph (right step)

Figure 6.5: (a) Test mean squared error (MSE) comparison on motion capture data. (b-c) Learned latent graphs (trained with 4 edge types, showing the second edge type). Trained with sparsity prior for easier visualization. We visualize test data not seen during training. Skeleton shown for reference. Red arrowheads denote directionality of a learned edge. The edge type shown favors a specific hand depending on the state of the movement and gathers information mostly from other extremities.

some leading approaches for modeling motion capture data (Jain et al., 2016) do indeed include hand to hand interactions.

6.5 CONCLUSION

In this chapter, we have introduced NRI, a method to simultaneously infer relational structure while learning the dynamical model of an interacting system. In a range of experiments with physical simulations we have demonstrated that our NRI model is highly effective at unsupervised recovery of ground-truth interaction graphs. We further found that it can model the dynamics of interacting physical systems and of real motion tracking data while learning reasonably interpretable edge types.

Many real-world examples, in particular multi-agent systems such as traffic, can be understood as an interacting system where the interactions are dynamic. While our model is trained to discover static interaction graphs, we have demonstrated that it is possible to apply a trained NRI model to this evolving case by dynamically re-estimating the latent graph. Nonetheless, our solution is limited to static graphs during training and extending the NRI model so that it can explicitly account for dynamic latent interactions even at training time is an interesting avenue for future work.

CHAPTER APPENDIX

6.A IMPLEMENTATION DETAILS

We will describe here the details of our encoder and decoder implementations.

6.A.1 Encoders

MLP Encoder

The basic building block of our MLP encoder is a 2-layer MLP with hidden and output dimension of 256, with batch normalization (Ioffe and Szegedy, 2015) and ELU activations (Clevert et al., 2015), used for both the node and the edge update functions, f_v and f_e respectively. We use a total of two rounds of message passing, i.e., two node update functions and two edge update functions. Before feeding the trajectory to the first node MLP, we flatten the trajectory into a single feature vector (i.e., we concatenate the feature vectors for all time steps). We utilize a skip connection between the first and the second edge update function, i.e., we concatenate the output of the first edge update function with the input to the second edge update function, which we found to improve performance.

CNN Encoder

The CNN encoder uses the identity function for the first node update f_v and replaces the first edge update function f_e with a two-layer 1D convolutional network with filter width of 5 and 256 feature maps per layer, each followed by batch normalization. We further found it helpful to use a pooling layer with kernel size 2 after the first CNN layer. The CNN is followed by a self-attentive pooling layer (Lin et al., 2017) that summarizes the trajectory into a single feature vector. Lastly, we use a two-layer MLP as in the MLP encoder on this feature vector. The rest of the architecture follows the MLP encoder.

Using a CNN with attention mechanism allows for encoding with changing trajectory size, and is also appropriate for tasks where the interaction can be strong for a small fraction of time.

6.A.2 Decoders

MLP Decoder

The MLP decoder uses a single round of message passing only and uses two-layer MLPs (three layers for the node update function) with ReLU activations, 256 hidden units and no batch normalization. We utilize K edge update functions, where K is the number of edge types.

RNN Decocder

The RNN decoder uses the same architecture as the MLP decoder, but adds a gated recurrent unit (GRU) module (Chung et al., 2014) with 256 hidden units prior to the final edge update function.

6.B EXPERIMENT DETAILS

All experiments were run using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 0.0005, decayed by a factor of 0.5 every 200 epochs. Unless otherwise noted, we train with a batch size of 128. The concrete distribution (Maddison et al., 2017) is used with $\tau = 0.5$. During testing, we replace the concrete distribution with a categorical distribution to obtain discrete latent edge types. Physical simulation experiments were run for 500 training epochs. For motion capture data we used 200 training epochs, as models tended to converge earlier. We saved model checkpoints after every epoch whenever the validation set performance (measured by path prediction MSE) improved and loaded the best performing model for test set evaluation. We observed that using significantly higher learning rates than 0.0005 often produced suboptimal decoders that ignored the latent graph structure.

6.B.1 Physics Simulations

The springs and Kuramoto datasets both contain 50k training instances and 10k validation and test instances. Training and validation trajectories were of length 49 while test trajectories continue for another 20 time steps (50 for visualization). We train an MLP encoder for the springs experiment, and CNN encoder for the Kuramoto experiments. All experiments used MLP decoders and two edge types. For the Kuramoto model experiments, we explicitly hard-coded the first edge type as a 'non-edge', i.e., no messages are passed along edges of this type.

The overall input/output dimension of our model is 4 for the springs experiments (2D position and velocity) and 3 for the Kuramoto model experiments (phase-difference, amplitude and intrinsic frequency). During training, we use teacher forcing in every 10-th time step (i.e., every 10-th time step, the model receives ground truth input, otherwise it receives its previous prediction as input). As we always have two edge types in these experiments and their ordering is arbitrary (apart from the Kuramoto model where we assign a special role to edge type 1), we choose the ordering for which the accuracy is highest.

Baselines

In edge recovery experiments, we report the following baselines along with the performance of our NRI (learned) model:

- **Corr. (path)**: We calculate a correlation matrix R, where $R_{i,j} = \frac{C_{i,j}}{\sqrt{C_{i,i}C_{j,j}}}$ with $C_{i,j}$ being the covariance between all trajectories $\mathbf{x}_i$ and $\mathbf{x}_j$ (for objects i and j) in the training and validation sets. We determine an ideal threshold θ so that $A_{i,j} = 1$ if $R_{i,j} > \theta$ and $A_{i,j} = 0$ otherwise, based on predictive accuracy on the combined training and validation set. $A_{i,j}$ denotes the presence of an interaction edge (arbitrary type) between object i and j. We repeat the same procedure for the absolute value of $R_{i,j}$, i.e., $A_{i,j} = 1$ if $|R_{i,j}| > \theta'$ and $A_{i,j} = 0$ otherwise. Lastly, we pick whichever of the two (θ or θ') produced the best match with the ground truth graph (i.e., highest accuracy score) and report test set accuracy with this setting.

- **Corr. (LSTM)**: Here, we train a two-layer LSTM (Hochreiter and Schmidhuber, 1997) with shared parameters and 256 hidden units that models each trajectory individually. It is trained to predict the position and veloc-

ity for every time step directly and is conditioned on the previous time steps. The input to the model is passed through a two-layer MLP (256 hidden units and ReLU activations) before it is passed to the LSTM, similarly we pass the LSTM output (last time step) through a two-layer MLP (256 hidden units and ReLU activation on the hidden layer). We provide ground truth trajectory information as input at every time step. We train to minimize MSE between model prediction and ground truth path. We train this model for 10 epochs and finally apply the same correlation matrix procedure as in Corr. (path), but this time calculating correlations between the output of the second LSTM layer at the last time step (instead of using the raw trajectory features). The LSTM is only trained on the training set. The optimal correlation threshold is estimated using the combined training and validation set.

- **NRI (sim.)**: In this setting, we replace the decoder of the NRI model with the ground-truth simulator (i.e., the integrator of the Newtonian equations of motion). We implement the springs simulator in PyTorch which gives us access to gradient information. We train the overall model with the same settings as the original NRI (learned) model by backpropagating directly through the simulator. We find that for the springs simulation, a single leap-frog integration step is sufficient to closely approximate the trajectory of the original simulation, which was generated with 100 leap-frog steps per time step.

- **Supervised**: For this baseline, we train the encoder in isolation and provide ground-truth interaction graphs as labels. We train using a cross-entropy error and monitor the validation accuracy (edge prediction) for model checkpointing. We train with dropout (Srivastava et al., 2014) of $p = 0.5$ on the hidden layer representation of every MLP in the encoder model, in order to avoid overfitting.

In the path prediction experiments, we use the following baselines along with our NRI (learned) model:

- **Static**: This baseline simply copies the previous state vector $\mathbf{x}^{t+1} = \mathbf{x}^t$.

- **LSTM (single)**: Same as the LSTM model in Corr. (LSTM), but trained to predict the state vector difference at every time step (as in the NRI model).

Instead of providing ground truth input at every time step, we use the same training protocol as for an NRI model with recurrent decoder.

- **LSTM (joint)**: This baseline differs from LSTM (single) in that it concatenates the input representations from all objects after passing them through the input MLP. This concatenated representation is fed into a single LSTM where the hidden unit number is multiplied by the number of objects — otherwise same setting as LSTM (single). The output of the second LSTM layer at the last time step is then divided into vectors of same size, one for each object, and fed through the output MLP to predict the state difference for each object separately. LSTM (joint) is trained with same training protocol as the LSTM (single) model.
- **NRI (full graph)**: For this model, we keep the latent graph fixed (fully-connected on edge type 2; note that edge types are exclusive, i.e., edges of type 1 are not present in this case) and train the decoder in isolation in the otherwise same setting as the NRI (learned) model.
- **NRI (true graph)**: Here, we train the decoder in isolation and provide the ground truth interaction graph as latent graph representation.

6.B.2 Motion Capture

Our extracted motion capture dataset has a total size of 8,063 frames for 31 tracked points each. We normalize all features (position/velocity) to maximum absolute value of 1. Training and validation set samples are 49 frames long (non-overlapping segments extracted from the respective trials). Test set samples are 99 frames long. We report results on the last 50 frames of this test set data.

We choose the same hyperparameter settings as in the physical simulation experiments, with the exception that we train models for 200 epochs and with a batch size of 8. Our model here uses an MLP encoder and an RNN decoder (as the dynamics are not Markovian). We further take samples from the discrete distribution during the forward pass in training and calculate gradients via the concrete relaxation. The baselines are identical to before (path prediction experiments for physical simulations) with the following exception: for LSTM (joint) we choose a smaller hidden layer size of 128 units and train with a batch size of 1, as the model did otherwise not fit in GPU memory.

7

COMPOSITIONAL IMITATION LEARNING AND EXECUTION

In the previous chapter, we have explored the problem of discovering *relational* structure in sequential data at the example of interacting (physical) systems. Structure in dynamical systems or in sequential data more generally can also come in the form of *events* or temporal segments of special interest, which we will be the main focus of this chapter[1].

Discovering compositional structure in sequential data, without supervision, is an important ability in human and machine learning. For example, when a cook prepares a meal, they re-use similar behavioral sub-sequences (e.g., slicing, dicing, chopping) and compose the components hierarchically (e.g., stirring together eggs and milk, pouring the mixture into a hot pan and stirring it to form scrambled eggs). Humans are adept at inferring event structure by hierarchically segmenting continuous sensory experience (Zacks et al., 2001; Baldassano et al., 2017; Radvansky and Zacks, 2017), which may support building efficient event representations in episodic memory (Ezzyat and Davachi, 2011) and constructing abstract plans (Richmond and Zacks, 2017).

An important benefit of compositional sub-sequence representations is combinatorial generalization to never-before-seen conjunctions (Davidson, 1984; Denil et al., 2017). Behavioral sub-components can also be used as high-level actions in hierarchical decision-making, offering improved credit assignment and efficient planning. To reap these benefits in machines, however, the event

1 This chapter is based on our ICML 2019 publication (Kipf et al., 2019). An earlier version of this paper appeared as arXiv preprint `arXiv:1812.01483`. Permission was given by the co-authors for reproduction in this thesis. The work described in this chapter was carried out during an internship at DeepMind Technologies Ltd, London, UK.

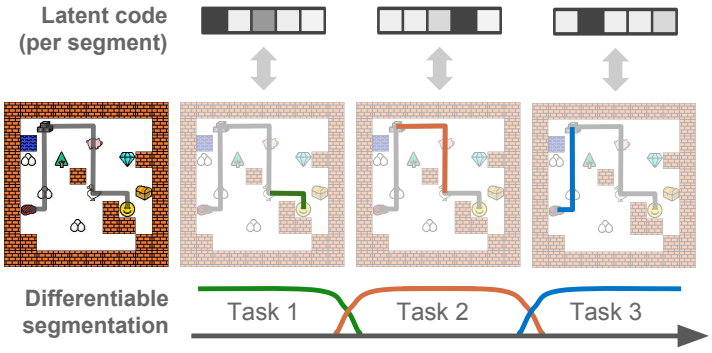


Figure 7.1: Joint unsupervised learning of task segmentation and encoding in CompILE. CompILE auto-encodes sequential demonstration data by 1) softly breaking an input sequence into segments of variable length, and 2) mapping each such segment into a latent code, which can be executed to reconstruct the input sequence. At test time, the latent code can be re-composed to produce novel behavior.

structure and composable representations must be discovered in an unsupervised manner, as sub-sequence labels are rarely available.

In this chapter and in Kipf et al. (2019), we focus on the problem of jointly learning to segment, explain, and imitate programmatic agent behavior via an unsupervised auto-encoding objective. The encoder learns to jointly infer event (or task) boundaries and high-level abstractions (latent encodings) of activity within each event segment, while the task of the decoder is to reconstruct or imitate the original behavior by executing the inferred sequence of latent codes. The latent code is structured as a sequence of latent variables that serves as a compact hierarchical abstraction of the original input sequence.

We introduce a *fully differentiable*, unsupervised segmentation model that we term CompILE (Compositional Imitation Learning and Execution) that addresses the segmentation problem by predicting soft *segment masks*. During training, the model makes multiple passes over the input sequence, explaining one segment of activity at a time. Segments explained by earlier passes are softly masked out and thereby ignored by the model. Our approach to masking is related to soft self-attention (Parikh et al., 2016; Vaswani et al., 2017), where each mask predicted by our model is localized in time (see Figure 7.1 for an example). At test time, these soft masks can be replaced with discrete, consecutive masks that mark the beginning and end of a segment. This allows us to process sequences of arbitrary length by 1) identifying the next segment, 2) explaining this segment with a latent variable, and 3) cutting/removing this

segment from the sequence and continue the process on the remainder of the input.

Formally, our model takes the form of a conditional variational auto-encoder (VAE) (Kingma and Welling, 2013; Rezende et al., 2014; Sohn et al., 2015). We introduce a method for modeling segment boundaries as softly relaxed discrete latent variables (Jang et al., 2017; Maddison et al., 2017) which allows for efficient, low-variance training.

We demonstrate the efficacy of our approach in a multi-task, instruction-following domain. Our model can reliably discover event boundaries and find effective event (sub-task) encodings. In a number of experiments, we found that CompILE possesses an inductive bias that allows it to generalize to unseen environment configurations and to task sequences which were longer than those seen during training.

7.1 METHODS

We consider the task of auto-encoding sequential data by 1) breaking an input sequence into disjoint segments of variable length, and 2) mapping each segment individually into some higher-level code, from which the input sequence can be reconstructed.

More specifically, we focus on modeling state-action trajectories of the form $\rho = ((s_1, a_1), (s_2, a_2), ..., (s_T, a_T))$ with states $s_t \in \mathcal{S}$ and actions $a_t \in \mathcal{A}$ for time steps $t = 1, ..., T$, e.g. obtained from a dataset $\mathcal{D} = \{\rho_1, \rho_2, ..., \rho_N\}$ of N expert demonstrations of variable length for a set of tasks.

7.1.1 Behavioral Cloning

Our basic setup follows that of behavioral cloning (BC), i.e., we want to find an imitation policy $\pi_{\boldsymbol{\theta}}$, parameterized by $\boldsymbol{\theta}$, by solving the following optimization problem:

$$\theta^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\rho \in \mathcal{D}} \left[p_{\boldsymbol{\theta}}(a_{1:T} | s_{1:T}) \right]. \tag{7.1}$$

In BC we have $p_{\boldsymbol{\theta}}(a_{1:T}|s_{1:T}) = \prod_{t=1:T} \pi_{\boldsymbol{\theta}}(a_t|s_t)$, where $\pi_{\boldsymbol{\theta}}(a|s)$ denotes the probability of taking action a in state s under the imitation policy $\pi_{\boldsymbol{\theta}}$.

7.1.2 Sub-Task Identification and Imitation

Differently from the default BC setup, our model breaks trajectories ρ into M disjoint segments $(c_1, c_2, ..., c_M)$:

$$c_i = ((s_{b_{i'}}, a_{b_{i'}}), (s_{b_{i'}+1}, a_{b_{i'}+1}), ..., (s_{b_i-1}, a_{b_i-1})), \tag{7.2}$$

where M is a hyperparameter, and $i' = i - 1$. Here, $b_i \in [1, T+1]$ are discrete (latent) boundary indicator variables with $b_0 = 1$, $b_M = T + 1$, and $b_i \geq b_{i'}$. We allow segments c_i to be empty if $b_i = b_{i'}$. We model each part independently with a sub-task policy $\pi_{\boldsymbol{\theta}}(a|s, z)$, where z is a latent variable summarizing the segment. Framing BC as a joint segmentation and auto-encoding problem allows us to obtain imitation policies that are specific to different inferred sub-tasks, and which can be re-combined for easier generalization to new settings. Each sub-task policy is responsible for explaining a variable-length segment of the demonstration trajectory.

We take the segment (sub-task) encoding z to be discrete in the following, but we note that other choices are possible and require only minor modifications to our framework. The probability of an action sequence $a_{1:T}$ given a sequence of states $s_{1:T}$ then takes the following form[2]:

$$\begin{aligned} p_{\boldsymbol{\theta}}(a_{1:T}|s_{1:T}) &= \sum_{b_{1:M}} \sum_{z_{1:M}} p_{\boldsymbol{\theta}}(a_{1:T}|s_{1:T}, b_{1:M}, z_{1:M}) p(b_{1:M}, z_{1:M}) \\ &= \sum_{\substack{b_{1:M} \\ z_{1:M}}} \prod_{i=1:M} p_{\boldsymbol{\theta}}(a_{b_{i'}:b_i-1}|s_{b_{i'}:b_i-1}, z_i) p(b_i|b_{i'}) p(z_i) \\ &= \sum_{\substack{b_{1:M} \\ z_{1:M}}} \prod_{i=1:M} \left[\prod_{j=b_{i'}:b_i-1} \pi_{\boldsymbol{\theta}}(a_j|s_j, z_i) \right] p(b_i|b_{i'}) p(z_i), \end{aligned} \tag{7.3}$$

where the double summation marginalizes over all allowed configurations of the discrete latent variables $z_{1:M}$ and $b_{1:M}$. We omit $p(b_0)$ since we set $b_0 = 1$. Note that our framework supports both discrete and continuous latent variables $z_{1:M}$ — for the latter case, the summation sign in Eq. 7.3 is replaced with an integral. Our (conditional) generative model $p_{\boldsymbol{\theta}}(a_{1:T}|s_{1:T}, b_{1:M}, z_{1:M})$ factorizes across time steps if we choose a non-recurrent policy $\pi_{\boldsymbol{\theta}}(a|s, z)$. Using recurrent policies is necessary, e.g., for partially observable environments and is left for future work.

For simplicity, we assume independent priors over b and z in the following form: $p(b_i, z_i|b_{1:i'}, z_{1:i'}) := p(b_i|b_{i'})p(z_i)$. We choose a uniform categorical prior

2 We again use the shorthand notation $i' = i - 1$ for clarity.

$p(z_i)$ and the following empirical categorical prior for the boundary latent variables:

$$p(b_i|b_{i'}) \propto \text{Poisson}(b_i - b_{i'}, \lambda) = e^{-\lambda}\frac{\lambda^{b_i - b_{i'}}}{(b_i - b_{i'})!}, \tag{7.4}$$

proportional to a Poisson distribution with rate λ, but truncated to the (discrete) interval $[b_{i'}, T+1]$ and renormalized, as we are dealing with sequences of finite length. This prior encourages segments to be close to λ in length and helps avoid two failure modes: 1) collapse of segments to unit length, and 2) a single segment covering the full sequence length.

7.1.3 Recognition Model

Following the standard VAE (Kingma and Welling, 2013; Rezende et al., 2014) framework, we introduce a recognition model $q_{\boldsymbol{\phi}}(b_{1:M}, z_{1:M}|a_{1:T}, s_{1:T})$ that allows us to infer a task decomposition via boundary variables $b_{1:M}$ and task encodings $z_{1:M}$ for a given trajectory ρ. We would like our recognition model to be able to generalize to new compositions of the underlying latent code. We can encourage this by dropping the dependence of $q_{\boldsymbol{\phi}}$ on any time steps before the previous boundary position. In practice, this means that once a segment (sub-task) has been identified and explained by a latent variable z, the corresponding part of the input trajectory will be masked out and the recognition model proceeds on the remainder of the trajectory, until the end is reached. This will facilitate generalization to sequences of longer length (and with more segments) than those seen during training.

Formally, we structure the recognition model as follows:

$$q_{\boldsymbol{\phi}}(b_{1:M}, z_{1:M}|a_{1:T}, s_{1:T}) = \prod_{i=1:M} q_{\boldsymbol{\phi}_z}(z_i|a_{b_{i'}:b_i-1}, s_{b_{i'}:b_i-1}) q_{\boldsymbol{\phi}_b}(b_i|a_{b_{i'}:T}, s_{b_{i'}:T}), \tag{7.5}$$

i.e., we re-use the same recognition model with shared parameters for each segment while masking out already explained segments. The core modules are the *encoding network* $q_{\boldsymbol{\phi}_z}(z|a,s)$ and the *boundary prediction network* $q_{\boldsymbol{\phi}_b}(b|a,s)$, both are modeled as categorical distributions. We use recurrent neural networks (RNN) — specifically, a uni-directional LSTM (Hochreiter and Schmidhuber, 1997) — with shared parameters, but with different output heads: one head for predicting unnormalized log-probabilities $h_{b_i,t}$ for the boundary latent variable b_i at every time step t, and one head for predicting a unnormalized log-probabilities $\mathbf{h}_{z_i}$ for the sub-task encoding z_i at the last time step in the current segment C_i.

We use multi-layer perceptrons (MLPs) to implement the output heads:

$$\mathbf{h}_{z_i} = \text{MLP}_z(\text{LSTM}_{b_i-1}(\mathbf{h}_{\text{enc},b_{i'}:b_i-1})), \tag{7.6}$$

$$h_{b_i,t} = \text{MLP}_b(\text{LSTM}_t(\mathbf{h}_{\text{enc},b_{i'}:T})), \tag{7.7}$$

where the MLPs have parameters specific to b or z (i.e., not shared between the output heads). The subscript t on LSTM_t denotes the time step at which the output is read. Note that $\mathbf{h}_{z_i}$ is a K-dimensional vector where K is the number of latent categories, whereas $h_{b_i,t}$ is a scalar specific to time step t. $\mathbf{h}_{\text{enc},t} = \text{ENC}(a_t, s_t)$ denotes a learned encoding of the input (a_t, s_t) at time step t using a neural network-based encoder ENC. In practice, we encode the state s_t (in the form of a 2D image) using a convolutional neural network (CNN) with layer normalization (Ba et al., 2016) and we learn an embedding of the discrete action a_t. Both the state encoding and the action embedding are concatenated to form $\mathbf{h}_{\text{enc},t}$. We summarize $h_{b_i,t}$ into a vector $\mathbf{h}_{b_i}$ for all time steps $t = [1, T]$. We then have $q_{\boldsymbol{\phi}_z}(z_i = k|a, s) = [\text{softmax}(\mathbf{h}_{z_i})]_k$ and $q_{\boldsymbol{\phi}_b}(b_i = t|a, s) = [\text{softmax}(\mathbf{h}_{b_i})]_t$, where $[.]_k$ denotes selection of the k-th element of a vector.

7.1.4 Continuous Relaxation

We jointly optimize for both the parameters of the sub-task policy $\pi_{\boldsymbol{\theta}}(a|s, z)$ and the recognition model $q_{\boldsymbol{\phi}}(b, z|a, s)$ by using the ELBO as an objective for learning:

$$\text{ELBO} = \mathbb{E}_{q_{\boldsymbol{\phi}}(b,z|a,s)}[\log p_{\boldsymbol{\theta}}(a|s, b, z) + \log p(b, z) - \log q_{\boldsymbol{\phi}}(b, z|a, s)], \tag{7.8}$$

where we have dropped time step and sub-task indices for ease of notation. The first term can be understood as the (negative) reconstruction error of the action sequence, given a sequence of states and inferred latent variables, whereas the last two terms, in expectation, form the (negative) Kullback-Leibler (KL) divergence between the prior $p(b, z)$ and the posterior $q_{\boldsymbol{\phi}}(b, z|a, s)$. The ELBO can be obtained from the original BC objective as follows, using Jensen's inequality:

$$\begin{aligned}\log p_{\boldsymbol{\theta}}(a|s) &= \log \sum_{b,z} p_{\boldsymbol{\theta}}(a|s, b, z)p(b, z) = \log \mathbb{E}_{q_{\boldsymbol{\phi}}(b,z|a,s)}\left[\frac{p_{\boldsymbol{\theta}}(a|s, b, z)p(b, z)}{q_{\boldsymbol{\phi}}(b, z|a, s)}\right] \\ &\geq \mathbb{E}_{q_{\boldsymbol{\phi}}(b,z|a,s)}\left[\log \frac{p_{\boldsymbol{\theta}}(a|s, b, z)p(b, z)}{q_{\boldsymbol{\phi}}(b, z|a, s)}\right] = \text{ELBO}\end{aligned} \tag{7.9}$$

To obtain low-variance gradient estimates for learning, we can use the reparameterization trick for VAEs (Kingma and Welling, 2013). Our current model formulation, however, does not allow for reparameterization as both b and z are discrete latent variables. To circumvent this issue, we make use of a (biased) continuous relaxation, i.e., we replace the respective categorical distributions with concrete (Maddison et al., 2017; Jang et al., 2017) distributions as similarly done in Chapter 6. While this is straightforward for the sub-task latent variables z, some extra consideration is required to translate the constraint $b_i \geq b_{i'}$ and the conditioning on trajectory segments of the form $s_{b_{i'}:b_i-1}$ and $a_{b_{i'}:b_i-1}$ to the continuous case. The continuous relaxation is only necessary at training time, during testing we can fall back to the discrete version explained in the previous section.

Soft Segment Masks

In the relaxed/continuous case at training time we cannot enforce a strict ordering $b_i \geq b_{i'}$ on the boundaries directly as we are now dealing with 'soft' distributions and don't have access to discrete samples at training time. It is still possible, however, to evaluate segment probabilities of the form $P(t \in C_i)$, i.e., the probability that a certain time step t in the trajectory ρ belongs to the i-th segment $C_i = [\max_{0 \leq j \leq i-1} b_j, b_i)$. The lower boundary of the segment is now given by the maximum value of all previous boundary variables, as the ordering $b_i \geq b_{i'}$ is no longer guaranteed to hold. C_i is assumed to be empty if any $b_j \geq b_i$ with $j < i$. We can evaluate segment probabilities as follows:

$$\begin{aligned} P(t \in C_i) &= P\left(\max_{0 \leq j \leq i-1} b_j \leq t < b_i\right) \\ &= \left[1 - \text{cumsum}(q_{\boldsymbol{\phi}_b}(b_i|a,s),t)\right] \prod_{j=0:i-1} \text{cumsum}(q_{\boldsymbol{\phi}_b}(b_j|a,s),t), \end{aligned} \tag{7.10}$$

where $\text{cumsum}(q_{\boldsymbol{\phi}_b}(b_j|a,s),t) = \sum_{k \leq t} q_{\boldsymbol{\phi}_b}(b_j = k|a,s)$ is a shorthand for the *inclusive* cumulative sum of the posterior $q_{\boldsymbol{\phi}_b}(b_j|a,s)$, evaluated at time step t. We further have $\text{cumsum}(q_{\boldsymbol{\phi}_b}(b_0|a,s),t) = 1$ and $\text{cumsum}(q_{\boldsymbol{\phi}_b}(b_M|a,s),t) = 0$ for $t \in [1,T]$. One can verify that $\sum_{i=1:M} P(t \in C_i) = 1$ for all $t \in [1,T]$. These segment probabilities can be seen as soft segment masks. See Figure 7.2 for an example.

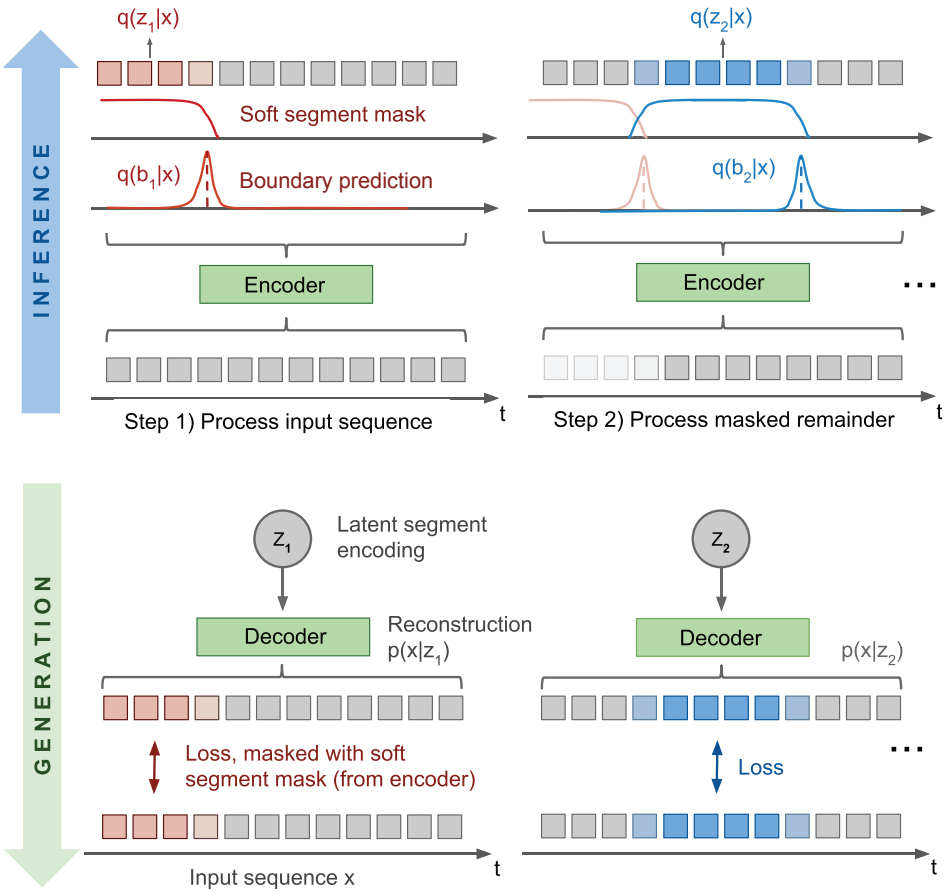


Figure 7.2: Differentiable segmentation of an input sequence x. The recognition model (encoder, marked as *inference*) predicts relaxed categorical (concrete) boundary distributions $q(b_i|x)$ from which we can obtain soft segment masks $P(t \in C_i)$. Each segment C_i is encoded via $q(z_i|x)$. The generative model $p(x|z_i)$ is executed once for every latent variable z_i. The reconstruction loss is masked with $P(t \in C_i)$, so that only the reconstructed part corresponding to the i-th segment receives a training signal. For imitation learning, the generative model (decoder, marked as *generation*) takes the form of a policy $\pi_{\theta}(a_t|s_t, z_i)$.

RNN State Masking

We softly mask out parts of the input sequence explained by earlier segments. Using a soft masking mechanism allows us to find suitable segment boundaries via backpropagation, without the need to perform explicit and potentially expensive/intractable marginalization over latent variables. Specifically, we mask out the *hidden states*[3] of the encoder RNNs. Thus, inputs belonging to earlier segments are effectively hidden from the model while still allowing gradients to be passed through. The hidden state mask for the i-th segment takes the following form:

$$\begin{aligned} \text{mask}_i(t) &= P\left(t \geq \max_{0 \leq j \leq i-1} b_j\right) \\ &= \prod_{j=0:i-1} P(t \geq b_j) = \prod_{j=0:i-1} \text{cumsum}(q_{\boldsymbol{\phi}_b}(b_j|a,s), t), \end{aligned} \tag{7.11}$$

where we set $\text{mask}_1 = 1$. In other words, it is given by the probability for a given time step to *not* belong to a previous segment. Masking is performed by multiplying the RNN's hidden state with mask_i (after the RNN update of the current time step). For every segment $i \in [1, M]$ we thus need to run the RNN over the full input sequence, while multiplying the hidden states with a segment-specific mask. Nonetheless, the parameters of the RNN are shared over all segments.

Soft RNN Readout

In addition to softly masking the RNN hidden states in both $q_{\boldsymbol{\phi}_b}(b_i|a,s)$ and $q_{\boldsymbol{\phi}_z}(z_i|a,s)$, we mask out illegal boundary positions by setting the respective logits to a large negative value. Specifically, we mask out the first time step (as any boundary placed on the first time step would result in an empty segment) and any time steps corresponding to padding values when training on mini-batches of sequences with different length. We allow boundaries (as they are exclusive) to be placed at time step $T+1$. Further, to obtain $q_{\boldsymbol{\phi}_z}(z_i|a,s)$ from the z-specific output head $\mathbf{h}_{z_i,t}$ — where t denotes the time step at which we are reading from the RNN — we perform the following weighted average:

$$q_{\boldsymbol{\phi}_z}(z_i|a,s) = \text{concrete}_\tau\left(\sum_{t=1:T} q_{\boldsymbol{\phi}_b}(b_i = t+1|a,s)\, \mathbf{h}_{z_i,t}\right), \tag{7.12}$$

3 Including the cell state in the LSTM architecture.

which can be understood as the 'soft' equivalent of reading the output head $\mathbf{h}_{z_i,t}$ for the last time step within the corresponding segment. concrete$_\tau$ is a concrete distribution (Jang et al., 2017; Maddison et al., 2017) with temperature τ. Note the necessary shift of the boundary distribution by 1 time step, as $q_{\boldsymbol{\phi}_b}(b_i|a,s)$ points to the first time step of the *following* segment.

Loss Masking

The reconstruction loss part of the ELBO, $\mathcal{L} = -\mathbb{E}_{q_{\boldsymbol{\phi}}(b,z|a,s)}[\log p_{\boldsymbol{\theta}}(a|s,b,z)]$, decomposes into independent loss terms for each segment, i.e., $\mathcal{L} = \sum_{i=1:M} \mathcal{L}_i$, due to the structure of our generative model, Eq. 7.3. To retain this property in the relaxed/continuous case, we softly mask out irrelevant parts of the action trajectory when evaluating the loss term for a single segment:

$$\mathcal{L}_i = -\mathbb{E}_{q_{\boldsymbol{\phi}}(b,z|a,s)}[\text{seg}_i \cdot \log p_{\boldsymbol{\theta}}(a|s,z_i)], \tag{7.13}$$

where the segment mask for time step t is given by $\text{seg}_i(t) = P(t \in C_i)$, i.e., the probability of time step t being explained by the i-th segment. The operator '$\cdot$' denotes element-wise multiplication. In practice, we use a single sample of the (reparameterized) posterior to evaluate Eq. 7.13.

Number of Segments

At training time, we need to specify the maximum number of segments M that the model is allowed to use when auto-encoding a particular sequence of length T. For efficient mini-batch training, we choose a single, fixed M for all training examples. Providing the correct number of segments can further be utilized as a form of weak supervision.

Complexity

Evaluating the model components $q_{\boldsymbol{\phi}_b}(b_i|a,s)$, $q_{\boldsymbol{\phi}_z}(z_i|a,s)$, and $p_{\boldsymbol{\theta}}(a|s,z_i)$ is $\mathcal{O}(T)$ for a single $i = 1,...,M$. The overall forward pass of the CompILE model for a single demonstration trajectory in terms of its length T and the number of segments M is therefore $\mathcal{O}(TM)$.

7.2 RELATED PRIOR WORK

Our framework is closely related to option discovery (Niekum et al., 2013; Kroemer et al., 2015; Fox et al., 2017; Hausman et al., 2017; Krishnan et al., 2017; Fox et al., 2018), with the main difference being that our inference algorithm is agnostic to what type of option (sub-task) encoding is used. Our framework allows for inference of continuous, discrete, or mixed continuous-discrete latent variables. Fox et al. (2017) introduce an EM-based inference algorithm for option discovery in settings similar to ours, however limited to discrete latent variables and to inference networks that are independent of the position of task boundaries: in their case without recurrency and only dependent on the current state/action pair. Their framework was later applied to continuous control tasks (Krishnan et al., 2017) and neural program modeling (Fox et al., 2018).

Option discovery has also been addressed in the context of inverse reinforcement learning (IRL) using generative adversarial networks (GANs) (Goodfellow et al., 2014) to find structured policies that are close to demonstration sequences (Hausman et al., 2017; Sharma et al., 2018). This approach requires being able to interact with the environment for imitation learning, whereas our model is based on BC and works on offline demonstration data.

Various solutions for supervised sequence segmentation or task decomposition exist which require varying degrees of supervision (Graves, 2012; Escorcia et al., 2016; Krishna et al., 2017; Shiarlis et al., 2018). In terms of two recent examples, Krishna et al. (2017) assume fully-annotated event boundaries and event descriptions at training time whereas TACO (Shiarlis et al., 2018) only requires *task sketches* (i.e., supervision on sub-task encodings but not on task boundaries) and solves an alignment problem to find a suitable segmentation. A related recent approach decomposes demonstration sequences into underlying programs (Sun et al., 2018b) in a fully-supervised setting, based on a seq2seq (Sutskever et al., 2014; Vinyals et al., 2015) model without explicitly modeling segmentation.

Outside of the area of *learning from demonstration*, hierarchical reinforcement learning (Sutton et al., 1999; Kulkarni et al., 2016; Bacon et al., 2017; Florensa et al., 2017; Vezhnevets et al., 2017; Riemer et al., 2018) and the options framework (Sutton et al., 1999; Kulkarni et al., 2016; Bacon et al., 2017; Riemer et al., 2018) similarly deal with learning segmentations and representations of behavior, but

in a purely generative way. Learning with task sketches (Andreas et al., 2017) and learning of transition policies (Lee et al., 2019) has also been addressed in this context.

Unsupervised segmentation and encoding of sequential data has also received considerable attention in natural language and speech processing (Blei and Moreno, 2001; Goldwater et al., 2009; Chan et al., 2017; Wang et al., 2017; Tang et al., 2018), and in the analysis of sequential activity data (Johnson et al., 2016; Dai et al., 2017). In concurrent work, Pertsch et al. (2019) introduced a differentiable model for keyframe discovery in sequence data, which is related to our setting. Sequence prediction models with adaptive step size (Neitz et al., 2018; Jayaraman et al., 2018) can provide segment boundaries as well, but do not directly learn a policy or latent encodings.

7.3 EXPERIMENTS

The goals of this experimental section are as follows: we would like to investigate whether our model is effective at both learning to find task boundaries and task encodings while being able to reconstruct and imitate unseen behavior. We further test whether our modular approach to task decomposition allows our model to generalize to longer sequences with more sub-tasks at test time.

7.3.1 Multi-Task Environment

We evaluate our model in a fully-observable 2D multi-task grid world, similar to the one introduced in Oh et al. (2017). An example instance for the environment is shown in Figure 7.3. See Appendix 7.B for additional implementation and evaluation details.

The environment is a 10x10 grid world with a single agent, impassable walls, and multiple objects scattered

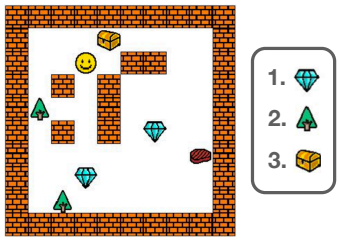


Figure 7.3: Example instance of the multi-task, instruction-following environment used in our experiments. The environment is a grid world with walls where an agent has to pick up or visit certain objects according to a list of instructions.

throughout the scene. We generate scenes with 6 objects selected uniformly at random from 10 different object types (excl. walls and player) jointly with task lists of 3-5 *visit* and *pick up* tasks. A single visit task can be solved by moving the agent to the location of an object of the correct type. For example, if the instruction is *visit tree*, the task is completed if any tree in the scene is visited. Similarly, a pick up task can be solved by picking up an object of the correct type (moving to a field adjacent to the object and executing a directional pick up action, e.g., *pick up north*). We generate a demonstration trajectory for each environment instance and task list by running a shortest path algorithm on the 2D environment grid (while marking walls as impassable).

7.3.2 Experimental Setup

In this set of experiments, we fit our CompILE model to demonstration trajectories generated for random instances of the multi-task environments (incl. randomly generated task lists). We train our model with discrete latent variables (as the target types are discrete) on demonstration trajectories with three consecutive tasks, either 3x visit instructions or 3x pick up instructions. Training is carried out on a single GPU with a fixed learning rate of 10^{-4} using the Adam (Kingma and Ba, 2014) optimizer, with a batch size of 256 and for a total of 50k training iterations. We further train a causal termination policy that shares the same architecture as the encoder of CompILE to mimic the boundary prediction module in an online setting, i.e., without *seeing the future*. See Appendix 7.A for details.

We evaluate our model on 1024 newly generated instances of the environment. We again generate demonstration trajectories with random task lists of either 3 consecutive tasks (same number as during training) or 5 consecutive tasks, to test for generalization to longer sequences, and we evaluate both boundary prediction performance and accuracy of action sequence reconstruction from the inferred latent code. We provide weak supervision by setting the number of segments to $M = 3$ and $M = 5$, respectively. We find that results slightly degrade with non-optimal choice of M.

An example implementation of the CompILE model in PyTorch (Paszke et al., 2017) for a simple sequence segmentation toy task is available under `https://github.com/tkipf/compile`.

7.3.3 Baselines

We compare against two baselines that are based on behavioral cloning (BC): an autoregressive baseline for evaluating segmentation performance, termed *LSTM surprisal*, where we find segment boundaries by thresholding the state-conditional likelihood of an action. We further compare against a VAE-based *BC baseline* that corresponds to a variant of our model without inferred task boundaries, i.e., with only a single segment. This baseline allows us to evaluate task reconstruction performance from an expert trajectory that is encoded in a *single* latent variable. We choose a 32-dim. Gaussian latent variable z (i.e., with significantly higher capacity) and a unit-variance, zero-mean Gaussian prior for this baseline. We further show results for two model variants: z- and b-CompILE, where we provide supervision on the latent variables z or b during training. z-CompILE is comparable to TACO (Shiarlis et al., 2018), where task sketches (z in our case) are provided both during training and testing (we only provide z during training), whereas b-CompILE is related to imitation learning of annotated, individual tasks.

7.3.4 Results

Results for the grid world tasks are summarized in Figure 7.4. For the pick up task, we see that our model reliably finds the correct boundary positions, i.e., it discovers the correct segments of behavior both in the 3-task setting (same as training) and in the longer 5-task setting. Reconstructions from the latent code sequence are almost perfect and only degrade slightly in the generalization setting to longer sequences, whereas the BC baseline without segmentation mechanism completely fails to generalize to longer sequences (see *exact match* score). In the visit task setting, ground truth boundary positions can be ambiguous (the agent can walk over an object unintentionally on its way somewhere else) which is reflected in the sometimes lower online evaluation score, as the termination policy can be sensitive to ambiguous termination conditions (e.g., unintentionally walked-over objects). Nonetheless, CompILE is often able to generalize to longer sequences whereas the baseline model without task segmentation consistently fails. In both tasks, our model beats a surprisal-driven segmentation baseline by a large margin.

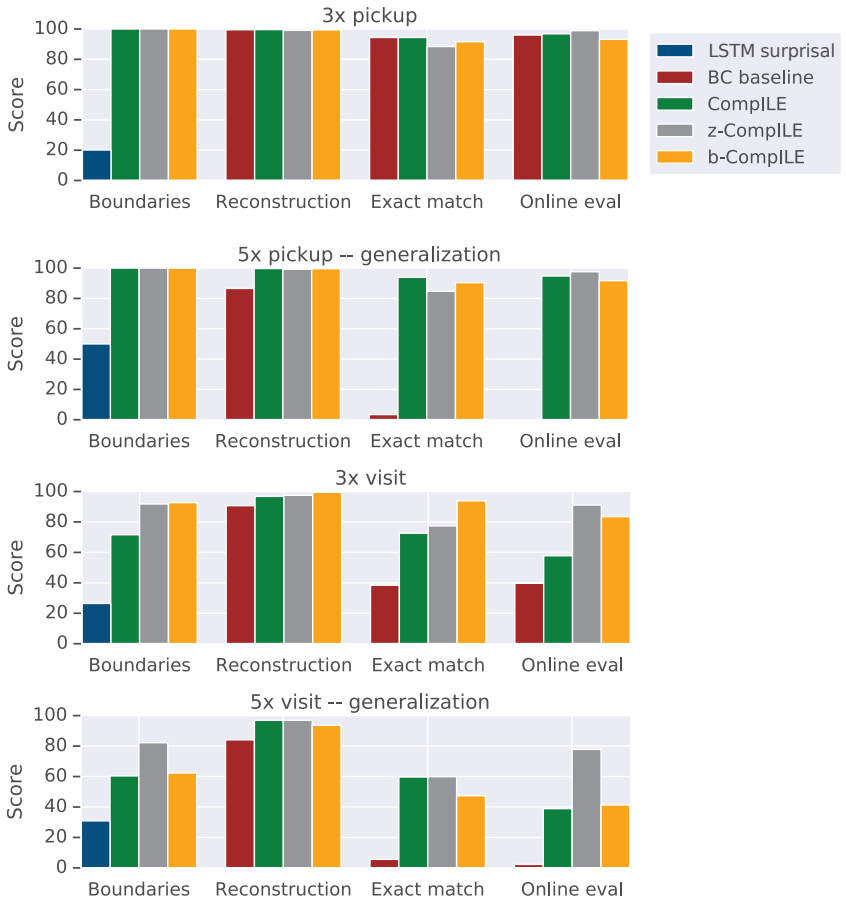


Figure 7.4: Imitation learning results in grid world domain. We report accuracy of segmentation boundary recovery, reconstruction accuracy (average over sequence vs. percentage of exact full-sequence matches) and *online evaluation*: average reward obtained when deploying the generative model (with termination policy) using the inferred latent code from the demonstration sequence in the environment, without re-training. See main text for additional details.

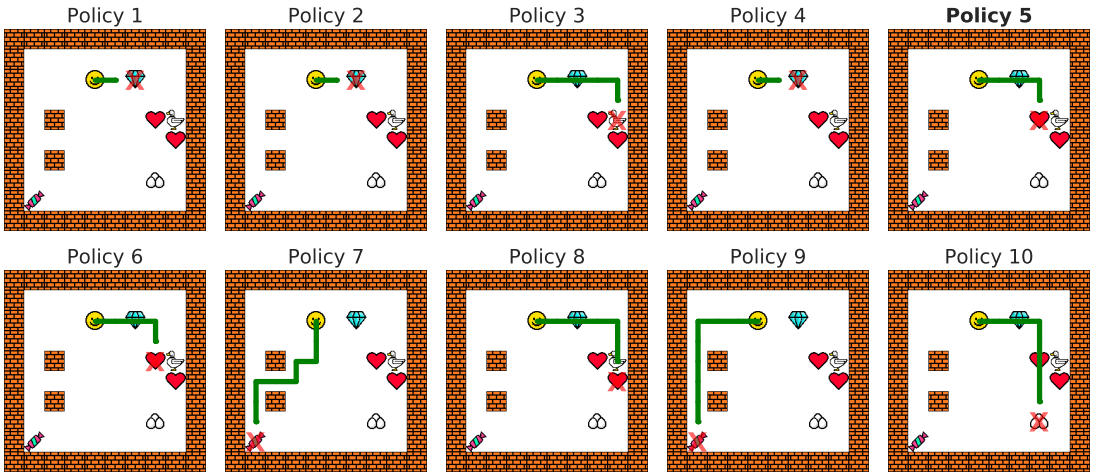


Figure 7.5: Example of sub-task policies discovered by the agent.

For qualitative analysis of the discovered sub-task policies, we run each sub-task policy for the pick up task on a random environment instance until termination, see Figure 7.5. The red cross marks the picked up object. We mark the policy in bold that the inference model of CompILE has inferred from a demonstration sequence for the task *pick up heart*. We find that the model learns latent codes that, when executed as a sub-task policy, result in the agent finding a particular object in a specific location and picking it up.

7.4 LIMITATIONS

As our training procedure is completely unsupervised, the model is free to choose any type of semantics for its latent code. For example, we found that the model can learn a location-specific latent code (with only a small degree of object specificity), whereas the ground truth task list is specific to object type. It remains to be seen to what degree the latent code can be grounded in a particular manner with only weak supervision, e.g., in a semi-supervised setting or using pairs of demonstrations with the same underlying task list. Furthermore, we have currently only explored fully-observable, Markovian settings. An extension to partially-observable environments will likely introduce further challenges, as the generative model will require some form of recurrency or memory.

7.5 CONCLUSION

In this chapter we have introduced CompILE, a model for discovering and imitating sub-components of behavior in sequential demonstration data. Our results show that CompILE can successfully discover sub-tasks and their boundaries in an imitation learning setting. While here we explored imitation learning, where inputs to the model are state-action sequences, in principle our method can be applied to any sequential data, and an interesting future direction is to apply our differentiable segmentation and auto-encoding mechanism to other data domains. Further promising directions for future work are extensions for partially-observable environments, using CompILE as an episodic memory module, and a hierarchical extension with multiple levels of abstractions for hierarchical planning.

CHAPTER APPENDIX

7.A IMPLEMENTATION DETAILS

7.A.1 Encoder

Both the recognition model and the generative model (i.e., the sub-task policies) use a two-layer CNN with 3×3 filters and 64 feature maps in each layer, followed by a ReLU activation each. We flatten the output representation into a vector and pass it through another trainable linear layer, without activation function. Only for the recognition model, we further concatenate a linear (trainable) embedding of the action ID to this representation. In all cases, we pass the output through a LayerNorm (Ba et al., 2016) layer before it is passed on to other parts of the model, e.g., the RNN in the recognition model or the sub-task policy MLP in the generative model. The LSTM state of the recognition model is reset to 0 between trajectories (and after each pass over the trajectory, i.e., for each segment).

7.A.2 Sub-Task Policies

The sub-task policies $\pi_{\boldsymbol{\theta}}(a|s,z)$ are composed of a CNN module to embed the environment state s_t and a subsequent MLP head to predict the probability of taking a particular action. This CNN shares the same architecture as the recognition model CNN. In initial experiments, we found that training separate policies $\pi_{\boldsymbol{\theta}_z}(a|s)$ for each sub-task $z \in \{1,...,K\}$ with shared CNN parameters led to better generalization performance than embedding the sub-task latent variable and providing it as input to just a single policy for all sub-tasks. For continuously relaxed latent variables z, i.e. during training, we use a soft mixture $\pi_{\boldsymbol{\theta}}(a|s,z) = \sum_{k=1:K} q_{\boldsymbol{\phi}_z}(z=k|a,s,b)\pi_{\boldsymbol{\theta}_k}(a|s)$ to obtain gradients, where we have omitted time step and segment indices to simplify notation.

7.A.3 Termination Policy

To allow for our model to be used in an online setting where the end of an event segment has to be identified before 'seeing the future', we jointly train a termination policy that shares the same model architecture (but without shared parameters) as the boundary prediction network $q_{\boldsymbol{\phi}_b}(b_i|a,s)$, but with a sigmoid$(x) = 1/(1+e^{-x})$ activation function on the log probabilities instead of a (Gumbel) softmax. It similarly passes over the input sequence M times (with softly masked out RNN hidden states) and is trained to predict an output of 1 (i.e., terminate) for the location of the i-th boundary $b_i = \text{argmax}_{t=1:T} q_{\boldsymbol{\phi}_b}(b_i = t|a,s)$ and zero otherwise. At test time, we use a threshold of 0.5 to determine termination.

7.A.4 Regularization

We use a scale hyperparameter $\beta \in [0,1]$ to scale the contribution of the KL term in Eq. 7.8 similar to the β-VAE framework (Higgins et al., 2017), which gives us control over the strength of the prior $p(b,z)$. As is common in applications of relaxed categorical posteriors in a VAE (Jang et al., 2017), we choose a simple (non-relaxed) categorical KL term for both the posterior distributions $q_{\boldsymbol{\phi}_b}(b_i|a,s)$ and $q_{\boldsymbol{\phi}_z}(z_i|a,s)$.

Further, as we do not know the precise location of the boundary latent variables b_i at training time, we cannot evaluate $p(b_i|b_{i-1})$ for $i > 1$ in the relaxed/continuous case. Under the assumption of independence between segments, behavior within each segment originating from the same distribution, and with a shared recognition model for all latents, see Eq. 7.5, we can equivalently evaluate the KL term related to b for the first boundary only, i.e. for $p(b_1)$, and multiply this term by M, where M is the number of segments (we use this setting in our experiments). Alternatively, one could place a prior on $\sum_{t=1:T} P(t \in C_i)$, which can be understood as a continuous relaxation of the length of a segment. This would allow for an individual KL contribution for every segment, which could be useful for other applications or environments, where our assumptions are too restrictive.

7.A.5 Hyperparameters

Number of Hidden Units and MLP Layers

We use 256 hidden units in all MLP layers and in the LSTM throughout all experiments, unless otherwise mentioned. A smaller number of hidden units mostly did not affect the boundary prediction accuracy, but slightly reduced performance in terms of reconstruction accuracy. For the output heads for $\mathbf{h}_{z_i,t}$, we use a single, trainable linear layer (we experimented with deeper MLPs but didn't find a difference in performance) and we use a single hidden layer MLP with ReLU activation function for the output head $h_{b_i,t}$ (the output is a scalar for every time step). The policy MLP is using a single hidden layer with ReLU activation. The termination policy uses an MLP with two hidden layers with ReLU activation functions on top of the RNN outputs.

Number of Segments

The hyperparameter M, i.e., the number of segments that the model is allowed to use to explain a particular input sequence, can have an impact on reconstruction and segmentation quality. We generally find that we obtain best results by providing the model with the true number of underlying segments (if this number is known). When providing the model with more than necessary segments, it often learns to place unneeded segmentation boundary indicators at the end of the sequence, while in some cases the model over-segments the trajectory (i.e., it breaks a single segment into parts).

Poisson Prior Rate

We fix the Poisson rate to $\lambda = 3$ in all experiments. We found that our model was not very sensitive to the precise value of λ.

Softmax Temperature

We experimented with annealing the Gumbel softmax temperature over the course of training, starting from a temperature of 1 and found that it could slightly improve results, depending on the precise choice of annealing schedule and final temperature. To simplify the exposition and to allow for easier reproduction, however, we report results with fixed temperature of 1 throughout training.

7.B EXPERIMENT DETAILS

7.B.1 Grid World Environment

The grid world environment is implemented in pycolab[4] with 8 different primitive actions: move north, move east, move south, move west, pick up north, pick up east, pick up south, pick up west. Each executed action corresponds to one time step in the environment. Observations s_t are tensors of shape $10 \times 10 \times N_{\text{things}}$, where N_{things} is the total number of things available in the environment, in our case these are 10 object types that can be interacted with, impassable walls and the player, i.e. $N_{\text{things}} = 12$. We ensure that the task is solvable and no walls make objects unreachable. Walls are placed using a recursive backtracking algorithm for unbiased maze generation. We further subsample walls using a sampling rate of 0.2 to simplify the task. The 2D grid is enclosed by a single row/column of walls that are not subsampled. Demonstration sequences are generated using a breadth-first search on the graph defined by all allowed movement transitions to find the shortest path to the goal object (ties are broken in a consistent manner). For pick up instructions, we replace the last move action in the demonstration sequence with a directional pick up action. We cut demonstration sequences to a maximum length of 42 at training time, and 200 at test time (as some of our tests involve more tasks).

7.B.2 Evalutation Metrics

In the following, we describe the evaluation metrics used in our experiments.

Boundaries

We measure the accuracy of predicted boundary position. For each boundary latent variable b_i, we check if it exactly matches the ground truth task boundary, i.e., the point where a task ends and a new task begins. Let b_i denote the ground truth position for the i-th boundary, then the accuracy is defined as

$$\frac{1}{M-1} \sum_{i=1}^{M-1} \mathbb{I}[\arg\max_{b_i} q_{\boldsymbol{\phi}_b}(b_i|x) = b_i], \tag{7.14}$$

4 https://github.com/deepmind/pycolab

where $\mathbb{I}[x = y]$ denotes the Iverson bracket that returns 1 if $x = y$ and 0 otherwise.

Reconstruction

This measures the average reconstruction accuracy of the original action sequence, given the ground truth state sequence, i.e., in a setting similar to teacher forcing:

$$\frac{1}{T} \sum_{i=1:M} \left(\sum_{j=b_{i'}:b_i-1} \mathbb{I}[\arg\max_{a_j} \pi_{\boldsymbol{\theta}}(a_j|s_j, z_i) = a_j] \right), \tag{7.15}$$

where $i' = i - 1$ and $b_i = \arg\max_{b_i} q_{\boldsymbol{\phi}_b}(b_i|x)$.

Exact Match

Here, we measure the percentage of *exact matches* of full reconstructed action sequences (i.e., this score is 1 if all actions match for a single demonstration sequence and 0 otherwise), given the ground truth state sequence (provided one step at a time) as input.

Online Eval

Here, we first run our recognition model on a demonstration trajectory to obtain a sequence of latent codes. Then, we run the sub-task policy corresponding to the first latent code in the environment, until the termination policy predicts termination, in which case we move on to the next latent code, run the respective sub-task policy, and so on. We terminate if the episode ends (more than 200 steps, wrong object picked up or all tasks completed) and measure the obtained reward (either 0 or 1). For the baseline model, we infer a single latent code and run the respective policy until the end of the episode (without termination policy). We report the average reward obtained (multiplied by a factor of 100).

7.B.3 Segmentation Baseline

To compare segmentation performance, we implemented a baseline algorithm based on auto-regressive behavioral cloning, termed *LSTM surprisal*. Given the

state-action sequence $((s_1, a_1), (s_2, a_2), \ldots, (s_T, a_T))$, this model maximizes the likelihood in the following form:

$$\max_{\boldsymbol{\theta}} P_{\boldsymbol{\theta}}(a_{1:T}|s_{1:T}) = \prod_{i=1}^{T} P(a_i|a_{1:i-1}, s_{1:i}) \tag{7.16}$$

Then, a natural approach to decide the segment boundary is based on the probability of each action. An action which is surprising (i.e., having low conditional probability) to the model should be an action that marks the beginning or end of a task segment.

Given the number of chunks M, we find the top $M-1$ boundary indicator variables $b_1, b_2, \ldots, b_{M-1}$ with minimum conditional likelihood, i.e.,

$$\underset{[b_1, b_2, \ldots, b_{M-1}], b_i \leq b_{i+1}}{\arg\min} \sum_{i=1}^{M-1} P(a_{b_i}|a_{1:b_i-1}, s_{1:b_i}) \tag{7.17}$$

In the experiments, we use the same CNN architecture for encoding the state as in CompILE. An LSTM with same embedding size as our CompILE model is used here to model the dependency on the history of states and actions. We use the same training procedure as in the other models, i.e., we only train on 3x tasks, but report performance both on 5x. Interestingly, this model finds boundaries more consistently in the generalization setting (5 tasks) for the pick up task than in the setting it was trained on (3 tasks). We hypothesize that this is due to the fact that it has never seen a 4-th and 5-th object being picked up during training, and therefore assigns low probability to these events, which corresponds to a large 'surprise' when these are observed in the generalization setting.

8

CONTRASTIVE LEARNING OF STRUCTURED WORLD MODELS

8.1 INTRODUCTION

How can we build models that reason about sensory inputs in terms of objects, abstractions, and relations, without being explicitly supervised or instructed to do so (e.g., in the form of object-annotated data)? This question aims at the core of what it means to learn a *structured* model of the world in an unsupervised way, e.g., by merely interacting with the world, and it will serve as a motivation for the types of models that we consider in this chapter[1].

Compositional reasoning in terms of objects, relations, and actions is a core ability in human cognition (Spelke and Kinzler, 2007). This ability serves as a central motivation behind a range of recent works that aim at enriching machine learning models with the ability to disentangle scenes into objects, their properties, and relations between them (Chang et al., 2017; Battaglia et al., 2016; Watters et al., 2017; Steenkiste et al., 2018; Kipf et al., 2018; Sun et al., 2018a; Sun et al., 2019b; Xu et al., 2019). These structured neural models greatly facilitate predicting physical dynamics and the consequences of actions, and provide a strong inductive bias for generalization to novel environment situations, allowing models to answer counterfactual questions such as "*What would happen if I pushed this block instead of pulling it?*".

Arriving at a structured description of the world in terms of objects and relations in the first place, however, is a challenging problem. While most methods

1 This chapter is based on our ICLR 2020 publication (Kipf et al., 2020). An earlier version of this paper appeared as arXiv preprint `arXiv:1911.12247`. Permission was given by the co-authors for reproduction in this thesis.

in this area require some form of human annotation for the extraction of objects or relations, several recent works study the problem of object discovery from visual data in a completely unsupervised or self-supervised manner (Eslami et al., 2016; Greff et al., 2017; Nash et al., 2017; Steenkiste et al., 2018; Kosiorek et al., 2018; Janner et al., 2019; Xu et al., 2019; Burgess et al., 2019; Greff et al., 2019; Engelcke et al., 2019). These methods follow a *generative* approach, i.e., they learn to discover object-based representations by performing visual predictions or reconstruction and by optimizing an objective in pixel space. Placing a loss in pixel space requires carefully trading off structural constraints on latent variables vs. accuracy of pixel-based reconstruction. Typical failure modes include ignoring visually small, but relevant features for predicting the future, such as a bullet in an Atari game (Kaiser et al., 2019), or wasting model capacity on visually rich, but otherwise potentially irrelevant features, such as static backgrounds.

To avoid such failure modes, we propose to adopt a *discriminative* approach using contrastive learning, which scores real against fake experiences in the form of state-action-state triples from an experience buffer (Lin, 1992), in a similar fashion as typical graph embedding approaches score true facts in the form of entity-relation-entity triples against corrupted triples or fake facts.

In this chapter and in Kipf et al. (2020), we introduce *contrastively-trained structured world models* (C-SWMs), a class of models for learning abstract state representations from observations in an environment. C-SWMs learn a set of abstract state variables, one for each object in a particular observation. Environment transitions are modeled using a graph neural network (Scarselli et al., 2009; Li et al., 2016; Kipf and Welling, 2017; Gilmer et al., 2017; Battaglia et al., 2018) that operates on latent abstract representations.

We further introduce a novel object-level contrastive loss for unsupervised learning of object-based representations. We arrive at this formulation by adapting methods for learning translational graph embeddings (Bordes et al., 2013; Wang et al., 2014) to our use case. By establishing a connection between contrastive learning of state abstractions (François-Lavet et al., 2018; Thomas et al., 2018) and relational graph embeddings (Nickel et al., 2015), we hope to provide inspiration and guidance for future model improvements in both fields.

In a set of experiments, where we use a novel ranking-based evaluation strategy, we demonstrate that C-SWMs learn interpretable object-level state abstrac-

tions, accurately learn to predict state transitions many steps into the future, demonstrate combinatorial generalization to novel environment configurations, and learn to identify objects from scenes without supervision.

8.2 METHODS

Our goal is to learn an object-oriented abstraction of a particular observation or environment state. In addition, we would like to learn an action-conditioned transition model of the environment that takes object representations and their relations and interactions into account.

We start by introducing the general framework for contrastive learning of state abstractions and transition models *without* object factorization in Sections 8.2.1–8.2.2, and in the following describe a variant that utilizes object-factorized state representations, which we term a *structured world model*.

8.2.1 State Abstraction

We consider an *off-policy* setting, where we operate solely on a buffer of offline experience, e.g., obtained from an exploration policy. Formally, this *experience buffer* $\mathcal{B} = \{(s_t, a_t, s_{t+1})\}_{t=1}^{T}$ contains T tuples of states $s_t \in \mathcal{S}$, actions $a_t \in \mathcal{A}$, and follow-up states $s_{t+1} \in \mathcal{S}$, which are reached after taking action a_t. We do not consider rewards as part of our framework for simplicity.

Our goal is to learn abstract or *latent* representations $\mathbf{z}_t \in \mathcal{Z}$ of environment states $s_t \in \mathcal{S}$ that discard any information which is not necessary to predict the abstract representation of the follow-up state $\mathbf{z}_{t+1} \in \mathcal{Z}$ after taking action a_t. Formally, we have an *encoder* $E : \mathcal{S} \to \mathcal{Z}$ which maps observed states to abstract state representations and a *transition model* $T : \mathcal{Z} \times \mathcal{A} \to \mathcal{Z}$ operating solely on abstract state representations.

8.2.2 Contrastive Learning

Our starting point is the graph embedding method TransE (Bordes et al., 2013): TransE embeds facts from a knowledge base $\mathcal{K} = \{(e_t, r_t, o_t)\}_{t=1}^{T}$, which consists of entity-relation-entity triples (e_t, r_t, o_t), where e_t is the subject entity

(analogous to the source state s_t in our case), r_t is the relation (analogous to the action a_t in our experience buffer), and o_t is the object entity (analogous to the target state s_{t+1}).

TransE defines the energy of a triple (e_t, r_t, o_t) as $H = d(F(e_t) + G(r_t), F(o_t))$, where F (and G) are embedding functions that map discrete entities (and relations) to $\mathbb{R}^D$, where D is the dimensionality of the embedding space, and $d(\cdot, \cdot)$ denotes the squared Euclidean distance. Training is carried out with an energy-based hinge loss (LeCun et al., 2006), with negative samples obtained by replacing the entities in a fact with random entities from the knowledge base.

We can port TransE to our setting with only minor modifications. As the *effect* of an action is in general not independent of the source state, we replace $G(r_t)$ with $T(\mathbf{z}_t, a_t)$, i.e., with the transition function, conditioned on both the action and the (embedded) source state via $\mathbf{z}_t = E(s_t)$. The overall energy of a state-action-state triple then can be defined as follows: $H = d(\mathbf{z}_t + T(\mathbf{z}_t, a_t), \mathbf{z}_{t+1})$.

This additive form of the transition model provides a strong inductive bias for modeling effects of actions in the environment as translations in the abstract state space. Alternatively, one could model effects as linear transformations or rotations in the abstract state space, which motivates the use of a graph embedding method such as RESCAL (Nickel et al., 2011), CompleX (Trouillon et al., 2016), or HolE (Nickel et al., 2016).

With the aforementioned modifications, we arrive at the following energy-based hinge loss:

$$\mathcal{L} = d(\mathbf{z}_t + T(\mathbf{z}_t, a_t), \mathbf{z}_{t+1}) + \max(0, \gamma - d(\tilde{\mathbf{z}}_t, \mathbf{z}_{t+1}))\,, \tag{8.1}$$

defined for a single (s_t, a_t, s_{t+1}) with a corrupted abstract state $\tilde{\mathbf{z}}_t = E(\tilde{s}_t)$. $\tilde{s}_t$ is sampled at random from the experience buffer. The margin γ is a hyperparameter for which we found $\gamma = 1$ to be a good choice. Unlike Bordes et al. (2013), we place the hinge only on the negative term instead of on the full loss and we do not constrain the norm of the abstract states $\mathbf{z}_t$, which we found to work better in our context. The overall loss is to be understood as an expectation of the above over samples from the experience buffer $\mathcal{B}$.

8.2.3 Object-Oriented State Factorization

Our goal is to take into account the compositional nature of visual scenes, and hence we would like to learn a relational and object-oriented model of the environment that operates on a factored abstract state space $\mathcal{Z} = \mathcal{Z}_1 \times \ldots \times \mathcal{Z}_K$, where K is the number of available object slots. We further assume an object-factorized action space $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_K$. This factorization ensures that each object is independently represented and it allows for efficient sharing of model parameters across objects in the transition model. This serves as a strong inductive bias for better generalization to novel scenes and facilitates learning and object discovery. The overall C-SWM model architecture using object-factorized representations is shown in Figure 8.1.

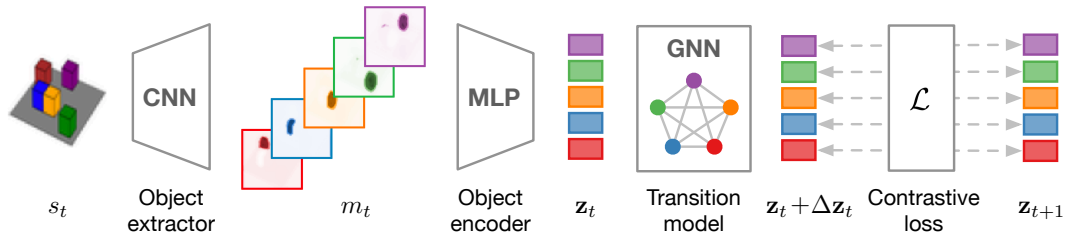


Figure 8.1: The C-SWM model is composed of the following components: 1) a CNN-based object extractor, 2) an MLP-based object encoder, 3) a GNN-based relational transition model, and 4) an object-factorized contrastive loss. Colored blocks denote abstract states for a particular object.

Encoder and Object Extractor

We split the encoder into two separate modules: 1) a CNN-based object extractor E_{ext}, and 2) an MLP-based object encoder E_{enc}. The object extractor module is a CNN operating directly on image-based observations from the environment with K feature maps in its last layer. Each feature map $m_t^k = [E_{\text{ext}}(s_t)]_k$ can be interpreted as an object mask corresponding to one particular object slot, where $[\ldots]_k$ denotes selection of the k-th feature map. For simplicity, we only assign a single feature map per object slot which sufficed for the experiments considered in this work. To allow for encoding of more complex object features (other than, e.g., position/velocity), the object extractor can be adapted to produce multiple feature maps per object slot. After the object extractor module, we flatten each feature map m_t^k (object mask) and feed it into the object encoder E_{enc}. The object encoder shares weights across objects and returns an abstract

state representation: $\mathbf{z}_t^k = E_{\text{enc}}(m_t^k)$ with $\mathbf{z}_t^k \in \mathcal{Z}_k$. We set $\mathcal{Z}_k = \mathbb{R}^D$ in the following, where D is a hyperparameter.

Relational Transition Model

We implement the transition model as a graph neural network (Scarselli et al., 2009; Li et al., 2016; Kipf and Welling, 2017; Battaglia et al., 2016; Gilmer et al., 2017; Battaglia et al., 2018), which allows us to model pairwise interactions between object states while being invariant to the order in which objects are represented. After the encoder stage, we have an abstract state description $\mathbf{z}_t^k \in \mathcal{Z}_k$ and an action $a_t^k \in \mathcal{A}_k$ for every object in the scene. We represent actions as one-hot vectors (or a vector of zeros if no action is applied to a particular object), but note that other choices are possible, e.g., for continuous action spaces. The transition function then takes as input the tuple of object representations $\mathbf{z}_t = (\mathbf{z}_t^1, \dots, \mathbf{z}_t^K)$ and actions $a_t = (a_t^1, \dots, a_t^K)$ at a particular time step:

$$\Delta \mathbf{z}_t = T(\mathbf{z}_t, a_t) = \text{GNN}(\{(\mathbf{z}_t^k, a_t^k)\}_{k=1}^K) \,. \tag{8.2}$$

$T(\mathbf{z}_t, a_t)$ is implemented as a graph neural network (GNN) that takes $\mathbf{z}_t^k$ as input node features. The model predicts updates $\Delta \mathbf{z}_t = (\Delta \mathbf{z}_t^1, \dots, \Delta \mathbf{z}_t^K)$. The object representations for the next time step are obtained via $\mathbf{z}_{t+1} = (\mathbf{z}_t^1 + \Delta \mathbf{z}_t^1, \dots, \mathbf{z}_t^K + \Delta \mathbf{z}_t^K)$. The GNN consists of node update functions f_{node} and edge update functions f_{edge} with shared parameters across all nodes and edges. These functions are implemented as MLPs and we choose the following form of message passing updates:

$$\mathbf{h}_t^{(i,j)} = f_{\text{edge}}([\mathbf{z}_t^i, \mathbf{z}_t^j]) \tag{8.3}$$

$$\Delta \mathbf{z}_t^j = f_{\text{node}}([\mathbf{z}_t^j, a_t^j, \textstyle\sum_{i \neq j} \mathbf{h}_t^{(i,j)}]) \,, \tag{8.4}$$

where $\mathbf{h}_t^{(i,j)}$ is an intermediate representation of the edge or interaction between nodes i and j. This corresponds to a single round of node-to-edge and edge-to-node message passing. Alternatively, one could apply multiple rounds of message passing, but we did not find this to be necessary for the experiments considered in this work. Note that this update rule corresponds to message passing on a fully-connected scene graph, which is $\mathcal{O}(K^2)$. This can be reduced to linear complexity by reducing connectivity to nearest neighbors in the abstract state space, which we leave for future work. We denote the output of the transition function for the k-th object as $\Delta \mathbf{z}_t^k = T^k(\mathbf{z}_t, a_t)$ in the following.

Multi-Object Contrastive Loss

We only need to change the energy function to take the factorization of the abstract state space into account, which yields the following energy H for positive triples and $\tilde{H}$ for negative samples:

$$H = \frac{1}{K}\sum_{k=1}^{K} d(\mathbf{z}_t^k + T^k(\mathbf{z}_t, a_t), \mathbf{z}_{t+1}^k)\,, \quad \tilde{H} = \frac{1}{K}\sum_{k=1}^{K} d(\tilde{\mathbf{z}}_t^k, \mathbf{z}_{t+1}^k)\,, \tag{8.5}$$

where $\tilde{\mathbf{z}}_t^k$ is the k-th object representation of the negative state sample $\tilde{\mathbf{z}}_t = E(\tilde{s}_t)$. The overall contrastive loss for a single state-action-state sample from the experience buffer then takes the form:

$$\mathcal{L} = H + \max(0, \gamma - \tilde{H})\,. \tag{8.6}$$

8.3 RELATED PRIOR WORK

For coverage of related work in the area of object discovery with autoencoder-based models, we refer the reader to the Introduction section. We further discuss related work on relational graph embeddings in Section 8.2.2.

Structured Models of Environments

Recent work on modeling structured environments such as interacting multi-object or multi-agent systems has made great strides in improving predictive accuracy by explicitly taking into account the structured nature of such systems (Sukhbaatar et al., 2016; Chang et al., 2017; Battaglia et al., 2016; Watters et al., 2017; Hoshen, 2017; Wang et al., 2018; Steenkiste et al., 2018; Kipf et al., 2018; Sanchez-Gonzalez et al., 2018; Xu et al., 2019). These methods generally make use of some form of graph neural network, where node update functions model the dynamics of individual objects, parts, or agents and edge update functions model their interactions and relations. Several recent works succeed in learning such structured models directly from pixels (Watters et al., 2017; Steenkiste et al., 2018; Xu et al., 2019; Watters et al., 2019), but in contrast to our work rely on pixel-based loss functions. The latest example in this line of research is the COBRA model (Watters et al., 2019), which learns an action-conditioned transition policy on object representations obtained from an unsupervised object discovery model (Burgess et al., 2019). Unlike C-SWM,

COBRA does not model interactions between object slots and relies on a pixel-based loss for training. Our object encoder, however, is more limited and utilizing an iterative object encoding process such as in MONet (Burgess et al., 2019) would be interesting for future work.

Contrastive Learning

Contrastive learning methods are widely used in the field of graph representation learning (Bordes et al., 2013; Perozzi et al., 2014; Grover and Leskovec, 2016; Bordes et al., 2013; Schlichtkrull et al., 2018; Veličković et al., 2018a), and for learning word representations (Mnih and Teh, 2012; Mikolov et al., 2013). The main idea is to construct pairs of related data examples (positive examples, e.g., connected by an edge in a graph or co-occuring words in a sentence) and pairs of unrelated or corrupted data examples (negative examples), and use a loss function that scores positive and negative pairs in a different way. Most energy-based losses (LeCun et al., 2006) are suitable for this task. Recent works (Oord et al., 2018; Hjelm et al., 2018; Hénaff et al., 2019; Sun et al., 2019a; Anand et al., 2019) connect objectives of this kind to the principle of learning representations by maximizing mutual information between data and learned representations, and successfully apply these methods to image, speech, and video data.

State Representation Learning

State representation learning in environments is often approached by models based on autoencoders (Corneil et al., 2018; Watter et al., 2015; Ha and Schmidhuber, 2018; Hafner et al., 2018; Laversanne-Finot et al., 2018) or via adversarial learning (Kurutach et al., 2018; Wang et al., 2019). Some recent methods learn state representations without requiring a decoder back into pixel space. Examples include the selectivity objective in Thomas et al. (2018), the contrastive objective in François-Lavet et al. (2018), the mutual information objective in Anand et al. (2019), the distribution matching objective in Gelada et al. (2019), or using causality-based losses and physical priors in latent space (Jonschkowski and Brock, 2015; Ehrhardt et al., 2018). Most notably, Ehrhardt et al. (2018) propose a method to learn an object detection module and a physics module jointly from raw video data without pixel-based losses. This approach, however, can only track a single object at a time and requires careful balancing of multiple loss functions.

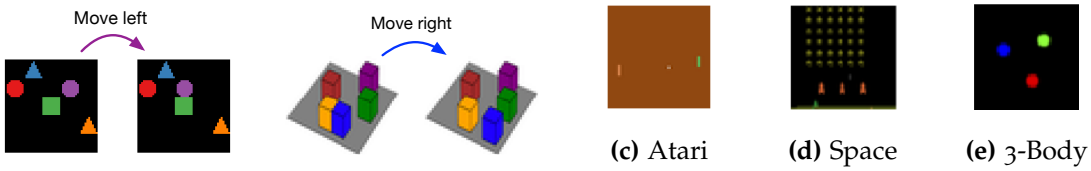
8.4 EXPERIMENTS

Our goal of this experimental section is to verify whether C-SWMs can 1) learn to discover object representations from environment interactions without supervision, 2) learn an accurate transition model in latent space, and 3) generalize to novel, unseen scenes. Our implementation uses PyTorch (Paszke et al., 2017) and is available under `https://github.com/tkipf/c-swm`.

8.4.1 Environments

We evaluate C-SWMs on two novel grid world environments (2D shapes and 3D blocks) involving multiple interacting objects that can be manipulated independently by an agent, two Atari 2600 games (Atari Pong and Space Invaders), and a multi-object physics simulation (3-body physics). See Figure 8.2 for example observations.

For all environments, we use a random policy to collect experience for both training and evaluation. Observations are provided as $50 \times 50 \times 3$ color images for the grid world environments and as $50 \times 50 \times 6$ tensors (two concatenated consecutive frames) for the Atari and 3-body physics environments. Additional details on environments and dataset creation can be found in Appendix 8.B.



(a) 2D Shapes **(b)** 3D Blocks **(c)** Atari Pong **(d)** Space Invaders **(e)** 3-Body Physics

Figure 8.2: Example observations from block pushing environments (a–b), Atari 2600 games (c–d) and a 3-body gravitational physics simulation (e). In the grid worlds (a–b), each block can be independently moved into the four cardinal directions unless the target position is occupied by another block or outside of the scene.

8.4.2 Evaluation Metrics

In order to evaluate model performance directly in latent space, we make use of ranking metrics, which are commonly used for the evaluation of link prediction

models, as in, e.g., Bordes et al. (2013). This allows us to assess the quality of learned representations directly without relying on auxiliary metrics such as pixel-based reconstruction losses, or performance in downstream tasks such as planning.

Given an observation encoded by the model and an action, we use the model to predict the representation of the next state, reached after taking the action in the environment. This predicted state representation is then compared to the encoded true observation after taking the action in the environment and a set of reference states (observations encoded by the model) obtained from the experience buffer. We measure and report both Hits at Rank 1 (H@1) and Mean Reciprocal Rank (MRR). Additional details on these evaluation metrics can be found in Appendix 8.C.

8.4.3 Baselines

Autoencoder-based World Models

The predominant method for state representation learning is based on autoencoders, and often on the VAE (Kingma and Welling, 2013; Rezende et al., 2014) model in particular. This World Model baseline is inspired by Ha and Schmidhuber (2018) and uses either a deterministic autoencoder (AE) or a VAE to learn state representations. Finally, an MLP is used to predict the next state after taking an action.

Physics As Inverse Graphics (PAIG)

This model by Jaques et al. (2019) is based on an encoder-decoder architecture and trained with pixel-based reconstruction losses, but uses a differentiable physics engine in the latent space that operates on explicit position and velocity representations for each object. Thus, this model is only applicable to the 3-body physics environment.

8.4.4 Training and Evaluation Setting

We train C-SWMs on an experience buffer obtained by running a random policy on the respective environment. We choose 1000 episodes with 100 environment steps each for the grid world environments, 1000 episodes with 10 steps

each for the Atari environments and 5000 episodes with 10 steps each for the 3-body physics environment.

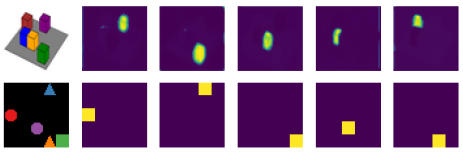
For evaluation, we populate a separate experience buffer with 10 environment steps per episode and a total of 10.000 episodes for the grid world environments, 100 episodes for the Atari environments and 1000 episodes for the physics environment. For the Atari environments, we minimize train/test overlap by 'warm-starting' experience collection in these environments with random actions before we start populating the experience buffer (see Appendix 8.B), and we ensure that not a single full test set episode coincides exactly with an episode from the training set. The state spaces of the grid world environments are large (approx. 6.4M unique states) and hence train/test coincidence of a full 10-step episode is unlikely. Overlap is similarly unlikely for the physics environment which has a continuous state space. Hence, performing well on these tasks will require some form of generalization to new environment configurations or an unseen sequence of states and actions.

All models are trained for 100 epochs (200 for Atari games) using the Adam (Kingma and Ba, 2014) optimizer with a learning rate of $5 \cdot 10^{-4}$ and a batch size of 1024 (512 for baselines with decoders due to higher memory demands, and 100 for PAIG as suggested by the authors). Model architecture details are provided in Appendix 8.A.

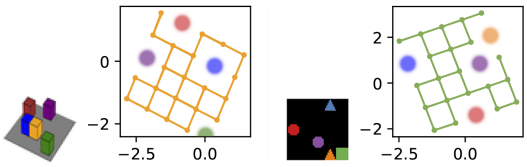
8.4.5 Qualitative Results

We present qualitative results for the grid world environments in Figure 8.3 and for the 3-body physics environment in Figure 8.4. All results are obtained on hold-out test data.

In the grid world environments, we can observe that C-SWM reliably discovers object-specific filters for a particular scene, without direct supervision. Further, each object is represented by two coordinates which correspond (up to a random linear transformation) to the true object position in the scene. Although we choose a two-dimensional latent representation per object for easier visualization, we find that results remain unchanged if we increase the dimensionality of the latent representation. The edges in this learned abstract transition graph correspond to the effect of a particular action applied to the object. The structure of the learned latent representation accurately captures the underlying grid structure of the environment. We further find that the

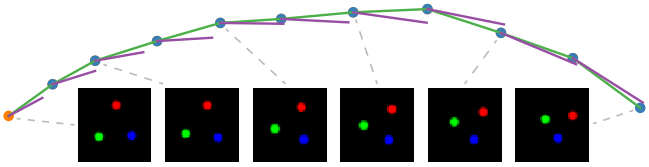


(a) Discovered object masks in a scene from the 3D cubes (top) and 2D shapes (bottom) environments.

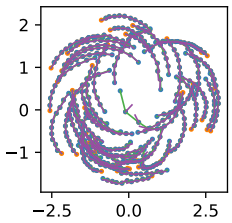


(b) Learned abstract state transition graph of the yellow cube (left) and the green square (right), while keeping all other object positions fixed at test time.

Figure 8.3: Discovered object masks (left) and direct visualization of the 2D abstract state spaces and transition graphs for a single object (right) in the block pushing environments. Nodes denote state embeddings obtained from a test set experience buffer with random actions and edges are predicted transitions. The learned abstract state graph clearly captures the underlying grid structure of the environment both in terms of object-specific latent states and in terms of predicted transitions, but is randomly rotated and/or mirrored. The model further correctly captures that certain actions do not have an effect if a neighboring position is blocked by another object (shown as colored spheres), even though the transition model does not have access to visual inputs.



(a) Observations from 3-body gravitational physics simulation (bottom) and learned abstract state transition graph for a single object slot (top).



(b) Abstract state transition graph from 50 test episodes for single object slot.

Figure 8.4: Qualitative results for 3-body physics environment for a single representative test set episode (left) and for a dataset of 50 test episodes (right). The model learns to smoothly embed object trajectories, with the circular motion represented in the latent space (projected from four to two dimensions via PCA). In the abstract state transition graph, orange nodes denote starting states for a particular episode, green links correspond to ground truth transitions and violet links correspond to transitions predicted by the model. One trajectory (in the center) strongly deviates from typical trajectories seen during training, and the model struggles to predict the correct transition.

transition model, which only has access to latent representations, correctly captures whether an action has an effect or not, e.g., if a neighboring position is blocked by another object.

Similarly, we find that the model can learn object-specific encoders in the 3-body physics environment and can learn object-specific latent representations that track location and velocity of a particular object, while learning an accurate latent transition model that generalizes well to unseen environment instances.

8.4.6 Quantitative Results

We set up quantitative experiments for evaluating the quality of both object discovery and the quality of the learned transition model. We compare against autoencoder baselines and model variants that do not represent the environment in an object-factorized manner, do not use a GNN, or do not make use of contrastive learning. Performing well under this evaluation setting requires some degree of (combinatorial) generalization to unseen environment instances.

We report ranking scores (in %) in latent space, after encoding source and target observations, and taking steps in the latent space using the learned model. Reported results are mean and standard error of scores over 4 runs on hold-out environment instances. Results are summarized in Table 8.1.

We find that baselines that make use of reconstruction losses in pixel space (incl. the C-SWM model variant without contrastive loss) typically generalize less well to unseen scenes and learn a latent space configuration that makes it difficult for the transition model to learn the correct transition function. This effect appears to be even stronger when using a VAE-based World Model, where the prior puts further constraints on the latent representations. C-SWM recovers this structure well, see Figure 8.3.

On the grid-world environments (2D shapes and 3D blocks), C-SWM models latent transitions almost perfectly, which requires taking interactions between latent representations of objects into account. Removing the interaction component, i.e., replacing the latent GNN with an object-wise MLP, makes the model insensitive to pairwise interactions and hence the ability to predict future states deteriorates. Similarly, if we remove the state factorization, the model has difficulties generalizing to unseen environment configurations.

For the Atari 2600 experiments, we find that results can have a high variance, and that the task is more difficult, as both the World Model baseline and

Table 8.1: Ranking results for multi-step prediction in latent space. Highest (mean) scores in **bold**.

		1 Step		5 Steps		10 Steps	
	Model	H@1	MRR	H@1	MRR	H@1	MRR
2D SHAPES	**C-SWM**	**100**$\pm$0.0	**100**$\pm$0.0	**100**$\pm$0.0	**100**$\pm$0.0	**99.9**$\pm$0.0	**100**$\pm$0.0
	– latent GNN	99.9$\pm$0.0	**100**$\pm$0.0	97.4$\pm$0.1	98.4$\pm$0.0	89.7$\pm$0.3	93.1$\pm$0.2
	– factored states	54.5$\pm$18.1	65.0$\pm$15.9	34.4$\pm$16.0	47.4$\pm$16.0	24.1$\pm$11.2	37.0$\pm$12.1
	– contrastive loss	49.9$\pm$0.9	55.2$\pm$0.9	6.5$\pm$0.5	9.3$\pm$0.7	1.4$\pm$0.1	2.6$\pm$0.2
	World Model (AE)	98.7$\pm$0.5	99.2$\pm$0.3	36.1$\pm$8.1	44.1$\pm$8.1	6.5$\pm$2.6	10.5$\pm$3.6
	World Model (VAE)	94.2$\pm$1.0	96.4$\pm$0.6	14.1$\pm$1.1	21.4$\pm$1.4	1.4$\pm$0.2	3.5$\pm$0.4
3D BLOCKS	**C-SWM**	**99.9**$\pm$0.0	**100**$\pm$0.0	**99.9**$\pm$0.0	**100**$\pm$0.0	**99.9**$\pm$0.0	**99.9**$\pm$0.0
	– latent GNN	**99.9**$\pm$0.0	99.9$\pm$0.0	96.3$\pm$0.4	97.7$\pm$0.3	86.0$\pm$1.8	90.2$\pm$1.5
	– factored states	74.2$\pm$9.3	82.5$\pm$8.3	48.7$\pm$12.9	62.6$\pm$13.0	65.8$\pm$14.0	49.6$\pm$11.0
	– contrastive loss	48.9$\pm$16.8	52.5$\pm$17.8	12.2$\pm$5.8	16.3$\pm$7.1	3.1$\pm$1.9	5.3$\pm$2.8
	World Model (AE)	93.5$\pm$0.8	95.6$\pm$0.6	26.7$\pm$0.7	35.6$\pm$0.8	4.0$\pm$0.2	7.6$\pm$0.3
	World Model (VAE)	90.9$\pm$0.7	94.2$\pm$0.6	31.3$\pm$2.3	41.8$\pm$2.3	7.2$\pm$0.9	12.9$\pm$1.3
ATARI PONG	**C-SWM** ($K = 5$)	20.5$\pm$3.5	41.8$\pm$2.9	9.5$\pm$2.2	22.2$\pm$3.3	5.3$\pm$1.6	15.8$\pm$2.8
	C-SWM ($K = 3$)	34.8$\pm$5.3	54.3$\pm$5.2	12.8$\pm$3.4	28.1$\pm$4.2	9.5$\pm$1.7	21.1$\pm$2.8
	C-SWM ($K = 1$)	**36.5**$\pm$5.6	**56.2**$\pm$6.2	**18.3**$\pm$1.9	**35.7**$\pm$2.3	**11.5**$\pm$1.0	**26.0**$\pm$1.2
	World Model (AE)	23.8$\pm$3.3	44.7$\pm$2.4	1.7$\pm$0.5	8.0$\pm$0.5	1.2$\pm$0.8	5.3$\pm$0.8
	World Model (VAE)	1.0$\pm$0.0	5.1$\pm$0.1	1.0$\pm$0.0	5.2$\pm$0.0	1.0$\pm$0.0	5.2$\pm$0.0
SPACE INVADERS	**C-SWM** ($K = 5$)	**48.5**$\pm$7.0	**66.1**$\pm$6.6	**16.8**$\pm$2.7	**35.7**$\pm$3.7	**11.8**$\pm$3.0	**26.0**$\pm$4.1
	C-SWM ($K = 3$)	46.2$\pm$13.0	62.3$\pm$11.5	10.8$\pm$3.7	28.5$\pm$5.8	6.0$\pm$0.4	20.9$\pm$0.9
	C-SWM ($K = 1$)	31.5$\pm$13.1	48.6$\pm$11.8	10.0$\pm$2.3	23.9$\pm$3.6	6.0$\pm$1.7	19.8$\pm$3.3
	World Model (AE)	40.2$\pm$3.6	59.6$\pm$3.5	5.2$\pm$1.1	14.1$\pm$2.0	3.8$\pm$0.8	10.4$\pm$1.3
	World Model (VAE)	1.0$\pm$0.0	5.3$\pm$0.1	0.8$\pm$0.2	5.2$\pm$0.0	1.0$\pm$0.0	5.2$\pm$0.0
3-BODY PHYSICS	**C-SWM**	**100**$\pm$0.0	**100**$\pm$0.0	97.2$\pm$0.9	98.5$\pm$0.5	**75.5**$\pm$4.7	**85.2**$\pm$3.1
	World Model (AE)	**100**$\pm$0.0	**100**$\pm$0.0	**97.7**$\pm$0.3	**98.8**$\pm$0.2	67.9$\pm$2.4	78.4$\pm$1.8
	World Model (VAE)	**100**$\pm$0.0	**100**$\pm$0.0	83.1$\pm$2.5	90.3$\pm$1.6	23.6$\pm$4.2	37.5$\pm$4.8
	Physics WM (PAIG)	89.2$\pm$3.5	90.7$\pm$3.4	57.7$\pm$12.0	63.1$\pm$11.1	25.1$\pm$13.0	33.1$\pm$13.4

C-SWM struggle to make perfect long-term predictions. While for Space Invaders, a large number of object slots ($K = 5$) appears to be beneficial, C-SWM achieves best results with only a single object slot in Atari Pong. This suggests that one should determine the optimal value of K based on performance on a validation set if it is not known a-priori. Using an iterative object encoding mechanism, such as in MONet (Burgess et al., 2019), would enable the model to assign 'empty' slots which could improve robustness w.r.t. the choice of K, which we leave for future work.

We find that both C-SWMs and the autoencoder-based World Model baseline excel at short-term predictions in the 3-body physics environment, with C-SWM having a slight edge in the 10 step prediction setting. Under our evaluation setting, the PAIG baseline (Jaques et al., 2019) underperforms using the hyperparameter setting recommended by the authors. Note that we do not tune hyperparameters of C-SWM separately for this task and use the same settings as in other environments.

We further find that the decoder of the World Model baseline usually learns to reconstruct blocks of certain color (*blue* for the 3D blocks dataset) or shape (*squares* in the 2D shapes dataset) with better quality and much earlier in the training process than other shapes or colors, likely owing to the use of a channel-wise sigmoid activation function and to the different surface areas of objects. This is an unwanted, if not harmful inductive bias: "If you want to determine how an object drops, you don't concern yourself with whether it is new or old, is red or green, or has an odor or not" (Asimov, 1988). C-SWMs, on the other hand, do not make use of a pixel-based reconstruction loss and can find suitable representations driven only by the contrastive loss in latent space.

8.4.7 Model Comparison in Pixel Space

To supplement our model comparison in latent space using ranking metrics, we carried out a direct comparison in pixel space at the example of the 2D shapes environment. This requires training a separate decoder model for C-SWM. For fairness of this comparison, we use the same protocol to train a separate decoder for the World Model (AE) baseline (discarding the one obtained from the original end-to-end auto-encoding training procedure). This decoder has

the same architecture as in the other baseline models and is trained for 100 epochs.

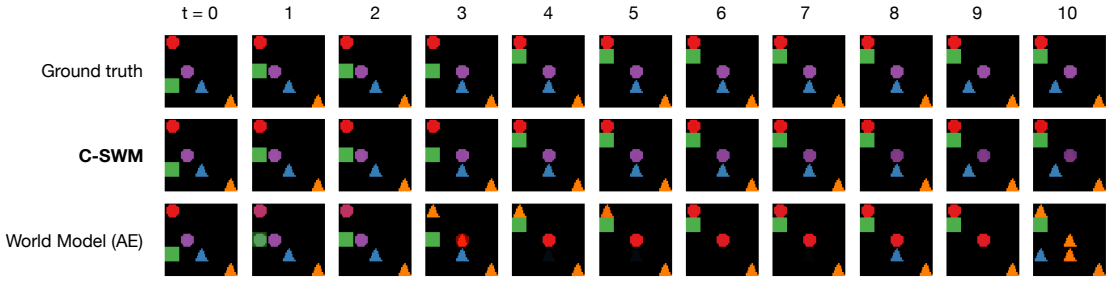


Figure 8.5: Qualitative model comparison in pixel space on a hold-out test instance of the 2D shapes environment. We train a separate decoder model for 100 epochs on both the C-SWM and the World Model baseline using all training environment instances to obtain pixel-based reconstructions for multiple prediction steps into the future.

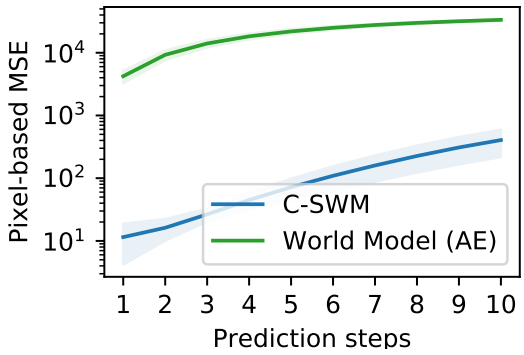


Figure 8.6: Quantitative model comparison in pixel space on a hold-out test set of the 2D shapes environment. The plot shows mean squared reconstruction error (MSE) in pixel space for multiple transition model prediction steps into the future (lower is better), averaged over 4 runs. Shaded area denotes standard error.

For a qualitative comparison, see Figure 8.5. The C-SWM model, as expected from our ranking analysis in latent space, performs almost perfectly at this task. Although the World Model (AE) baseline makes clear mistakes which compound over time, it nonetheless often gets several object positions correct after many time steps. The ranking loss in latent space captures this behaviour well, and, for example, assigns an almost perfect score for 1-step prediction to the World Model (AE) baseline. The typically used mean-squared error (MSE) in pixel space (see Figure 8.6), however, differs by *several orders of magnitude* between the two models and does not capture any of the nuanced differences in qualitative predictive behavior.

8.5 LIMITATIONS

Instance Disambiguation

In our experiments, we chose a simple feed-forward CNN architecture for the object extractor module. This type of architecture cannot disambiguate multiple instances of the same object present in one scene and relies on distinct visual features or labels (e.g., *the green square*) for object extraction. To better handle scenes which contain potentially multiple copies of the same object (e.g., in the Atari Space Invaders game), one would require some form of iterative disambiguation procedure to break symmetries and dynamically bind individual objects to slots or *object files* (Kahneman and Treisman, 1984; Kahneman et al., 1992), such as in the style of dynamic routing (Sabour et al., 2017), iterative inference (Greff et al., 2019; Engelcke et al., 2019), or sequential masking (Burgess et al., 2019; Kipf et al., 2019).

Stochasticity & Markov Assumption

Our formulation of C-SWMs does not take into account stochasticity in environment transitions or observations, and hence is limited to fully deterministic worlds. A probabilistic extension of C-SWMs is an interesting avenue for future work. For simplicity, we make the Markov assumption: state and action contain all the information necessary to predict the next state. This allows us to look at single state-action-state triples in isolation. To go beyond this limitation, one would require some form of memory mechanism, such as an RNN as part of the model architecture, which we leave for future work.

8.6 CONCLUSION

Structured world models offer compelling advantages over pure connectionist methods, by enabling stronger inductive biases for generalization, without necessarily constraining the generality of the model: for example, the contrastively trained model on the 3-body physics environment is free to store identical representations in each object slot and ignore pairwise interactions, i.e., an unstructured world model still exists as a special case. Experimentally, we find that C-SWMs make effective use of this additional structure, likely be-

cause it allows for a transition model of significantly lower complexity, and learn object-oriented models that generalize better to unseen situations.

We are excited about the prospect of using C-SWMs for model-based planning and reinforcement learning in future work, where object-oriented representations will likely allow for more accurate counterfactual reasoning about effects of actions and novel interactions in the environment. We further hope to inspire future work to think beyond autoencoder-based approaches for object-based, structured representation learning, and to address some of the limitations outlined in this chapter.

CHAPTER APPENDIX

8.A ARCHITECTURE AND HYPERPARAMETERS

Object Extractor

For the 3D cubes environment, the object extractor is a 4-layer CNN with 3×3 filters, zero-padding, and 16 feature maps per layer, with the exception of the last layer, which has $K = 5$ feature maps, i.e., one per object slot. After each layer, we apply BatchNorm (Ioffe and Szegedy, 2015) and a $\mathrm{ReLU}(x) = \max(0, x)$ activation function. For the 2D shapes environment, we choose a simpler CNN architecture with only a single convolutional layer with 10×10 filters and a stride of 10, followed by BatchNorm and a ReLU activation or LeakyReLU (Xu et al., 2015) for the Atari 2600 and physics environments. This layer has 16 feature maps and is followed by a channel-wise linear transformation (i.e., a 1×1 convolution), with 5 feature maps as output. For both models, we choose a sigmoid activation function after the last layer to obtain object masks with values in $(0, 1)$. We use the same two-layer architecture for the Atari 2600 environments and the 3-body physics environment, but with 9×9 filters (and zero-padding) in the first layer, and 5×5 filters with a stride of 5 in the second layer.

Object Encoder

After reshaping/flattening the output of the object extractor, we obtain a vector representation per object (2500-dim for the 3D cubes environment, 25-dim for the 2D shapes environment, and 1000-dim for Atari 2600 and physics environments). The object encoder is an MLP with two hidden layers of 512 units and each, followed by ReLU activation functions. We further use LayerNorm (Ba et al., 2016) before the activation function of the second hidden layer. The output of the final output layer is 2-dimensional (4-dimensional for Atari 2600 and physics environments), reflecting the ground truth object state, i.e., the object

coordinates in 2D (although this is not provided to the model), and velocity (if applicable).

Transition Model

Both the node and the edge model in the GNN-based transition model are MLPs with the same architecture / number of hidden units as the object encoder model, i.e., two hidden layers of 512 units each, LayerNorm, and ReLU activations.

Loss Function

The margin in the hinge loss is chosen as $\gamma = 1$. We further multiply the squared Euclidean distance $d(x, y)$ in the loss function with a factor of $0.5/\sigma^2$ with $\sigma = 0.5$ to control the spread of the embeddings. We use the same setting in all experiments.

8.B DATASET DETAILS

8.B.1 Grid Worlds

To generate an experience buffer for training, we initialize the environment with random object placements and uniformly sample an object and an object-specific action at every time step.

We provide state observations as $50 \times 50 \times 3$ tensors with RGB color channels, normalized to $[0, 1]$. Actions are provided as a 4-dim one-hot vector (if an action is applied) or a vector of zeros per object slot in the environment. The action one-hot vector encodes the directional movement action applied to a particular object, or is represented as a vector of zeros if no action is applied to a particular object. Note that only a single object receives an action per time step. For the Atari environments, we provide a copy of the one-hot encoded action vector to every object slot, and for the 3-body physics environment, which has no actions, we do not provide an action vector.

2D Shapes

This environment is a 5×5 grid world with 5 different objects placed at random positions. Each location can only be occupied by at maximum one object. Each object is represented by a unique shape/color combination, occupying 10×10 pixels on the overall 50×50 pixel grid. At each time step, one object can be selected and moved by one position along the four cardinal directions. See Figure 8.2a for an example. The action has no effect if the target location in a particular direction is occupied by another object or outside of the 5×5 grid. Thus, a learned transition model needs to take pairwise interactions between object properties (i.e., their locations) into account, as it would otherwise be unable to predict the effect of an action correctly.

3D Blocks

To investigate to what degree our model is robust to partial occlusion and perspective changes, we implement a simple block pushing environment using Matplotlib (Hunter, 2007) as a rendering tool. The underlying environment dynamics are the same as in the 2D Shapes dataset, and we only change the rendering component to make for a visually more challenging task that involves a different perspective and partial occlusions.

8.B.2 Atari 2600 Games

Atari Pong

We make use of the Arcade Learning Environment (Bellemare et al., 2013) to create a small environment based on the Atari 2600 game Pong which is restricted to the first interaction between the ball and the player-controlled paddle, i.e., we discard the first couple of frames from the experience buffer where the opponent behaves completely independent of any player action. Specifically, we discard the first 58 (random) environment interactions. We use the PongDeterministic-v4 variant of the environment in OpenAI Gym (Brockman et al., 2016). We use a random policy and populate an experience buffer with 10 environment interactions per episode, i.e., $T = 10$. An observation consists of two consecutive frames, cropped (to exclude the score) and resized to 50×50 pixels each.

Space Invaders

This environment is based on the Atari 2600 game Space Invaders, using the SpaceInvadersDeterministic-v4 variant of the environment in OpenAI Gym (Brockman et al., 2016), and processed / restricted in a similar manner as the Pong environment. We discard the first 50 (random) environment interactions for each episode and only begin populating the experience buffer thereafter.

8.B.3 3-Body Physics

The 3-body physics simulation environment is an interacting system that evolves according to classical gravitational dynamics. Different from the other environments considered here, there are no actions. This environment is adapted from Jaques et al. (2019) using their publicly available implementation[2], where we set the step size (dt) to 2.0 and the initial maximum x and y velocities to 0.5. We concatenate two consecutive frames of 50×50 pixels each to provide the model with (implicit) velocity information.

8.C EVALUATION METRICS

Hits at Rank k (H@k)

This score is 1 for a particular example if the predicted state representation is in the k-nearest neighbor set around the encoded true observation, where we define the neighborhood of a node to include the node itself. Otherwise this score is 0. In other words, this score measures whether the rank of the predicted state representation is smaller than or equal to k, when ranking all reference state representations by distance to the true state representation. We report the average of this score over a particular evaluation dataset.

Mean Reciprocal Rank (MRR)

This score is defined as the average inverse rank, i.e., $\text{MRR} = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{\text{rank}_n}$, where rank_n is the rank of the n-th sample.

2 https://github.com/seuqaj114/paig

8.D BASELINES

World Model Baseline

The World Model baseline is trained in two stages. First, we train an auto-encoder or a VAE with a 32-dim latent space, where the encoder is a CNN with the same architecture as the object extractor used in the C-SWM model, followed by an MLP with the same architecture as the object encoder module in C-SWM on the flattened representation of the output of the encoder CNN. The decoder exactly mirrors this architecture where we replace convolutional layers with deconvolutional layers. We verified that this architecture can successfully build representations of single frames.

Example reconstructions from the latent code are shown in Figure 8.D.1. We experimented both with mean squared error and binary cross entropy (using the continuous channel values in [0, 1] as targets) as reconstruction loss in the (V)AE models, both of which are typical choices in most practical implementations. We generally found binary cross entropy to be more stable to optimize and to produce better results, which is why we opted for this loss in all the baselines using decoders considered in the experimental section.

(a) 3D Cubes **(b)** 2D Shapes

Figure 8.D.1: Reconstructions from the latent code of a trained VAE-based World Model baseline.

In the second stage, we freeze the model parameters of the auto-encoder and train a transition model with mean-squared error on the latent representations. For the VAE model, we use the predicted mean values of the latent representations. This transition model takes the form of an MLP with the same architecture and number of hidden units as the node model in C-SWM. We experimented both with a translational transition model (i.e., the transition model only predicts the latent state difference, instead of the full next state) and direct prediction of the next state. We generally found that the translational transition model performed better and used it throughout all the reported results.

The World Model baselines are trained with a smaller batch size of 512, which slightly improved performance and simplified memory management.

Ablations

We perform the following ablations: 1) we replace the latent GNN with an MLP (per object, i.e., we remove the edge update function) to investigate whether a structured transition model is necessary, 2) we remove the state factorization and embed the full scene into a single latent variable of higher dimensionality (original dimensionality $\times$ number of original object slots) with an MLP as transition model, and 3) we replace the contrastive loss with a pixel-based reconstruction loss on both the current state and the predicted next state (we add a decoder that mirrors the architecture of the encoder).

Physics-as-Inverse-Graphics (PAIG)

For this baseline, we train the PAIG model from Jaques et al. (2019) with the code provided by the authors[3] on our dataset with the standard settings recommended by the authors for this particular task, namely: model=PhysicsNet, epochs=500, batch_size=100, base_lr=1e-3, autoencoder_loss=5.0, anneal_lr=true, color=true, and cell_type=gravity_ode_cell. We use the same input size as in C-SWM, i.e., frames are of shape $50 \times 50 \times 3$. We further set input_steps=2, pred_steps=10 and extrap_steps=0 to match our setting of predicting for a total of 10 frames while conditioning on a pair of 2 initial frames to obtain initial position and velocity information. We train this model with four different random seeds. For evaluation, we extract learned position representations for all frames of the test set and further run the latent physics prediction module (conditioned on the first two initial frames) to obtain model predictions. We further augment position representations with velocity information by taking the difference between two consecutive position representations and concatenating this representation with the 2-dim position representation, which we found to slightly improve results. In total, we obtain a 12-dim (4-dim $\times$ 3 objects) representation for each time step, i.e., the same as C-SWM. From this, we obtain ranking metrics. We found that one of the training runs (seed=2) collapsed all latent representations to a single point. We exclude this run in the results reported in Table 8.1, i.e., we report average and standard error over the three other runs only.

3 https://github.com/seuqaj114/paig

9

CONCLUSION

The main contribution of this thesis is the integration of highly structured representations with neural network-based models in the context of deep learning, which we apply to learning tasks with both *explicit* and *implicit* structure. Our focus lies in graph-structured or *relational* representations for which we introduce and investigate a number of *graph neural network* (GNN) models that structure representations and computations in the form of entities and pairwise relations (Chapters 3–6 and 8). We further investigate how to integrate modular or *compositional* structure in sequential data in the form of *events* into a deep learning framework (Chapter 7).

Having introduced the main body of our work in Parts 1 and 2 of this thesis, we can now attempt to provide answers to our research questions posed in Chapter 1 and point towards interesting directions for future work. The first two research questions address learning with explicitly graph-structured data, covered in Part 1 of this thesis, whereas the other questions are concerned with learning with implicit structure in Part 2 of this thesis.

Research Question 1: *Can we develop and efficiently implement deep neural network-based models for large-scale node classification tasks in graph-structured datasets?*

The *graph convolutional network* (GCN) (Kipf and Welling, 2017; Chapter 3) and its extension for relational data (Schlichtkrull and Kipf et al., 2018; Chapter 5) provide an important first step towards resolving this question. We find that our proposed GCN model significantly improves upon earlier methods for the task of semi-supervised node classification on a range of undirected graph datasets (Chapter 3). For relational graphs (i.e., directed graphs with

multiple edge types), we find that the relational GCN (Chapter 5) can achieve competitive results on entity classification tasks in knowledge bases with up to approx. 6M edges.

Later work has addressed some of the shortcomings of our original proposal: e.g., GraphSAGE (Hamilton et al., 2017a) achieves improved scalability by subsampling messages. Graph attention networks (Veličković et al., 2018b) generalize the GCN message passing function by learning data-dependent edge weights using an attention mechanism, which allows for improved predictive performance across a range of application domains. We can conclude that GNN-based approaches to node classification are a viable alternative to earlier methods based on pre-trained node embeddings (Perozzi et al., 2014; Grover and Leskovec, 2016) or graph-based regularization techniques (Zhu et al., 2003; Zhou et al., 2004; Belkin et al., 2006; Weston et al., 2012), and hence Research Question 1 can be answered in the affirmative.

Interesting open questions remain, for example, with respect to the ideal model architecture for a particular dataset or graph structure: recent work has shown that even linear propagation models combined with a simple classifier can achieve competitive performance (Wu et al., 2019) on some graph datasets, whereas more expressive model architectures show benefits in other cases (Xu et al., 2018) such as for classification of molecular structures. Studying the representational geometry of GNNs (Liu et al., 2019) is another promising avenue for future research.

Research Question 2: *Can graph neural networks be utilized for link prediction and unsupervised node representation learning?*

The (variational) *graph auto-encoder* (GAE) (Kipf and Welling, 2016; Chapter 4) has served as a first exploration of utilizing GNNs in the absence of node labels. In experiments on link prediction on undirected graphs we obtain competitive results and our method naturally allows for inclusion of node features while taking into account the structure of the graph.

The GAE model has inspired a variety of follow-up works (e.g., Ying et al., 2018; Grover et al., 2018; Davidson et al., 2018; Veličković et al., 2018a) that overcome limitations in terms of scalability (Ying et al., 2018), explore alternative choices of decoders and scoring functions (Grover et al., 2018; Veličković et al., 2018a), and alternative embedding geometries (Davidson et al., 2018). These

methods achieve encouraging results both in terms of unsupervised learning and link prediction in a variety of application domains.

We can conclude that GNN-based models for unsupervised node representation learning are a promising successor to earlier models based on matrix factorization (Tang and Liu, 2011) or direct optimization of a scoring function without an encoder (Perozzi et al., 2014; Grover and Leskovec, 2016). Hence, we can answer Research Question 2 in the affirmative.

Research Question 3: *Can deep neural networks infer hidden relations and interactions between entities, such as forces in physical systems?*

We have introduced the *neural relational inference* (NRI) model (Kipf and Fetaya et al., 2018; Chapter 6) to address this question. NRI is framed as a latent variable model over edges in a graph: nodes correspond to entities such as atoms in a physical system, cars in traffic, or sports players on the field, whereas edges model their hidden relations or interactions. Using GNNs both in the encoder and in the decoder of NRI allows the model to effectively learn about dynamical systems that consist of multiple interacting components. Our investigation on simulations of simple physical systems shows that the neural network-based NRI model can indeed infer hidden interactions between particles based on observations of trajectories only.

This marks an important first step towards addressing Research Question 3 and suggests an affirmative answer, but NRI makes several limiting assumptions which have to be overcome in future work. NRI assumes static interaction graphs, whereas many real-world examples involve dynamically changing interactions in the form of *interaction events* (e.g., basketball players passing the ball on the court). We further assume deterministic dynamics and access to individual object trajectories. Combining NRI with a module for dynamic inference of object identities from raw video data (e.g., Steenkiste et al., 2018) and with a mechanism to model stochasticity as in Sun et al. (2019c) are interesting directions for future research.

Research Question 4: *How can we improve upon neural network-based models that infer event structure and latent program descriptions in sequential data?*

The *compositional imitation learning and execution* (CompILE) model (Kipf et al., 2019; Chapter 7) addresses this problem by means of a novel differentiable sequence segmentation mechanism. CompILE is framed as an imitation learning model that uses a sequence of latent variables. Each latent variable describes a segment of varying length in the input, which allows CompILE to infer latent sub-programs and their encodings without supervision. Compared to earlier works (e.g., Fox et al., 2017; Shiarlis et al., 2018), that assume partial annotation (Shiarlis et al., 2018) or discrete sub-task encodings (Fox et al., 2017), CompILE makes fewer assumptions on the nature of the task and achieves promising results in a multi-task imitation learning setting.

We can conclude that structure discovery in sequential data is possible by means of the inductive bias provided by a specialized neural network architecture. A promising direction for future work is to explore applications of the CompILE model beyond the setting of imitation learning, e.g., for event discovery in video data.

Research Question 5: *Can deep neural networks learn to discover and build effective representations of objects, their relations, and effects of actions by interacting with an environment?*

This is a challenging yet important problem and we only make a small step towards the resolution of this question in this thesis. In Chapter 8 and in Kipf et al. (2020), we have introduced the *contrastively-trained structured world model* (C-SWM) that utilizes graph-structured representations for objects and their relations in a scene. C-SWMs discover objects from pixel-based observations without direct supervision and learn a structured model of environment dynamics conditioned on actions of an agent. Learning is performed using a contrastive loss in latent space that avoids typical failure modes of pixel-based losses (e.g., placing less relevance on small objects). This highly structured model allows for significantly improved generalization on environments that are composed of multiple interacting objects compared to earlier work on unstructured world models (Ha and Schmidhuber, 2018).

To conclude, we find that it is indeed possible to develop and train specialized neural architectures that can discover objects and model their relations and interactions upon intervention by an agent, without explicit supervision. Our findings, however, are limited to simple environments without stochastic-

ity, containing a fixed number of objects, and with Markovian dynamics. We expect that these limitations can be overcome in future work and that therefore an affirmative answer to Research Question 5 is within reach.

This thesis was motivated by the idea of using our understanding about the hierarchical and modular structure of the world around us in the design of neural network-based models and intelligent systems. This theme has taken us on a journey towards finding novel solutions to classical prediction problems on graphs, relational modeling, object discovery, and beyond, based on a variety of (graph-)structured neural architectures. Incorporating structure as prior knowledge takes inspiration from theories in cognitive science, such as *core knowledge* (Spelke and Kinzler, 2007), that posit that humans (and to some degree non-human primates) almost universally develop a number of core systems that serve as an invaluable *prior* (e.g., that of *objectness*) for understanding the structure of the world around us. Incorporating such prior knowledge in the form of architectural priors and inductive biases into neural network-based models might ultimately help us develop more *animal-like* (and even *human-like*) intelligent systems that can plan, act, and achieve goals in richly structured environments and generalize to new and unseen circumstances.

BIBLIOGRAPHY

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. (2016). "Tensorflow: A system for large-scale machine learning." In: *12th USENIX Symposium on Operating Systems Design and Implementation*.

Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Fei-Fei Li, and Silvio Savarese (2016). "Social LSTM: Human Trajectory Prediction in Crowded Spaces." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R Devon Hjelm (2019). "Unsupervised State Representation Learning in Atari." In: *arXiv preprint arXiv:1906.08226*.

Jacob Andreas, Dan Klein, and Sergey Levine (2017). "Modular multitask reinforcement learning with policy sketches." In: *International Conference on Machine Learning (ICML)*.

Isaac Asimov (1988). *Prelude to Foundation*. Doubleday Foundation.

James Atwood and Don Towsley (2016). "Diffusion-Convolutional Neural Networks." In: *Advances in Neural Information Processing Systems*.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton (2016). "Layer normalization." In: *arXiv preprint arXiv:1607.06450*.

Pierre-Luc Bacon, Jean Harb, and Doina Precup (2017). "The option-critic architecture." In: *AAAI Conference on Artificial Intelligence*.

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

Christopher Baldassano, Janice Chen, Asieh Zadbood, Jonathan W Pillow, Uri Hasson, and Kenneth A Norman (2017). "Discovering event structure in continuous narrative perception and memory." In: *Neuron* 95.3.

Junwei Bao, Nan Duan, Ming Zhou, and Tiejun Zhao (2014). "Knowledge-based question answering as machine translation." In: *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Albert-László Barabási and Réka Albert (1999). "Emergence of scaling in random networks." In: *Science* 286.5439.

Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an (2017). "Graph convolutional encoders for syntax-aware neural machine translation." In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Peter W Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu (2016). "Interaction networks for learning about objects, relations and physics." In: *Advances in Neural Information Processing Systems*.

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. (2018). "Relational inductive biases, deep learning, and graph networks." In: *arXiv preprint arXiv:1806.01261*.

Mikhail Belkin and Partha Niyogi (2003). "Laplacian eigenmaps for dimensionality reduction and data representation." In: *Neural Computation* 15.6.

Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani (2006). "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples." In: *Journal of Machine Learning Research* 7.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling (2013). "The arcade learning environment: An evaluation platform for general agents." In: *Journal of Artificial Intelligence Research*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin (2003). "A neural probabilistic language model." In: *Journal of Machine Learning Research* 3.

Rianne van den Berg, Thomas N Kipf, and Max Welling (2017). "Graph convolutional matrix completion." In: *arXiv preprint arXiv:1706.02263*.

David M Blei and Pedro J Moreno (2001). "Topic segmentation with an aspect hidden Markov model." In: *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Aleksandar Bojchevski and Stephan Günnemann (2017). "Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking." In: *arXiv preprint arXiv:1707.03815*.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko (2013). "Translating embeddings for modeling multi-relational data." In: *Advances in Neural Information Processing Systems*.

Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston (2015). "Large-scale simple question answering with memory networks." In: *arXiv preprint arXiv:1506.02075*.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba (2016). "Openai gym." In: *arXiv preprint arXiv:1606.01540*.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst (2017). "Geometric deep learning: going beyond euclidean data." In: *IEEE Signal Processing Magazine* 34.4.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun (2014). "Spectral networks and locally connected networks on graphs." In: *International Conference on Learning Representations (ICLR)*.

Christopher P Burgess, Loic Matthey, Nicholas Watters, Rishabh Kabra, Irina Higgins, Matt Botvinick, and Alexander Lerchner (2019). "MONet: Unsupervised Scene Decomposition and Representation." In: *arXiv preprint arXiv: 1901.11390*.

Shaosheng Cao, Wei Lu, and Qiongkai Xu (2016). "Deep neural networks for learning graph representations." In: *AAAI Conference on Artificial Intelligence*.

William Chan, Yu Zhang, Quoc Le, and Navdeep Jaitly (2017). "Latent sequence decompositions." In: *International Conference on Learning Representations (ICLR)*.

Kai-Wei Chang, Wen tau Yih, Bishan Yang, and Chris Meek (2014). "Typed Tensor Decomposition of Knowledge Bases for Relation Extraction." In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum (2017). "A compositional object-based approach to learning physical dynamics." In: *International Conference on Learning Representations (ICLR)*.

Hsinchun Chen, Xin Li, and Zan Huang (2005). "Link prediction approach to collaborative filtering." In: *Proceedings of the 5th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL)*. IEEE.

Jie Chen, Tengfei Ma, and Cao Xiao (2018). "FastGCN: Fast learning with graph convolutional networks via importance sampling." In: *International Conference on Learning Representations (ICLR)*.

Xi Chen, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel (2016). "Variational lossy autoencoder." In: *arXiv preprint arXiv:1611.02731*.

Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna (2019). "On the equivalence between graph isomorphism testing and function approximation with GNNs." In: *arXiv preprint arXiv:1905.12560*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). "Learning phrase representations using RNN encoder-decoder for statistical machine translation." In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Sumit Chopra, Raia Hadsell, and Yann LeCun (2005). "Learning a similarity metric discriminatively, with application to face verification." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio (2014). "Empirical evaluation of gated recurrent neural networks on sequence modeling." In: *arXiv preprint arXiv:1412.3555*.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter (2015). "Fast and accurate deep network learning by exponential linear units (elus)." In: *arXiv preprint arXiv:1511.07289*.

CMU (2003). *Carnegie-Mellon Motion Capture Database*. `http://mocap.cs.cmu.edu`.

Dane Corneil, Wulfram Gerstner, and Johanni Brea (2018). "Efficient model-based deep reinforcement learning with variational state tabulation." In: *arXiv preprint arXiv:1802.04325*.

George E Dahl, Navdeep Jaitly, and Ruslan Salakhutdinov (2014). "Multi-task neural networks for QSAR predictions." In: *arXiv preprint arXiv:1406.1231*.

Hanjun Dai, Bo Dai, Yan-Ming Zhang, Shuang Li, and Le Song (2017). "Recurrent hidden semi-Markov model." In: *International Conference on Learning Representations (ICLR)*.

Jeffrey Dalton, Laura Dietz, and James Allan (2014). "Entity query feature expansion using knowledge base links." In: *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Donald Davidson (1984). *Inquiries into Truth and Interpretation*. Clarendon Press, Oxford.

Tim R Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M Tomczak (2018). "Hyperspherical variational auto-encoders." In: *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Nicola De Cao and Thomas Kipf (2018). "MolGAN: An implicit generative model for small molecular graphs." In: *arXiv preprint arXiv:1805.11973*.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst (2016). "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering." In: *Advances in Neural Information Processing Systems*.

Misha Denil, Sergio Gómez Colmenarejo, Serkan Cabi, David Saxton, and Nando de Freitas (2017). "Programmable agents." In: *arXiv preprint arXiv:1706.06383*.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel (2018). "Convolutional 2d knowledge graph embeddings." In: *AAAI Conference on Artificial Intelligence*.

Li Dong, Furu Wei, Ming Zhou, and Ke Xu (2015). "Question Answering over Freebase with Multi-Column Convolutional Neural Networks." In: *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Yan Duan, Marcin Andrychowicz, Bradly C Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba (2017). "One-shot imitation learning." In: *Advances in Neural Information Processing Systems*.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams (2015). "Convolutional networks on graphs for learning molecular fingerprints." In: *Advances in Neural Information Processing Systems*.

Chris Dyer (2014). "Notes on noise contrastive estimation and negative sampling." In: *arXiv preprint arXiv:1410.8251*.

Sebastien Ehrhardt, Aron Monszpart, Niloy Mitra, and Andrea Vedaldi (2018). "Unsupervised intuitive physics from visual observations." In: *Asian Conference on Computer Vision*. Springer.

Venkatesan N Ekambaram, Giulia Fanti, Babak Ayazifar, and Kannan Ramchandran (2013). "Wavelet-regularized graph semi-supervised learning." In: *Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE.

Martin Engelcke, Adam R Kosiorek, Oiwi Parker Jones, and Ingmar Posner (2019). "GENESIS: Generative Scene Inference and Sampling with Object-Centric Latent Representations." In: *arXiv preprint arXiv:1907.13052*.

Paul Erdös and Alfréd Rényi (1959). "On Random Graphs I." In: *Publicationes Mathematicae Debrecen* 6.

Victor Escorcia, Fabian Caba Heilbron, Juan Carlos Niebles, and Bernard Ghanem (2016). "Daps: Deep action proposals for action understanding." In: *European Conference on Computer Vision (ECCV)*. Springer.

SM Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Koray Kavukcuoglu, and Geoffrey E Hinton (2016). "Attend, infer, repeat: Fast scene understanding with generative models." In: *Advances in Neural Information Processing Systems*.

Youssef Ezzyat and Lila Davachi (2011). "What constitutes an episode in episodic memory?" In: *Psychological Science* 22.2.

Carlos Florensa, Yan Duan, and Pieter Abbeel (2017). "Stochastic neural networks for hierarchical reinforcement learning." In: *International Conference on Learning Representations (ICLR)*.

Roy Fox, Sanjay Krishnan, Ion Stoica, and Ken Goldberg (2017). "Multi-level discovery of deep options." In: *arXiv preprint arXiv:1703.08294*.

Roy Fox, Richard Shin, Sanjay Krishnan, Ken Goldberg, Dawn Song, and Ion Stoica (2018). "Parametrized Hierarchical Procedures for Neural Programming." In: *International Conference on Learning Representations (ICLR)*.

Vincent François-Lavet, Yoshua Bengio, Doina Precup, and Joelle Pineau (2018). "Combined Reinforcement Learning via Abstract Representations." In: *arXiv preprint arXiv:1809.04506*.

Victor Garcia and Joan Bruna (2018). "Few-Shot Learning with Graph Neural Networks." In: *International Conference on Learning Representations (ICLR)*.

Alberto García-Durán, Antoine Bordes, and Nicolas Usunier (2015). "Composing relationships with translations." In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G Bellemare (2019). "DeepMDP: Learning Continuous Latent Space Models for Representation Learning." In: *arXiv preprint arXiv:1906.02736*.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl (2017). "Neural Message Passing for Quantum Chemistry." In: *International Conference on Machine Learning (ICML)*.

Xavier Glorot and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks." In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Sharon Goldwater, Thomas L Griffiths, and Mark Johnson (2009). "A Bayesian framework for word segmentation: Exploring the effects of context." In: *Cognition* 112.1.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik (2016). "Automatic chemical design using a data-driven continuous representation of molecules." In: *arXiv preprint arXiv:1610.02415*.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). "Generative Adversarial Nets." In: *Advances in Neural Information Processing Systems*.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. MIT press.

Marco Gori, Gabriele Monfardini, and Franco Scarselli (2005). "A new model for learning in graph domains." In: *International Joint Conference on Neural Networks*. IEEE.

Clive Granger (1969). "Investigating Causal Relations by Econometric Models and Cross-Spectral Methods." In: *Econometrica* 37.

Alex Graves (2012). "Supervised sequence labelling." In: *Supervised sequence labelling with recurrent neural networks*. Springer.

Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber (2017). "Neural expectation maximization." In: *Advances in Neural Information Processing Systems*.

Klaus Greff, Raphaël Lopez Kaufmann, Rishab Kabra, Nick Watters, Chris Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner (2019). "Multi-Object Representation Learning with Iterative Variational Inference." In: *arXiv preprint arXiv:1903.00450*.

Aditya Grover and Jure Leskovec (2016). "node2vec: Scalable Feature Learning for Networks." In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM.

Aditya Grover, Aaron Zweig, and Stefano Ermon (2018). "Graphite: Iterative generative modeling of graphs." In: *arXiv preprint arXiv:1803.10459*.

Michael Gutmann and Aapo Hyvärinen (2010). "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models." In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Nicholas Guttenberg, Nathaniel Virgo, Olaf Witkowski, Hidetoshi Aoki, and Ryota Kanai (2016). "Permutation-equivariant neural networks applied to dynamics prediction." In: *arXiv preprint arXiv:1612.04530*.

Kelvin Guu, John Miller, and Percy Liang (2015). "Traversing knowledge graphs in vector space." In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

David Ha and Jürgen Schmidhuber (2018). "World models." In: *arXiv preprint arXiv:1803.10122*.

Raia Hadsell, Sumit Chopra, and Yann LeCun (2006). "Dimensionality reduction by learning an invariant mapping." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson (2018). "Learning latent dynamics for planning from pixels." In: *arXiv preprint arXiv:1811.04551*.

Will Hamilton, Zhitao Ying, and Jure Leskovec (2017a). "Inductive representation learning on large graphs." In: *Advances in Neural Information Processing Systems*.

William L Hamilton, Rex Ying, and Jure Leskovec (2017b). "Representation learning on graphs: Methods and applications." In: *IEEE Data Engineering Bulletin*.

David K Hammond, Pierre Vandergheynst, and Rémi Gribonval (2011). "Wavelets on graphs via spectral graph theory." In: *Applied and Computational Harmonic Analysis* 30.2.

Karol Hausman, Yevgen Chebotar, Stefan Schaal, Gaurav Sukhatme, and Joseph J Lim (2017). "Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets." In: *Advances in Neural Information Processing Systems*.

Mikael Henaff, Joan Bruna, and Yann LeCun (2015). "Deep convolutional networks on graph-structured data." In: *arXiv preprint arXiv:1506.05163*.

Olivier J Hénaff, Ali Razavi, Carl Doersch, SM Eslami, and Aaron van den Oord (2019). "Data-efficient image recognition with contrastive predictive coding." In: *arXiv preprint arXiv:1905.09272*.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner (2017). "Beta-VAE: Learning basic visual concepts with a constrained variational framework." In: *International Conference on Learning Representations (ICLR)*.

Ben Hixon, Peter Clark, and Hannaneh Hajishirzi (2015). "Learning Knowledge Graphs for Question Answering through Conversational Dialog." In: *Proceedings of NAACL HLT*.

R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Adam Trischler, and Yoshua Bengio (2018). "Learning deep representations by mutual information estimation and maximization." In: *arXiv preprint arXiv:1808.06670*.

Sepp Hochreiter (1991). "Untersuchungen zu dynamischen neuronalen Netzen." In: *Diploma, Technische Universität München* 91.1.

Sepp Hochreiter and Jürgen Schmidhuber (1997). "Long short-term memory." In: *Neural Computation* 9.8.

Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt (1983). "Stochastic blockmodels: First steps." In: *Social Networks* 5.2.

Yedid Hoshen (2017). "Vain: Attentional multi-agent predictive modeling." In: *Advances in Neural Information Processing Systems*.

John D Hunter (2007). "Matplotlib: A 2D graphics environment." In: *Computing In Science & Engineering*.

Sergey Ioffe and Christian Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In: *arXiv preprint arXiv:1502.03167*.

Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena (2016). "Structural-RNN: Deep Learning on Spatio-Temporal Graphs." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Eric Jang, Shixiang Gu, and Ben Poole (2017). "Categorical reparameterization with Gumbel-softmax." In: *International Conference on Learning Representations (ICLR)*.

Michael Janner, Sergey Levine, William T Freeman, Joshua B Tenenbaum, Chelsea Finn, and Jiajun Wu (2019). "Reasoning about physical interactions with object-oriented prediction and planning." In: *International Conference on Learning Representations (ICLR)*.

Miguel Jaques, Michael Burke, and Timothy Hospedales (2019). "Physics-as-Inverse-Graphics: Joint Unsupervised Learning of Objects and Physics from Video." In: *arXiv preprint arXiv:1905.11169*.

Dinesh Jayaraman, Frederik Ebert, Alexei A Efros, and Sergey Levine (2018). "Time-agnostic prediction: Predicting predictable video frames." In: *International Conference on LearningRepresentations*.

Daniel D Johnson (2017). "Learning graphical state transitions." In: *International Conference on Learning Representations (ICLR)*.

Matthew Johnson, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R Datta (2016). "Composing graphical models with neural networks for structured representations and fast inference." In: *Advances in Neural Information Processing Systems*.

Rico Jonschkowski and Oliver Brock (2015). "Learning state representations with robotic priors." In: *Autonomous Robots*.

Daniel Kahneman and Anne Treisman (1984). *Changing views of Attention and Automaticity*. San Diego, CA: Academic Press, Inc.

Daniel Kahneman, Anne Treisman, and Brian J Gibbs (1992). "The reviewing of object files: Object-specific integration of information." In: *Cognitive psychology*.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. (2019). "Model-Based Reinforcement Learning for Atari." In: *arXiv preprint arXiv:1903.00374*.

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

Diederik P Kingma and Max Welling (2013). "Auto-encoding variational bayes." In: *arXiv preprint arXiv:1312.6114*.

Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel (2018). "Neural relational inference for interacting systems." In: *International Conference on Machine Learning (ICML)*.

Thomas Kipf, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Alvaro Sanchez-Gonzalez, Edward Grefenstette, Pushmeet Kohli, and Peter Battaglia (2019). "CompILE: Compositional Imitation Learning and Execution." In: *International Conference on Machine Learning (ICML)*.

Thomas Kipf, Elise van der Pol, and Max Welling (2020). "Contrastive Learning of Structured World Models." In: *International Conference on Learning Representations (ICLR)*.

Thomas N Kipf and Max Welling (2016). "Variational graph auto-encoders." In: *NeurIPS Workshop on Bayesian Deep Learning*.

Thomas N Kipf and Max Welling (2017). "Semi-supervised classification with graph convolutional networks." In: *International Conference on Learning Representations (ICLR)*.

Tamara G Kolda and Brett W Bader (2009). "Tensor decompositions and applications." In: *SIAM Review* 51.3.

Adam Kosiorek, Hyunjik Kim, Yee Whye Teh, and Ingmar Posner (2018). "Sequential attend, infer, repeat: Generative modelling of moving objects." In: *Advances in Neural Information Processing Systems*.

Alexander Kotov and ChengXiang Zhai (2012). "Tapping into knowledge base for concept feedback: leveraging conceptnet to improve search results for difficult queries." In: *Proceedings of the fifth ACM International Conference on Web Search and Data Mining*.

Ranjay Krishna, Kenji Hata, Frederic Ren, Li Fei-Fei, and Juan Carlos Niebles (2017). "Dense-Captioning Events in Videos." In: *International Conference on Computer Vision (ICCV)*.

Sanjay Krishnan, Roy Fox, Ion Stoica, and Ken Goldberg (2017). "DDCO: Discovery of Deep Continuous Options for Robot Learning from Demonstrations." In: *Conference on Robot Learning (CoRL)*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks." In: *Advances in Neural Information Processing Systems*.

Oliver Kroemer, Christian Daniel, Gerhard Neumann, Herke Van Hoof, and Jan Peters (2015). "Towards learning hierarchical skills for multi-phase manipulation tasks." In: *International Conference on Robotics and Automation (ICRA)*. IEEE.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum (2016). "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation." In: *Advances in Neural Information Processing Systems*.

Yoshiki Kuramoto (1975). "Self-entrainment of a population of coupled nonlinear oscillators." In: *International Symposium on Mathematical Problems in Theoretical Physics. (Lecture Notes in Physics, Vol. 39.)*

Thanard Kurutach, Aviv Tamar, Ge Yang, Stuart J Russell, and Pieter Abbeel (2018). "Learning plannable representations with causal infogan." In: *Advances in Neural Information Processing Systems*.

Dawid Laszuk (2017). *Python implementation of Kuramoto systems*. `http://www.laszukdawid.com/codes`.

Adrien Laversanne-Finot, Alexandre Péré, and Pierre-Yves Oudeyer (2018). "Curiosity driven exploration of learned disentangled goal spaces." In: *arXiv preprint arXiv:1807.01521*.

Hoang Minh Le, Yisong Yue, Peter Carr, and Patrick Lucey (2017). "Coordinated Multi-Agent Imitation Learning." In: *International Conference on Machine Learning (ICML)*.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11.

Yann LeCun, Sumit Chopra, Raia Hadsell, Marc'Aurelio Ranzato, and Fu Jie Huang (2006). "A tutorial on energy-based learning." In: *Predicting structured data*.

Youngwoon Lee, Shao-Hua Sun, Sriram Somasundaram, Edward Hu, and Joseph J Lim (2019). "Composing Complex Skills by Learning Transition Policies with Proximity Reward Induction." In: *International Conference on Learning Representations (ICLR)*.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel (2016). "Gated Graph Sequence Neural Networks." In: *International Conference on Learning Representations (ICLR)*.

David Liben-Nowell and Jon Kleinberg (2007). "The link-prediction problem for social networks." In: *Journal of the American Society for Information Science and Technology* 58.7.

Long-Ji Lin (1992). "Self-improving reactive agents based on reinforcement learning, planning and teaching." In: *Machine Learning* 8.3-4.

Yankai Lin, Zhiyuan Liu, Huanbo Luan, Maosong Sun, Siwei Rao, and Song Liu (2015). "Modeling relation paths for representation learning of knowledge bases." In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio (2017). "A structured self-attentive sentence embedding." In: *International Conference on Learning Representations (ICLR)*.

Scott W Linderman and Ryan P Adams (2014). "Discovering Latent Network Structure in Point Process Data." In: *International Conference on Machine Learning (ICML)*.

Scott W Linderman, Ryan P Adams, and Jonathan W Pillow (2016). "Bayesian latent structure discovery from multi-neuron recordings." In: *Advances in Neural Information Processing Systems*.

Huan Ling, Jun Gao, Amlan Kar, Wenzheng Chen, and Sanja Fidler (2019). "Fast interactive object annotation with curve-gcn." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Qi Liu, Maximilian Nickel, and Douwe Kiela (2019). "Hyperbolic graph neural networks." In: *Advances in Neural Information Processing Systems*.

Gilles Louppe, Kyunghyun Cho, Cyril Becot, and Kyle Cranmer (2019). "QCD-aware recursive neural networks for jet physics." In: *Journal of High Energy Physics* 2019.1.

David G Lowe (1999). "Object Recognition from Local Scale-Invariant Features." In: *International Conference on Computer Vision (ICCV)*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning (2015). "Effective approaches to attention-based neural machine translation." In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh (2017). "The concrete distribution: A continuous relaxation of discrete random variables." In: *International Conference on Learning Representations (ICLR)*.

Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman (2019). "On the universality of invariant networks." In: *arXiv preprint arXiv:1901.09342*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean (2013). "Distributed representations of words and phrases and their compositionality." In: *Advances in Neural Information Processing Systems*.

Tom M Mitchell (1980). *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ. New Jersey.

Andriy Mnih and Yee Whye Teh (2012). "A fast and simple algorithm for training neural probabilistic language models." In: *arXiv preprint arXiv:1206.6426*.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller (2013). "Playing atari with deep reinforcement learning." In: *arXiv preprint arXiv:1312.5602*.

Federico Monti, Davide Boscaini, and Jonathan Masci (2017). "Geometric deep learning on graphs and manifolds using mixture model CNNs." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Charlie Nash, Ali Eslami, Chris Burgess, Irina Higgins, Daniel Zoran, Theophane Weber, and Peter Battaglia (2017). "The multi-entity variational autoencoder." In: *NIPS Workshops*.

Arvind Neelakantan, Benjamin Roth, and Andrew McCallum (2015). "Compositional vector space models for knowledge base completion." In: *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Alexander Neitz, Giambattista Parascandolo, Stefan Bauer, and Bernhard Schölkopf (2018). "Adaptive skip intervals: Temporal abstraction for recurrent dynamical models." In: *Advances in Neural Information Processing Systems*.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel (2011). "A Three-Way Model for Collective Learning on Multi-Relational Data." In: *International Conference on Machine Learning (ICML)*.

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich (2015). "A review of relational machine learning for knowledge graphs." In: *Proceedings of the IEEE*.

Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio (2016). "Holographic embeddings of knowledge graphs." In: *AAAI Conference on Artificial Intelligence*.

Scott Niekum, Sachin Chitta, Andrew G Barto, Bhaskara Marthi, and Sarah Osentoski (2013). "Incremental Semantically Grounded Learning from Demonstration." In: *Robotics: Science and Systems*. Vol. 9. Berlin, Germany.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov (2016). "Learning convolutional neural networks for graphs." In: *International Conference on Machine Learning (ICML)*.

Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli (2017). "Zero-shot task generalization with multi-task deep reinforcement learning." In: *International Conference on Machine Learning (ICML)*.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals (2018). "Representation learning with contrastive predictive coding." In: *arXiv preprint arXiv:1807.03748*.

Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang (2016). "Tri-party deep network representation." In: *Network* 11.9.

Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit (2016). "A decomposable attention model for natural language inference." In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer (2017). "Automatic differentiation in PyTorch." In: *NIPS Autodiff Workshop*.

Heiko Paulheim and Johannes Fümkranz (2012). "Unsupervised generation of data mining features from linked open data." In: *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. (2011). "Scikit-learn: Machine learning in Python." In: *Journal of Machine Learning Research* 12.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena (2014). "Deepwalk: Online learning of social representations." In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM.

Karl Pertsch, Oleh Rybkin, Jingyun Yang, Kosta Derpanis, Joseph Lim, Kostas Daniilidis, and Andrew Jaegle (2019). "KeyIn: Discovering Subgoal Structure with Keyframe-based Video Prediction." In: *arXiv preprint arXiv:1904.05869*.

Gabriel A Radvansky and Jeffrey M Zacks (2017). "Event boundaries in memory and cognition." In: *Current Opinion in Behavioral Sciences* 17.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra (2014). "Stochastic backpropagation and approximate inference in deep generative models." In: *International Conference on Machine Learning (ICML)*.

Lauren L Richmond and Jeffrey M Zacks (2017). "Constructing experience: event models from perception to action." In: *Trends in Cognitive Sciences*.

Matthew Riemer, Miao Liu, and Gerald Tesauro (2018). "Learning abstract options." In: *Advances in Neural Information Processing Systems*.

Petar Ristoski and Heiko Paulheim (2016). "Rdf2vec: Rdf graph embeddings for data mining." In: *International Semantic Web Conference*. Springer.

Petar Ristoski, Gerben Klaas Dirk de Vries, and Heiko Paulheim (2016). "A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web." In: *International Semantic Web Conference*. Springer.

Frank Rosenblatt (1961). *Principles of neurodynamics. Perceptrons and the theory of brain mechanisms*. Tech. rep. Cornell Aeronautical Lab Inc Buffalo NY.

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton (2017). "Dynamic routing between capsules." In: *Advances in Neural Information Processing Systems*.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia (2018). "Graph networks as learnable physics engines for inference and control." In: *International Conference on Machine Learning (ICML)*.

Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap (2017). "A simple neural network module for relational reasoning." In: *Advances in Neural Information Processing Systems*.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini (2009). "The graph neural network model." In: *IEEE Transactions on Neural Networks*.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling (2018). "Modeling relational data with graph convolutional networks." In: *European Semantic Web Conference (ESWC)*.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad (2008). "Collective classification in network data." In: *AI magazine* 29.3.

Dominic Seyler, Mohamed Yahya, and Klaus Berberich (2015). "Generating quiz questions from knowledge graphs." In: *Proceedings of the 24th International Conference on World Wide Web*.

Arjun Sharma, Mohit Sharma, Nicholas Rhinehart, and Kris M Kitani (2018). "Directed-Info GAIL: Learning Hierarchical Policies from Unsegmented Demonstrations using Directed Information." In: *International Conference on Learning Representations (ICLR)*.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt (2011). "Weisfeiler-lehman graph kernels." In: *Journal of Machine Learning Research* 12.

Kyriacos Shiarlis, Markus Wulfmeier, Sasha Salter, Shimon Whiteson, and Ingmar Posner (2018). "TACO: Learning Task Decomposition via Temporal Alignment for Control." In: *International Conference on Machine Learning (ICML)*.

David I Shuman, Mohammadjavad Faraji, and Pierre Vandergheynst (2011). "Semi-supervised learning with spectral graph wavelets." In: *Proceedings of the International Conference on Sampling Theory and Applications (SampTA)*.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. (2016). "Mastering the game of Go with deep neural networks and tree search." In: *Nature* 529.7587.

Martin Simonovsky and Nikos Komodakis (2018). "GraphVAE: Towards generation of small graphs using variational autoencoders." In: *International Conference on Artificial Neural Networks*. Springer.

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng (2013). "Reasoning with neural tensor networks for knowledge base completion." In: *Advances in Neural Information Processing Systems*.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan (2015). "Learning structured output representation using deep conditional generative models." In: *Advances in Neural Information Processing Systems*.

Elizabeth S Spelke and Katherine D Kinzler (2007). "Core knowledge." In: *Developmental Science*.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of Machine Learning Research* 15.

Sjoerd van Steenkiste, Michael Chang, Klaus Greff, and Jürgen Schmidhuber (2018). "Relational Neural Expectation Maximization: Unsupervised Discovery of Objects and their Interactions." In: *International Conference on Learning Representations (ICLR)*.

Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus (2016). "Learning multiagent communication with backpropagation." In: *Advances in Neural Information Processing Systems*.

Chen Sun, Abhinav Shrivastava, Carl Vondrick, Kevin Murphy, Rahul Sukthankar, and Cordelia Schmid (2018a). "Actor-centric relation network." In: *European Conference on Computer Vision (ECCV)*.

Chen Sun, Fabien Baradel, Kevin Murphy, and Cordelia Schmid (2019a). "Contrastive Bidirectional Transformer for Temporal Representation Learning." In: *arXiv preprint arXiv:1906.05743*.

Chen Sun, Abhinav Shrivastava, Carl Vondrick, Rahul Sukthankar, Kevin Murphy, and Cordelia Schmid (2019b). "Relational Action Forecasting." In: *arXiv preprint arXiv:1904.04231*.

Chen Sun, Per Karlsson, Jiajun Wu, Joshua B Tenenbaum, and Kevin Murphy (2019c). "Stochastic Prediction of Multi-Agent Interactions from Partial Observations." In: *arXiv preprint arXiv:1902.09641*.

Shao-Hua Sun, Hyeonwoo Noh, Sriram Somasundaram, and Joseph Lim (2018b). "Neural program synthesis from diverse demonstration videos." In: *International Conference on Machine Learning (ICML)*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le (2014). "Sequence to sequence learning with neural networks." In: *Advances in Neural Information Processing Systems*.

Richard S Sutton, Doina Precup, and Satinder Singh (1999). "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning." In: *Artificial intelligence* 112.1-2.

Da Tang, Xiujun Li, Jianfeng Gao, Chong Wang, Lihong Li, and Tony Jebara (2018). "Subgoal Discovery for Hierarchical Dialogue Policy Learning." In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei (2015). "Line: Large-scale information network embedding." In: *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee.

Lei Tang and Huan Liu (2011). "Leveraging social media networks for classification." In: *Data Mining and Knowledge Discovery* 23.3.

Valentin Thomas, Emmanuel Bengio, William Fedus, Jules Pondard, Philippe Beaudoin, Hugo Larochelle, Joelle Pineau, Doina Precup, and Yoshua Bengio (2018). "Disentangling the independently controllable factors of variation by interacting with the world." In: *arXiv preprint arXiv:1802.09484*.

Kristina Toutanova, Victoria Lin, Wen-tau Yih, Hoifung Poon, and Chris Quirk (2016). "Compositional Learning of Embeddings for Relation Paths in Knowledge Base and Text." In: *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard (2016). "Complex embeddings for simple link prediction." In: *International Conference on Machine Learning (ICML)*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is all you need." In: *Advances in Neural Information Processing Systems*.

Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm (2018a). "Deep graph infomax." In: *arXiv preprint arXiv:1809.10341*.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio (2018b). "Graph Attention Networks." In: *International Conference on Learning Representations (ICLR)*.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu (2017). "FeUdal networks for hierarchical reinforcement learning." In: *International Conference on Machine Learning (ICML)*.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan (2015). "Show and tell: A neural image caption generator." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Gerben Klaas Dirk de Vries and Steven de Rooij (2015). "Substructure counting graph kernels for machine learning from RDF data." In: *Web Semantics: Science, Services and Agents on the World Wide Web* 35.

Angelina Wang, Thanard Kurutach, Kara Liu, Pieter Abbeel, and Aviv Tamar (2019). "Learning Robotic Manipulation through Visual Planning and Acting." In: *arXiv preprint arXiv:1905.04411*.

Chong Wang, Yining Wang, Po-Sen Huang, Abdelrahman Mohamed, Dengyong Zhou, and Li Deng (2017). "Sequence modeling via segmentations." In: *International Conference on Machine Learning (ICML)*.

Daixin Wang, Peng Cui, and Wenwu Zhu (2016). "Structural deep network embedding." In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM.

Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler (2018). "Nervenet: Learning structured policy with graph neural networks." In: *International Conference on Learning Representations (ICLR)*.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen (2014). "Knowledge graph embedding by translating on hyperplanes." In: *AAAI Conference on Artificial Intelligence*.

Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller (2015). "Embed to control: A locally linear latent dynamics model for control from raw images." In: *Advances in Neural Information Processing Systems*.

Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti (2017). "Visual interaction networks: Learning a physics simulator from video." In: *Advances in Neural Information Processing Systems*.

Nicholas Watters, Loic Matthey, Matko Bosnjak, Christopher P Burgess, and Alexander Lerchner (2019). "COBRA: Data-Efficient Model-Based RL through Unsupervised Object Discovery and Curiosity-Driven Exploration." In: *arXiv preprint arXiv:1905.09275*.

Duncan J Watts and Steven H Strogatz (1998). "Collective dynamics of 'small-world' networks." In: *Nature* 393.6684.

Paul J Werbos (1982). "Applications of advances in nonlinear sensitivity analysis." In: *System modeling and optimization*. Springer.

Paul J Werbos (1990). "Backpropagation through time: what it does and how to do it." In: *Proceedings of the IEEE* 78.10.

Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert (2012). "Deep learning via semi-supervised embedding." In: *Neural Networks: Tricks of the Trade*. Springer.

Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger (2019). "Simplifying graph convolutional networks." In: *arXiv preprint arXiv:1902.07153*.

Chenyan Xiong and Jamie Callan (2015a). "Esdrank: Connecting query and documents through external semi-structured data." In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*.

Chenyan Xiong and Jamie Callan (2015b). "Query expansion with Freebase." In: *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*.

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li (2015). "Empirical evaluation of rectified activations in convolutional network." In: *arXiv preprint arXiv:1505.00853*.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka (2018). "How powerful are graph neural networks?" In: *arXiv preprint arXiv:1810.00826*.

Zhenjia Xu, Zhijian Liu, Chen Sun, Kevin Murphy, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu (2019). "Unsupervised Discovery of Parts, Structure, and Dynamics." In: *arXiv preprint arXiv:1903.05136*.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng (2014). "Embedding entities and relations for learning and inference in knowledge bases." In: *arXiv preprint arXiv:1412.6575*.

Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Chang (2015). "Network representation learning with rich text information." In: *International Joint Conference on Artificial Intelligence (IJCAI)*.

Zhilin Yang, William Cohen, and Ruslan Salakhutdinov (2016). "Revisiting Semi-Supervised Learning with Graph Embeddings." In: *International Conference on Machine Learning (ICML)*.

Xuchen Yao and Benjamin Van Durme (2014). "Information Extraction over Structured Data: Question Answering with Freebase." In: *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec (2018). "Graph convolutional neural networks for web-scale recommender systems." In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM.

Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec (2018). "Graph convolutional policy network for goal-directed molecular graph generation." In: *Advances in Neural Information Processing Systems*.

Bing Yu, Haoteng Yin, and Zhanxing Zhu (2018). "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting." In: *International Joint Conference on Artificial Intelligence (IJCAI)*.

Jeffrey M Zacks, Barbara Tversky, and Gowri Iyer (2001). "Perceiving, remembering, and communicating structure in events." In: *Journal of Experimental Psychology: General* 130.1.

Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf (2004). "Learning with local and global consistency." In: *Advances in Neural Information Processing Systems*.

Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty (2003). "Semi-supervised learning using Gaussian fields and harmonic functions." In: *International Conference on Machine Learning (ICML)*.

SAMENVATTING - SUMMARY IN DUTCH

In dit proefschrift, *Deep Learning with Graph-Structured Representations*, introduceren we nieuwe benaderingen voor *machine learning* met gestructureerde data. Deze benaderingen zijn grotendeels gebaseerd op het idee om representaties en berekeningen van neurale netwerken te structureren in de vorm van een graaf. Dit zorgt voor een verbeterde generalisatie bij het leren van data met zowel een *expliciete* als een *impliciete* modulaire structuur.

We dragen het volgende bij:

- We introduceren ***graph convolutional networks*** (GCNs) (Kipf and Welling, 2017; Hoofdstuk 3) voor *semi-supervised* classificatie van knopen in graaf gestructureerde data. GCNs zijn een bepaald soort *graph neural network* die geparameterizeerde *message passing* operaties uitvoeren op een graaf. Deze operaties worden gemodelleerd als eerste-ordebenaderingen van *spectral graph convolutions*. Ten tijde van publicatie behaalden GCNs *state-of-the-art* resultaten op classificatietaken op knoopniveau uit een aantal ongerichte graaf datasets.

- We introduceren ***graph auto-encoders*** (GAEs) (Kipf and Welling, 2016; Hoofdstuk 4) voor *unsupervised learning* en het voorspellen van zijden in graaf gestructureerde data. GAEs hebben een *encoder* gebaseerd op *graph neural networks*, en een *decoder* die zijden in een graaf reconstrueert op basis van een paarsgewijze scorefunctie. We introduceren ook een variant van GAEs, ontworpen als een probabilistisch generatief model dat getraind wordt met *variational inference*. We noemen deze variant *variational GAE*. GAEs and *variational* GAEs zijn bijzonder geschikt voor *representation learning* op grafen als er geen labels voor de knopen beschikbaar zijn.

- We introduceren ***relational GCNs*** (Schlichtkrull and Kipf et al., 2018; Hoofdstuk 5) die het GCN model uitbreiden naar een gerichte relationele graaf met verschillende soorten zijden. *Relational* GCNs zijn geschikt om relationele data te modelleren en we laten de toepasbaarheid zien op *semi-supervised* entiteitsclassficatie in *knowledge bases*.

- We introduceren ***neural relational inference*** (NRI) (Kipf and Fetaya et al., 2018; Hoofdstuk 6) voor het ontdekken van latente relationele structuur in interacterende systemen. NRI combineert *graph neural networks* met een probabilistisch *latent variable model* over verschillende soorten zijden in een graaf. We passen NRI toe om interacterende dynamische systemen te modelleren, zoals systemen met meerdere deeltjes in de natuurkunde.

- We introduceren ***compositional imitation learning and execution*** (CompILE) (Kipf et al., 2019; Hoofdstuk 7), een model voor het ontdekken van structuur in sequentiële gedragsdata. CompILE maakt gebruik van een nieuw differentieerbaar mechanisme dat sequenties kan segmenteren. Dit mechanisme wordt gebruikt om losse gedragingen en vaardigheden te ontdekken in de context van *imitation learning*.

- We introduceren ***contrastively-trained structured world models*** (C-SWMs) (Kipf et al., 2020; Hoofdstuk 8) om zonder supervisie van rauwe pixel observaties object-gefactoriseerde modellen te leren van omgevingen. C-SWMs maken gebruik van *graph neural networks* om de representatie van een omgeving te structureren in de vorm van een graaf. In deze graaf representeren knopen objecten en zijden representeren paarsgewijze relaties of interacties onder invloed van een bepaalde actie. C-SWMs worden getraind met *constrastive learning* zonder op pixels gebaseerde kostenfuncties en zijn geschikt voor het leren van modellen van omgevingen met een compositionele structuur.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Max Welling for giving me the opportunity to pursue a PhD under his supervision. I am tremendously grateful for his continuous support, for countless inspiring discussions, for giving me the freedom to pursue a variety of research directions, and for frequently reminding me to "think big" in the past four years. Many thanks also go to Ivan Titov for co-advising me in the beginning of my PhD before his move to Edinburgh.

Being part of AMLab, the Amsterdam Machine Learning Lab, was a wonderful and truly enriching experience. I have fond memories of the "early days" of AMLab where I shared an office with Karen Ullrich, Patrick Putzky, Christos Louizos, Taco Cohen, and Ted Meeds. I could not have hoped for a more inspiring environment for the start of my PhD. I also count myself lucky to have had fantastic close collaborators at University of Amsterdam throughout my PhD: Michael Schlichtkrull, Rianne van den Berg, Elise van der Pol, Peter Bloem, and Jakub Tomczak. To all my other colleagues and friends in AMLab and at the Informatics Institute, thank you for making the past four years an unforgettable experience: Matthias Reisser, Patrick Forré, Herke van Hoof, Zeynep Akata, Bas Veeling (thank you for agreeing to serve as paranymph on my defense!), Daniel Worral, ChangYong Oh, Peter O'Connor, Andy Keller, Emiel Hoogeboom, Jorn Peters, Maurice Weiler, Tom Runia, Marco Federici, Maximilian Ilse, Tim Bakker, Wendy Shang, Victor Garcia, Wouter Kool, Sindy Löwe, Ana Lucic, Jörn Jacobsen (thank you for sharing your thesis template!), Giorgio Patrini, Mijung Park, Riaan Zoetmulder, Rajat Thomas, Sara Magliacane, Serhii Havrylov, Joost Bastings, Shi Hu, Tessa van der Heiden, Stephan Alaniz, and many more. Many thanks also to Félice Arends for her administrative support and to Tassilo Klein for managing our relation with SAP.

I have had the chance to work with several very talented MSc students more closely during my time at University of Amsterdam, including Nicola De Cao, Luca Falorsi, Tim Davidson, Davide Belli, Daniel Daza, Mart van Baalen, and Pascal Esser. It was a pleasure and I learned a lot from working with you!

I am also very happy to have had the opportunity to collaborate with fantastic researchers and faculty outside of University of Amsterdam: Ethan Fetaya, Raghav Selvan, Petar Veličković, Jackson Wang, Richard Zemel, Frans Oliehoek, Pietro Liò, Cătălina Cangea, Nikola Jovanović, Peter Boncz, Bernhard Radke, Viktor Leis, Peter Boncz, Alfons Kemper, Jesper Pedersen, Jens Petersen, and my brother Andreas Kipf.

I would like to thank Carlos Guestrin for hosting me in his team at Apple for an internship in the summer of 2017. Thank you to Chris, Drew, and Charles for being great mentors and collaborators. Credit also goes to my co-interns at Apple, especially Lars, Sharan, Almas, Lilian, and Shobhit, for making this a wonderful summer in Seattle.

I am also very grateful to have had the chance to intern at DeepMind under Peter Battaglia in the summer of 2018. Thank you, Pete, for your inspirational and supportive mentorship, and thank you to Yujia, Pushmeet, Edward, Hanjun, Vinicius, and Alvaro for being amazing collaborators. This summer wouldn't have been the same without all the wonderful conversations and activities with my co-interns and others at DeepMind, including Matthias, Natasha, Andy, Anurag, Adam, Jessica, Klaus, Guillaume, Victor, David, Kai, Razvan, Caglar, Kim, Andrea, and Toni.

Many thanks also go to friends from both near and far for providing support and welcome distraction, and simply for being in my life: Gabor, Anjali, Thomas, Mirthe, Matthias, Wiebke, Karolin, Max, Max, Flo, Jenny, Marvin, Nil-Jana, Sonya, Nathan, Jasmine, Pasindu, Nathalie, Elena, Markus, and others.

Maartje, thank you for enriching the past two years of my life. I could not wish for a better person to share my journey with. Thank you also for proofreading my thesis, for agreeing to serve as a paranymph on my defense, and for translating my thesis summary to Dutch.

Finally, I would like to thank my family. My brother, Andreas, not only had significant impact on the course of my PhD (we managed to write a paper together!), but I would not be where I am today as a person without having him in my life. To my parents, Harald (I wish you were still around) and Marianne: thank you for your incredible support, for enduring my absence in the past few years, and for always being or having been there for me.

Sincerely,

Thomas Kipf