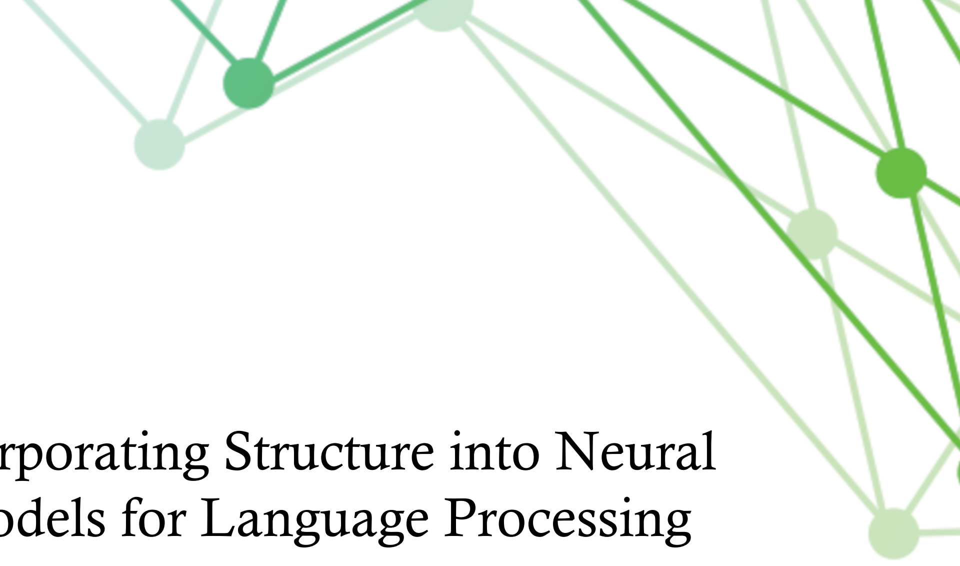
Incorporating Structure into Neural Models for Language Processing

Michael Sejr Schlichtkrull



# Incorporating Structure into Neural Models for Language Processing

Michael Sejr Schlichtkrull

# Incorporating Structure into Neural Models for Language Processing

ILLC Dissertation Series DS-2021-08

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

For further information about ILLC-publications, please contact

Institute for Logic, Language and Computation
Universiteit van Amsterdam
Science Park 107
1098 XG Amsterdam
phone: +31-20-525 6051
e-mail: `illc@uva.nl`
homepage: `http://www.illc.uva.nl/`

The investigations were supported by the Netherlands Organization for Scientific Research (NWO).

Cover design by Tegan Griggs.
Printed and bound by Minuteman Press, Cambridge.

# Incorporating Structure into Neural Models for Language Processing

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 dinsdag 29 juni 2021, te 15.00 uur

door Michael Sejr Schlichtkrull
geboren te Lyngby-Taarbæk

***Promotiecommissie***

| | | |
|---|---|---|
| *Promotor:* | dr. I.A. Titov | Universiteit van Amsterdam |
| *Copromotor:* | dr. W.H. Zuidema | Universiteit van Amsterdam |
| *Overige leden:* | prof. dr. I. Dagan | Bar-Ilan University |
| | dr. I. Augenstein | University of Copenhagen |
| | prof. dr. G.J.M. van Noord | Rijksuniversiteit Groningen |
| | prof. dr. K. Sima'an | Universiteit van Amsterdam |
| | dr. C. Monz | Universiteit van Amsterdam |
| | dr. W. Ferreira Aziz | Universiteit van Amsterdam |
| | prof. dr. E. Kanoulas | Universiteit van Amsterdam |

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

# Contents

# Acknowledgments

I would like to begin by thanking Ivan for his fantastic supervision and mentorship. I am incredibly grateful for your support, your guidance, your inspiration, and all the opportunities I have had over the years thanks to you. Thank you for making this journey possible. Many thanks also to my co-supervisor in Amsterdam, Jelle, who – despite the distance – has been very helpful and welcoming.

Towards the end of my first year as a PhD student, our group relocated from Amsterdam to Edinburgh. As such, I have had the good fortune of getting to know amazing people at not one, but two universities. I would like to thank first the wonderful people I met in our itinerant group: Nicola, Diego, Jasmijn, Serhii, Chunchuan, David, Caio, Phong, Xinchi, Lena, Bailin, Yanpeng, Arthur, Tom, and Ehsan. Thank you all for your support, and for all the good times; the beers, the food, the hikes, the badminton, and all the collaborations and discussions.

When I first arrived as a green student in Amsterdam, I was immediately taken in by the amazing people at UvA and ILLC. I would like to thank Wilker, Miguel, Marco, Elia, Samira, Raquel, Mostafa, Dieuwke, and Sara. Eight months was far too short to spend with you all, and my visits the past years have been far too infrequent.

In Edinburgh I received a warm welcome as well, met wonderful people, and had countless fruitful discussions. I would like to thank first of all the denizens, past and present, of office 3.50 – Benedek, Maria, Ivana, Paul, Ludovica, Pippa, and Natalia, it was a pleasure having you around. I would also like to thank all the other people who made my stay memorable; Desmond, Stella, and Milos, whom I already knew from Amsterdam, as well as Aciel, Laura, Annie, Rachel, Lea, Dominikus, Rico, Sharon, Shashi, Sabine, Ida, Jonathan, and many others.

Throughout the course of my PhD I have had the pleasure of collaborating with and learning from many incredible people at UvA and beyond. Many thanks go to Thomas, Rianne, Peter, and Max, as well as to Nicola and Wilker. I would like to also thank Siva, without whom Chapter 4 of this thesis would not have materialised.

Many thanks also to the members of my doctoral committee – Ido, Isabelle, Khalil, Gertjan, Christof, Wilker, Evangelos. Thank you for taking the time to read and discuss my work!

I would also like to thank Weiwei Cheng for hosting me at Amazon for the summer of 2018. Thank you to Weiwei for your mentorship and collaboration, and to Alexandre, Lea, Philip, Gyuri, and the rest of the team for your warm welcome and your support. I would like to thank also my fellow interns, Mathias and Lahari.

I am also tremendously grateful for the opportunity to intern with Sebastian Riedel and Mike Lewis at Facebook AI Research in the strange summer of 2020. Although circumstances conspired to prevent me from physically meeting the team, I was still welcomed with open arms. Thank you to Sebastian, Mike, Vladimir, Barlas, and Scott for our fantastic collaborations, and to all the other great people at FAIR London: Fabio, Patrick, Aleksandra, Tim, and many others. Thank you also to my fellow interns, especially Nicola and Ivana.

I would like to take this opportunity to also thank some of my oldest and best friends, Christopher and Charlotte. Thank you for supporting me, for keeping me sane, and for putting up with my ramblings about neural networks. I am also very grateful to Daniela, for all your support, and for being by my side for many of the years of my PhD.

Finally, I would like to thank my family – my parents, Henrik and Birgitte, and my sister Benedicte. Thank you for always being there for me, for believing in me, and for all the other ways you have supported me and continue to support me. I would not be where I am without you.

# List of Publications

This thesis is based, in part, on the following publications:

- **Michael Sejr 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).*

- **Michael Sejr Schlichtkrull**, Vladimir Karpukhin, Barlas Oğuz, Mike Lewis, Wen-tau Yih, and Sebastian Riedel (2020). "Joint Verification and Reranking for Open Fact Checking Over Tables". *arXiv:2012.15115.*

- **Michael Sejr Schlichtkrull**, Nicola De Cao, and Ivan Titov (2021). "Interpreting Graph Neural Networks for NLP With Differentiable Edge Masking". In: *International Conference on Learning Representations (ICLR).*

Ideas, text, figures, and experiments originate in majority from MSS. For the paper on *Modeling Relational Data with Graph Convolutional Networks*, the first two authors (denoted by *) contributed equally; MSS produced the part on relational link prediction, which Chapter 3 is based on, while TNK produced the part on node classification. All other authors had important advisory roles, and/or contributed to the writing of the original publications.

During the course of the project, the author has furthermore contributed to the following publications:

- Nicola De Cao, **Michael Sejr Schlichtkrull**, Wilker Aziz, and Ivan Titov (2020). "How do Decisions Emerge across Layers in Neural Models? Interpretation with Differentiable Masking". In: *Empirical Methods in Natural Language Processing (EMNLP).*

- **Michael Sejr Schlichtkrull**, Weiwei Cheng (2020). "Evaluating for Diversity in Question Generation over Text". *arXiv:2008.07291.*

- Barlas Oğuz, Xilun Chen, Vladimir Karpukhin, Stan Peshterliev, Dmytro Okhonko, **Michael Sejr Schlichtkrull**, Sonal Gupta, Yashar Mehdad, Wen-Tau Yih (2020). "Unified Open-Domain Question Answering with Structured and Unstructured Knowledge". *arXiv:2012.14610.*

Chapter 1

# Introduction

Computer systems around the world curate massive amounts of data, which can often only be accessed through artificial machine languages. From early experiments like Terry Winograd's SHRDLU to modern virtual assistants like Amazon's Alexa, widening accessibility to these systems through natural language interfaces is a long-standing dream. Neural networks have in recent years resulted in tremendous progress, especially for tasks where the knowledge source takes the form of English text. State-of-the-art models approach human performance on the leaderboards, even for challenging datasets like TriviaQA (Joshi et al., 2017; Zaheer et al., 2020). Models that query against massive collections rather than single documents are often not far behind (Izacard & Grave, 2020).

Many of the most interesting applications require querying not only against textual data, but also against various structured sources. Virtual assistants often operate in environments where the most relevant data comes in the form of tables, maps, social networks, and various metadata associated to elements in such structures. Systems operating in medical or scientific contexts must often process formalised descriptions of, for example, drugs, molecules, or interactions between proteins. And, as we will explore in this thesis in Chapters 3 and 4, knowledge bases like Freebase and Wikidata enable large-scale applications by storing millions upon millions of relational facts in the form of graph vertices and edges. Building models capable of querying against many forms of data has been shown to result in increased performance, even for datasets constructed with the aim of querying only against text (Sun et al., 2018; Sun et al., 2019b; Oğuz et al., 2020).

In this thesis, we investigate the construction of effective neural network models for natural language processing that incorporate structured information. We focus on two common forms of structure, knowledge bases and tables, and experiment with these in various NLP contexts. We study neural encoders for such structural objects, investigating how best to build these models, and how to interpret their predictions once they are trained.

When we began this project in 2016, a new class of models for directly encoding structured data had just started to pick up steam – graph neural networks (Gori et al., 2005; Scarselli et al., 2009; Kipf & Welling, 2017; Gilmer et al., 2017). Graph neural networks (GNNs) held the promise of incorporating structured sources into neural network systems in full end-to-end fashion. This stood in contrast to other contemporary models, which at the time mostly translated natural language into formal executable queries in machine languages like SPARQL or SQL (Berant & Liang, 2014; Reddy et al., 2014; Berant & Liang, 2015). With similarities to successful prior techniques for tree-structured data (Tai et al., 2015), GNNs seemed ideal for pushing the performance of models for querying against structured data to new heights. We set out to tailor variants of GNNs to NLP, and to deploy these models to directly integrate structured sources into NLP models.

We initially applied GNNs to model relational graphs for link prediction (see Chapter 3 as well as Schlichtkrull et al. (2018)), with promising results. Armed with techniques for applying GNNs to relational data, we then moved to the full natural language querying setting. As time and experience would prove, this was a much more difficult prospect – especially as the black-box nature of large neural networks made it difficult to understand and analyse the behaviour of our trained models. Modelling knowledge base question answering with GNNs is not an insurmountable challenge, as we demonstrate in Chapter 4. Furthermore, as later shown in Sun et al. (2018) and Sun et al. (2019b), applying GNNs can lead to state-of-the-art results on this task with a few tricks. Nevertheless, we decided to focus on two additional endeavours. First, the development of a technique for interpreting trained GNNs, to facilitate a less frustrating development process. Second, the investigation of a deceptively simple, rapidly rising competitor for modelling certain specific kinds of structured data: heuristic transformation into textual form, followed by modelling with large pretrained transformers (Chen et al., 2020; Yin et al., 2020).

Here, we chronicle our efforts towards effective neural modelling of structure within natural language processing; our initial development of models for link prediction, our experiences with knowledge base question answering, and our experiments with linearisation as well as with interpretability. We focus on graph-structured data, with a foray into table modelling to experiment with linearisation in Chapter 5. We develop novel graph neural network techniques suitable for modelling large knowledge bases, and investigate their strengths and weaknesses for different tasks; in Chapter 3 purely for modelling the knowledge base, and in Chapter 4 for querying the knowledge base in combination with a neural reader. We also investigate in Chapter 6 how users and developers can interpret and understand the predictions made by such models.

## 1.1 Objectives and Scope

At the start of our journey in 2016, graph neural networks were very much in an infant state. However, between several task-specific architectures (Duvenaud et al., 2015; Li et al., 2016b), performant but slower graph spectral convolutions (Bruna et al., 2014; Henaff et al., 2015), and the initial publication of Kipf and Welling's (2017) graph convolutional networks, the field was warming up. We initially intended to directly apply Kipf and Welling's (2017) framework to natural language querying against knowledge bases. In this, we were thwarted by the inability of existing models to move beyond simple graphs to the directed, multirelational case. As such, we found it necessary to begin by addressing the modelling of such graphs.

In the first chapter of this thesis, and in Schlichtkrull et al. (2018), we therefore extend Kipf and Welling (2017) to directed, multirelational graphs. We do not yet introduce natural language at this state. Instead, we apply our model to *relational link prediction*. That is, assessing the likelihood of a given fact being inferable from the facts already existing in a knowledge base. We use the GNN as an encoder, summarising the information in the neighbourhood of a vertex through a learnable function. We couple this with a factorisation model from the literature, acting as a decoder by scoring the likelihood of edges based on vertex embeddings produced by the GNN. Through this, we produce a model that, at the time of our experiments, attained state-of-the-art performance. We furthermore investigate which aspects of the problem our GNN-based approach models especially well.

Armed with a GNN architecture suitable for modelling relational data, we proceed in Chapter 4 to make our attempt on a GNN-based approach to the full knowledge base question answering problem. We experiment with and compare different variants of relational GNN-architectures, as well as two different strategies for modeling the problem. We base these on the strategy of combining an LSTM encoding the question with a GNN encoding the knowledge base. Through a thorough series of ablation tests, we probe our models to investigate what is necessary to develop a GNN for this task; we find, similar to the concurrent work by Sun et al. (2018), that the very high ratio of edges to vertices in real-world knowledge bases can overwhelm GNNs if not addressed through e.g. induced sparsity.

As we have mentioned, several recent papers (Chen et al., 2020; Yin et al., 2020) have reported successes using an alternative strategy for modelling specific kinds of structured data. In situations where the structure contains a pattern clear enough to interpret heuristically – tables, for example – an effective approach is implicit modelling by linearising the structure and treating it as text, rather than explicit modelling with a suitable neural network architecture. In Chapter 5, we experiment with such a technique as well. The problem we target is open-domain fact verification over tables – that is, verifying whether given facts are true or not, based on massive collections of tables. We demonstrate that linearisation is an effective strategy for modelling tables, allowing us to build the first open-domain table fact verification model. Through linearisation and modelling with RoBERTa (Liu et al., 2019), our model demonstrates open-domain performance exceeding even the previous closed-domain state of the art. We furthermore experiment with various strategies for improving performance by exploiting features of existing datasets, and finally apply our model to a truly large-scale setting: Querying against all tables on Wikipedia.

Throughout our process of experimentation, we realised that a significant barrier to the development of strong GNN-based models is the difficulty in understanding why certain models work while others break. Indeed, as has also been the experience elsewhere (Zaheer et al., 2017; Xu et al., 2019), seemingly small implementation differences can in some cases drastically change performance. Unfortunately, the nature of GNNs as highly complex, nonlinear functions often prevents researchers from understanding the inner workings of such models; in other words, they are *black boxes* (see Belinkov and Glass (2019)). To remedy this, we develop in Chapter 6, and in Schlichtkrull et al. (2020b) a technique for *interpreting* graph neural networks. Specifically, we investigate how to provide – in a tractable and faithful manner – rationales for the predictions of a given GNN. We make comparisons to recent related work, showing that our method performs favourably. We furthermore apply our technique to analyse two GNN-based models from the NLP literature.

## 1.2 Contributions

The primary contributions of this thesis can be summarised as follows:

(I) We introduce a method for relational link prediction in knowledge bases. Our approach combines Relational Graph Convolutional Networks (R-GCN), a novel GNN-based encoder, with a factorisation decoder from the literature. R-GCN achieves strong performance on standard benchmarks, with especially large gains for modelling high-degree neighbourhoods and low-frequency relations. While our focus is on relational link prediction, R-GCNs can potentially be applied to incorporate any graph-structured data in neural NLP models.

(II) We develop and experiment with GNN-based approaches to factoid question answering, introducing novel models based either on choosing individual answer vertices or on choosing the most likely path to the answer. In addition to R-GCN, we furthermore introduce a gated extension, Gated Relational Graph Neural Networks (GR-GNN), especially suitable for entity-based factoid question answering.

(III) We introduce the first model for open-domain fact verification over tables, proposing a method that exhibits open-domain performance exceeding the previous *closed-domain* state of the art. Our technique relies on named entity recognition to identify entities in claims, TF-IDF to retrieve relevant tables from the knowledge source, and an attention-based strategy for fusing the information of several tables encoded with RoBERTa (Liu et al., 2019).

(IV) We propose GraphMask, an interpretation technique for GNNs applicable potentially to any end-to-end neural network model which incorporates a GNN as a component. GraphMask shows which edges are useful for a GNN, and at which layer they are used. We apply our technique to artificial data to showcase how we address shortcomings in existing methods. We furthermore to perform two case studies where we analyse two models from the NLP literature.

Chapter 2

# Background

In this thesis, we address the modeling of complex structures with neural networks for natural language processing. The term "structure" has come to possess several different connotations within the literature – here, we focus specifically on *graphs* and *tables*. In what follows, we will give an introduction to the techniques we use to model said structures. We will later introduce additional background pertaining to the details of specific chapters where necessary, including details of the knowledge sources we work with.

The primary models used in this thesis are Graph Neural Networks (GNNs), neural models designed to encode and process graph-structured data. To properly define GNNs, we begin in Section 2.1 with a brief introduction to neural networks as a class of models. We then give a thorough discussion of the graph neural network framework in Section 2.2, introducing this class of models and summarizing the relevant literature. We furthermore discuss two popular subclasses of GNNs commonly used to facilitate efficient implementation. Finally, as we also rely on neural networks for sequence modeling in some parts of our work, we introduce these in Section 2.3.

## 2.1 Neural Networks

Neural Networks (NNs) are differentiable functions composed of simple building blocks, learned through backpropagation (Werbos, 1982). Neural networks take many forms, with the compositional nature allowing practitioners to develop unique modules suitable for dealing with various tasks and inputs. To give a very general definition, we can say that:

**2.1.1.** Definition. A neural network $F_\theta$ with parameters $\theta = \theta_1, ..., \theta_l$ is a differentiable composite function $F_\theta = f^{(1)} \circ ... \circ f^{(k)}$ such that $f^{(m)}$ is either an affine transformation with parameters $\theta_n$ or some intermediate function.

That is, the parameterised building blocks are affine transformations, which are combined in various ways to produce complex, nonlinear functions. Since the learnable parameters originate from affine transformations $f(h) = W^{(n)}x + b^{(n)}$ such that $\theta_n = W^{(n)}, b^{(n)}$, parameters are also commonly referred to as *weights* and *biases*. Some larger, composite building blocks often reoccur in stacked fashion within the network. It is common to refer to these as *layers*.

The classical case of neural networks is the Multilayer Perceptron (MLP), in which the components are simply affine transformations interspersed with non-linear "activation functions", e.g. sigmoids or Rectified Linear Units. In our work, we often use MLPs to transform the output of some complex network into a low-dimensional prediction.

**2.1.2.** Definition. A multilayer perceptron $F_\theta$ is a neural network with parameters $\theta = \theta_1, ..., \theta_L$ and $L$ layers such that $F_\theta$ is composed only of affine transformations interspersed with nonlinear activations:

$$F_\theta = \sigma \circ f^{(L)}_{\theta_L} \circ ... \circ \sigma \circ f^{(1)}_{\theta_1}$$

where $f^{(l)}(x) = W^{(l)}x + b^{(l)}$, and $\sigma$ is an activation function.

## 2.2 Graph Neural Networks

Graph neural networks (GNNs) are a class of neural network models developed for the processing of graph-structured data. Formally, a GNN is a layered neural network architecture which takes as input a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E}, \mathcal{R} \rangle$ (i.e., nodes, edges, and relation types), along with a set of initial features $F^0$. Among these initial features are a set of vertex embeddings $H^{(0)} \in \mathcal{V} \times \mathbb{R}^d$ providing a $n$-dimensional encoding $h_v^{(0)}$ for every vertex $v$. The GNN then computes, in layerwise fashion, new contextualized vertex embeddings $H^{(l)} \in \mathcal{V} \times \mathbb{R}^m$ for each layer $l$, taking into account the neighbourhood surrounding each vertex. By feeding the embeddings $H^{(l)}$ into the network to compute $H^{(l+1)}$, every new layer is capable of encoding information from wider contexts than the previous one. GNNs as such provide contextualized vertex features for downstream prediction tasks, learnable in a fully differentiable fashion.

At every layer $k$, a GNN computes a representation $h_v^{(k)}$ for every vertex $v \in \mathcal{V}$ based on the vertex representations $H^{(k-1)}$ computed at the previous layer. To incorporate information about edges and other vertices in the local neighbourhood, this computation takes the form of a *message passing* step. That is, every vertex communicates information about itself to adjacent vertices. In applications with directed graphs, messages are commonly sent along both directions of every edge (see e.g. Marcheggiani and Titov (2017), Schlichtkrull et al. (2018), and Kampffmeyer et al. (2019)). Then, the direction is treated as a feature $\delta \in \Delta$, where $\Delta = \{\leftarrow, \rightarrow\}$. The message passing step of a graph neural network is as such defined at layer $k$ through a learnable *message function* $M_\theta^{(k)}$:

**2.2.1.** Definition. A message function at layer $k$ with learned parameters $\theta$ is a function $M_\theta^{(k)} : \mathbb{R}^{d_v} \times \mathbb{R}^{d_v} \times \mathcal{R} \times \Delta \rightarrow \mathbb{R}^{d_m}$.

Through the message function at each layer, the graph neural network computes a *message* for every edge. Formally:

**2.2.2.** Definition. Given an edge $e = (u, r, v, \delta)$ with source node $u$, target node $v$, relation type $r$, and direction $\delta$, the *message* $m_e^{(k)}$ associated with $(u, r, v, \delta)$ at layer $k$ is defined as:

$$m_e^{(k)} = M_\theta^{(k)} \left( h_u^{(k-1)}, h_v^{(k-1)}, r, \delta \right)$$

In addition to the message passing step, the graph neural network also performs an *aggregation step* to combine the information of multiple messages. The aggregation step is defined through a learnable *aggregation function*. Aggregation relies on the concept of the neighbourhood edge set $\mathcal{N}_G(v)$ of the vertex $v$, which is defined as follows:

**2.2.3.** Definition. The neighbourhood edge set $\mathcal{N}_\mathcal{G}(v)$ of the vertex $v$ in the graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E}, \mathcal{R} \rangle$ is the set of ingoing edges to $v$:

$$\mathcal{N}_\mathcal{G}(v) = \{(s, r, t, \delta) \in \mathcal{E} \mid t = v\}$$

The graph neural network then defines a learnable aggregation function $A_\pi^{(k)}$ such that:

**2.2.4.** Definition. An aggregation function at layer $k$ with learned parameters $\pi$ is a function $A_\pi^{(k)} : \mathbb{R}^{d_v} \times \mathcal{P}(\mathbb{R}^{d_v}) \rightarrow \mathbb{R}^{d_v}$ where $\mathcal{P}(\mathbb{R}^{d_m})$ is the powerset of $\mathbb{R}^{d_m}$.

That is, it is a function which aggregates a set of $d_m$-dimensional vectors on the basis of another $d_v$-dimensional vector. The aggregation step can then be defined for each vertex, and the whole computation of the graph neural network can be recursively specified as:

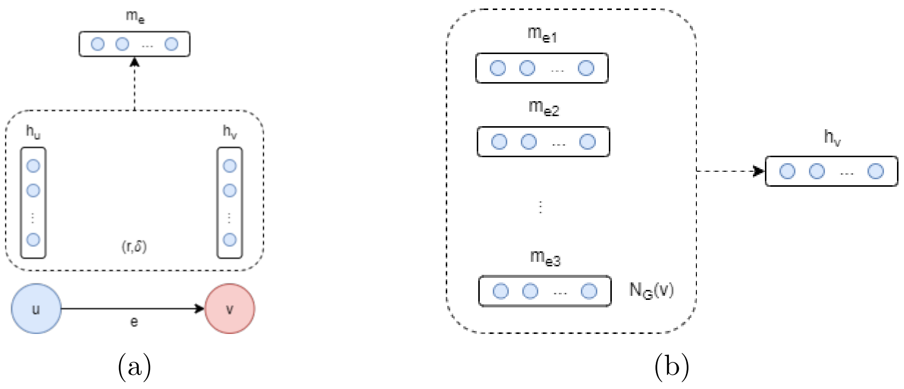


Figure 2.1: Graph neural networks are defined through (a) a learnable message function $M_\theta^{(k)}$ computing a message $m_e$ for every edge $e$, and (b) a learnable aggregation function $A_\pi^{(k)}$ combining the messages in the neighbour set $\mathcal{N}_\mathcal{G}(v)$ to produce higher-layer embeddings for the vertex $v$.

**2.2.5.** DEFINITION. Given a vertex $v$ with encoding $h_v^{(k-1)}$ at layer $k-1$, the encoding $h_v^{(k)}$ is computed as:

$$h_v^{(k)} = A_\pi^{(k)}\left(h_v^{(k-1)}, \left\{m_e^{(k)} : (u, r, v, \delta) \in \mathcal{N}_\mathcal{G}(v)\right\}\right)$$

This very general definition of graph neural networks captures the full class of models represented within the framework. The terminology introduced here is based on *neural message passing* as developed by Gilmer et al. (2017). Their formulation is itself an attempt to unify several earlier models which restrict either message passing or aggregation to particular families of functions, e.g. graph convolutional networks (Kipf & Welling, 2016) and interaction networks (Battaglia et al., 2016).

GNNs were first introduced in Gori et al. (2005) and Scarselli et al. (2009) as an extension of the earlier, very general recursive neural network framework (Goller & Kuchler, 1996; Frasconi et al., 1998). These were later instantiated for specific applications and updated with modern training practices in Duvenaud et al. (2015) and Li et al. (2016b). Concurrently, Bruna et al. (2014) and Henaff et al. (2015) introduced slower – but more generally applicable – variants based on spectral graph theory. Models more closely resembling modern GNNs were proposed in Niepert et al. (2016), Defferrard et al. (2016), and Kipf and Welling (2017), with the latter being the forefather of the computationally efficient subclass known as graph convolutional networks (see below).

In NLP, Socher et al. (2011) applied recursive neural networks for parsing tree-structured representations in text and images – syntax trees and scene graphs. Subsequently, Tai et al. (2015) improved greatly on these results through the introduction of Tree-LSTMs, a combination of Long Short-Term Memory Networks (Hochreiter & Schmidhuber, 1997) and recursive neural networks. Tree-LSTMs can be seen as a subclass of GNNs designed for trees rather than general graphs.

In the years since we started working with GNNs, the framework has exploded with variety in structure as well as in application.[1] Subfamilies have been proposed to handle unseen vertices at test time (Hamilton et al., 2017), select relevant messages through attention mechanisms (Veličković et al., 2018), enable deeper GNNs by updating only select dimensions of vertex embeddings (Beck et al., 2018), infer relation- and entity-types on the fly (Sun et al., 2019a), and many other uses. We refer the reader to Wu et al. (2020) for a comprehensive survey.

The full graph message passing framework is highly useful for reasoning about graph neural networks. It is furthermore necessary when designing algorithms that function for *all* GNNs, rather than any particular subset (e.g. our interpretation technique GraphMask, which we introduce in Chapter 6). Nevertheless, several subclasses of GNNs are very useful for their lower computational or space complexity, or their ease of implementation. Here, we will discuss two such subfamilies, *graph convolutional networks* and *incidence matrix networks*.

## 2.2.1 Graph Convolutional Networks

Graph convolutional networks[2] (GCNs) are a class of graph neural networks first introduced by Kipf and Welling (2017). GCNs take advantage of sparse matrix multiplication with the adjacency matrix to perform very efficient message passing. Formally, this corresponds to defining message passing and aggregation as:

$$\begin{aligned} M_\theta^{(k)} &= \sigma(W_\theta h_u^{(k-1)}), \\ A_\pi^{(k+1)} &= W_{self} h_v^{(k)} + \frac{1}{N_v} \sum_{(u,r,v,\delta)\in\mathcal{N}_\mathcal{G}(v)} m_e^{(k+1)} \end{aligned}$$

where $\sigma$ is an activation function such as the Rectified Linear Unit, and $N_v$ is a normalization constant associated with $v$ (typically the number of incoming edges). This can be expressed very efficiently in terms of matrix multiplication, as we do below in Definition 2.2.6.

[1] At time of writing, PyTorch Geometric (Fey & Lenssen, 2019), one of the most popular GNN libraries, implements GNN variants from 72 different papers.

[2] It is worth noting that the terms *graph convolutional network* and *graph neural network* are sometimes used interchangeably. Nevertheless, some GNNs, such as the variant of GraphSAGE (Hamilton et al., 2017) which uses an LSTM as the aggregation function, explictly move beyond convolution. We choose to use the term *graph convolutional network* to denote the subclass introduced under that name by Kipf and Welling (2017).

**2.2.6.** Definition. Given a normalized adjacency matrix $A'$, the embedding matrix $H^{(k+1)}$ of all vertices at layer $(k+1)$ in a graph convolutional network is:

$$H^{(k+1)} = A'\sigma(W^{(k+1)}H^{(k)}) + \sigma(W_{self}H^{(k)})$$

Certain additional modifications beyond normalization are often made to the adjacency matrix. For example, it is common to also add self-loops to the matrix to efficiently reuse lower-layer embedding matrices without having to explicitly compute self-connections through $W_{self}$. This is commonly done by defining the modified adjacency matrix as:

$$A' = D^{-1}A + I_N$$

where $D^{-1}$ is the inverse degree matrix and $I_N$ the identity matrix.

GNNs expressible within the graph convolutional network scheme are fast to compute and easy to implement, since they require only a single sparse matrix multiplication beyond standard neural network operators. As multiplication with a sparse adjacency matrix requires one computational step per edge, the computational complexity is simply $O(|\mathcal{E}|)$. Furthermore, as the only storage required is the embeddings of source and target vertices, the space complexity is $O(|\mathcal{V}|)$. These models have been employed to apply GNNs in situations where the number of edges would make more complex formulations intractable (e.g. De Cao et al. (2019)).

## 2.2.2 Incidence Matrix Networks

The GCN framework, while highly efficient, does come with a significant drawback. Following Definition 2.2.6, all messages are constructed on the basis of vertices, through the matrix multiplication $W^{(k+1)}H^{(k)}$. As such, it is difficult to implement models which perform different computations for different edges that share a source node. This is relevant for example if those edges have different relation types, and the message passing function as a consequence defines different computations for them.

In Chapter 3, we extend the GCN framework to labeled, directed graphs. Our extension requires different steps of GCN computation as described in Definition 2.2.6 for every relation type in the dataset, because different weight matrices are used. If the set of relations is small, this can be efficiently computed as separate instantiations of graph convolutional network updates; if the set of relations is large, this can become prohibitively expensive.

In such cases, it is simpler to instead compute two sparse matrix multiplications – one step to distribute information to messages, and one step to distribute messages to target vertices. While this scheme has not been formally stated as

GCNs were in Kipf and Welling (2017), it is a common way[3] to circumvent the limitations of GCNs without resorting to the full graph message passing framework. As the latter is only necessary for a few models, e.g. the variant of GraphSAGE (Hamilton et al., 2017) which uses an LSTM as the aggregation function, this two-step scheme captures most commonly used GNNs. Since this subclass of GNNs is expressed through multiplication with an incidence matrix, we introduce the term Incidence Matrix Network (IMN) to describe it.

The incidence matrix and inverse incidence matrix necessary to define such models describe how vertices and edges in the graph relate. Formally:

**2.2.7.** Definition. Given a directed graph $\mathcal{G}$ with vertices $\mathcal{V} = v_1, ..., v_n$ and edges $\mathcal{E} = e_1, ..., e_m$, the incidence matrix $I \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{V}|}$ and inverse incidence matrix $\hat{I} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{E}|}$ are defined such that

$$I_{ij} = \begin{cases} 1 & \text{if } v_j \text{ is the target vertex of } e_i \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{I}_{ji} = \begin{cases} 1 & \text{if } v_j \text{ is the source vertex of } e_i \\ 0 & \text{otherwise} \end{cases}$$

Computing an individual message for every edge can be efficiently done through sparse multiplication with the inverse incidence matrix, especially if the embedding of the target vertex is excluded. This can be accomplished by modifying the message passing step from Definition 2.2.2 as follows:

**2.2.8.** Definition. In an incidence matrix network, the message matrix $\hat{M}^{(k+1)}$ associated with edges at layer $k$ is defined given the embedding matrix $H^{(k)}$ edges with relation types $R = r_1, ..., r_m$ and directions $\hat{\delta} = \delta_1, ..., \delta_m$ as:

$$\hat{M}^{(k)} = M_\theta^{(k)}\left(IH^{(k-1)}, R, \hat{\delta}\right)$$

Aggregation can then be efficiently computed by summation over the incoming messages, in similar fashion to how GCNs sum over adjacent vertices:

[3] Libraries for graph neural networks often implicitly implement the incidence matrix network scheme (see for example PyTorch Geometric (Fey & Lenssen, 2019)), as it strikes a good balance between speed and expressibility. It is furthermore easy to implement efficiently through the *scatter* and *gather* operations.

**2.2.9.** Definition. In an incidence matrix network the embedding matrix $H^{(k+1)}$ at layer $k+1$ is defined as:

$$H^{(k+1)} = \hat{I}\hat{M}^{(k+1)}$$

As with GCNs, self-loops and normalization can be efficiently handled by appropriately scaling the incidence and inverse incidence matrices.

IMNs require two steps of sparse matrix multiplication, each with a computational complexity of $O(|\mathcal{E}|)$; this is similar to the single sparse matrix multiplication step required for GCNs. However, IMNs must store every message individually, as opposed to simply storing vertex embeddings. The space complexity is therefore $O(|\mathcal{E}|+|\mathcal{V}|)$. As the number of edges for many graphs is much greater than the number of vertices, this can be prohibitive compared to GCNs.

## 2.3 Sequence Modelling

In addition to graphs, several of the tasks we examine also require models to incorporate sequences. This is necessary to handle raw text and other similarly unstructured data. In Chapter 4 we model questions for a question answering task, and in Chapter 5 statements and linearised tables for a fact verification task. Modeling sequences plays an important role in NLP, as sentences are sequences of tokens.

The classical neural approach is to use Recurrent Neural Networks (RNNs), specifically Long Short-Term Memory networks (LSTMs). These have been applied with great success in tasks ranging from translation (Sutskever et al., 2014) to dependency parsing (Kiperwasser & Goldberg, 2016). RNNs operate by recursively applying a neural network function to the elements of the sequence, computing output $y_t$ for the token at position $t$ using the output $y_{t-1}$ of the prior token along with an input $x_t$ at position $t$. In this manner, RNNs compute a sequence of outputs $(y_1, ..., y_t)$ given a sequence of inputs $(x_1, ...x_t)$. LSTMs are a specific variant designed to prevent vanishing gradients (Hochreiter & Schmidhuber, 1997). They track a cell-state $c_t$ and output a hidden state $h_t$ for every timestep $t$. These hidden states can then be further processed into outputs through e.g. an MLP.

**2.3.1.** Definition. A *Long Short-Term Memory* unit is a recursive function $LSTM_\theta : \mathbb{R}^{d_i} \times \mathbb{R}^{d_h} \times \mathbb{R}^{d_c} \rightarrow \mathbb{R}^{d_h} \times \mathbb{R}^{d_c}$ with parameters $\theta$ such that:

$$\begin{aligned} c_t &= \sigma(W_f I + b_f) \cdot c_{t-1} + \sigma(W_i I + b_i) \cdot tanh(W_c I + b_c), \\ h_t &= \sigma(W_o I + b_o) \cdot tanh(c_t), \\ LSTM_\theta(x, h_{t-1}, c_{t-1}) &= (h_t, c_t) \end{aligned}$$

where $\sigma$ is the sigmoid function, and $\cdot$ represents element-wise multiplication.

We use LSTMs to model questions in Chapter 4.

In recent years, LSTMs have been overtaken for many tasks by another class of models – *transformers* (Vaswani et al., 2017). In Chapter 5, we rely on such a model for parsing tables. Like RNNs, transformers compute sequences of vectorial token embeddings $(h_1, ..., h_t)$ given sequences of input $(x_1, ...x_t)$.[4] However, unlike RNNs, this computation relies on learned weighted averages rather than recurrence to produce output for each position.

The idea of using learned weighted averages to summarize sequences began in machine translation with Bahdanau et al. (2014) and Luong et al. (2015), and was originally used as a component of RNN-type models. In these models, attention was proposed as a learnable method for computing alignment between different elements. Vaswani et al. (2017) introduced the transformer, wherein stacked layers of attention replace the RNN entirely. The specific type of attention function used for the transformer architecture is known as *scaled dot-product attention*. A simple neural network produces for each position $t$ in the sequence a query $Q_t$, a key $K_t$, and a value $V_t$. Then, the dot product of the key and the query are used to compute a weighting, according to which the values are averaged.

**2.3.2.** DEFINITION. Scaled dot-product attention is a function $A : \mathbb{R}^{l\times d} \times \mathbb{R}^{l\times d} \times \mathbb{R}^{l\times d} \to \mathbb{R}^{l\times d}$ such that

$$A(Q, K, V) = \sigma\left(\frac{QK^\intercal}{\sqrt{d}}\right)V$$

where $\sigma$ is the softmax function.

Rather than rely on a single alignment score for each pair of tokens, transformers rely on multiple *heads* computing scaled dot-product attention in parallel. The outputs are then concatenated and linearly transformed into the expected dimensions. This allows the model to learn to perform multiple different functions at each layer, and to combine information from different parts of the sequence.

**2.3.3.** DEFINITION. Multi-head attention is a function $A : \mathbb{R}^{l\times d} \times \mathbb{R}^{l\times d} \times \mathbb{R}^{l\times d} \to \mathbb{R}^{l\times d}$ such that

$$\begin{aligned} MultiHead(Q, K, V) &= W_O[head_1, ..., head_h], \\ head_i &= A(QW_i^Q, KW_i^K, VW_i^V) \end{aligned}$$

[4] We note that transformers have been successfully applied to non-sequential data as well, such as images (Parmar et al., 2018). For the purposes of this thesis, however, we can restrict ourselves to the sequential case.

The transformer intersperses attention functions, affine transformations, and layer normalization (Ba et al., 2016) to produce contextualized embeddings for the sequences.

**2.3.4.** DEFINITION. A transformer is a neural network $F_\theta$ composed of attention layers $a_1, ..., a_k$, multi-layer perceptron layers $m_1, ...m_k$, and layer normalizations $l_1, ..., l_k$ combined such that

$$F_\theta = l_k \circ m_k \circ a_k \circ ... \circ l_k \circ m_k \circ a_k$$

We refer the reader to Vaswani et al. (2017) for further details.

While developed independently of graph neural networks, transformers can be seen as a special case of graph neural networks applied to the fully connected graph. From that perspective, transformers implicitly learn which edges are relevant through attention. The matrix multiplication between $V$ and the softmax-scaled attention scores in Definition 2.3.2 correspond to multiplication between prior vertex embeddings (e.g. $V$) and an adjacency matrix scaled by learnable factors.

That is, given a graph $\mathcal{G}_S = \langle \mathcal{V}, \mathcal{E} \rangle$ for the sequence $S$ such that $\mathcal{V}$ contains the tokens of $S$ and $\mathcal{E}$ all possible edges between any two vertices, scaled dot-product attention is equivalent to a layer of the GNN-architecture introduced in Veličković et al. (2018). The difference between the transformer and their model then lies only in the details of how transformations and layer normalization *between* attention steps are used. By exploiting this equivalence, some of the ideas developed in this thesis (e.g. GRAPHMASK) could then be applied to the transformer architecture.

## 2.3.1 Pretrained Language Models

A recently popularized method for high-performance modeling of natural language sequences is to *pretrain* sequences models on large datasets gathered from the Internet. These models can then be applied as sequence encoding components in task-specific neural networks. In Chapters 4 and 5, we make use of such models.

We model questions in the question answering task in Chapter 4 using so-called ELMo-embeddings (Peters et al., 2018). ELMo (shorthand for Embeddings from Language Models) relies on bidirectional LSTM-embeddings of sentences, trained as the name suggests with a language modeling objective. That is, given a sequence of $N$ tokens $(t_1, t_2, ..., t_N)$, ELMo learns a forward language model which expresses the probability of token $t_k$ given the history $(t_1, ..., t_{k-1})$:

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_1, ..., t_{k-1}) \tag{2.1}$$

and a backward model which expresses the probability of token $t_k$ given the future $(t_{k+1}, ..., t_N)$:

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_{k+1}, ..., t_N) \tag{2.2}$$

ELMo uses $L$-layer bidirectional LSTMs to learn these probabilities, training on a massive corpus such as the One Billion Word Benchmark (Chelba et al., 2013). These pretrained LSTMs can then be applied to produce contextualized embeddings of the words in sentences for concrete NLP tasks, such as the question answering task in Chapter 4. When applied, the embeddings are finetuned by updating the parameters of a linear combination between the $L$ layers.

Since the initial publication of ELMo, the use and variety of such pretrained encoders have increased explosively. An important step was the development of BERT (Devlin et al., 2019), which replaced the LSTM in ELMo with a transformer. Furthermore, since BERT no longer relies on a model which reads the input sentence token-by-token, the authors also replaced the classical language modeling objective described above in Equations 2.1 and 2.2 with the more appropriate Masked Language Modeling objective (also known as the Cloze task (Taylor, 1953)). That is, BERT masks some set of input tokens at random, and then predicts those masked tokens given the rest. The tokens $(t_1, t_2, ..., t_N)$ are randomly split into two sets $T_{masked}$ and $T_{clear}$, and the objective is to predict the former given the latter:

$$p(T_{masked}) = \prod_{t \in T_{masked}} p(t | T_{clear}) \tag{2.3}$$

The variety of BERT-inspired pretrained encoders is so large that some authors have referred to the study of their respective strengths and weaknesses as "BERTology" (Rogers et al., 2021). In Chapter 5, we employ one such variant – RoBERTa (Liu et al., 2019) – to encode statements and linearised tables for our fact verification task. RoBERTa does not change the BERT model itself; rather, the authors carefully tune the hyperparameters and expand the training corpus from a size of 16GB to 160GB.

By themselves, the language models we have discussed here construct LSTM- or transformer-based embeddings of individual sequence elements. That is, given a sequence of inputs $t_1, ..., t_N$, they produce embeddings $h_1, ..., h_N$ where $h_t \in \mathbb{R}^d$ for every $h_t$. A common use case, however, is to embed entire sentences. In these cases, a special token $t_0$ is commonly added to the sequence, and the corresponding embedding $h_0$ is used to represent the entire sentence.

Chapter 3

# Graph Neural Networks for Relational Link Prediction

## 3.1 Introduction

An important form of graph-structured information for systems to draw on are *Knowledge Bases* (KBs). Relational data – that is, data representing relations between subjects and objects – is abundant in the world, and knowledge bases organise collections of such data into a format that can be queried and used for many tasks. Knowledge bases enable a wide variety of applications both within and without NLP, including question answering (Yao & 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 & Zhai, 2012; Dalton et al., 2014; Xiong & Callan, 2015b, 2015a). We note that the words *knowledge base* and *knowledge graph* are often used interchangeably.

The amount of information stored in real-world knowledge bases is massive,[1] and difficult to tractably model. Previous techniques often rely on approximations modelling entities through random walks in their neighbourhoods (Perozzi et al., 2014), or through pretrained embeddings storing information about entities (Yang et al., 2015). While such techniques often perform well, they can struggle to exploit the structure of the underlying graph fully – random walks are unlikely to cover a neighbourhood fully, and techniques relying on pretraining cannot filter the information encoded in a vertex embedding to focus on what is relevant for

[1] For example, Freebase (Bollacker et al., 2008) contains approximately 1.9 billion triples, DBPedia (Auer et al., 2007) 3.2 billion, YAGO (Rebele et al., 2016) 1.2 billion, and Wikidata (Vrandečić & Krötzsch, 2014) a continuously growing 4.9 billion. These overlap, in part due to the ongoing efforts to import Freebase into Wikidata, and Wikidata into DBPedia and YAGO.

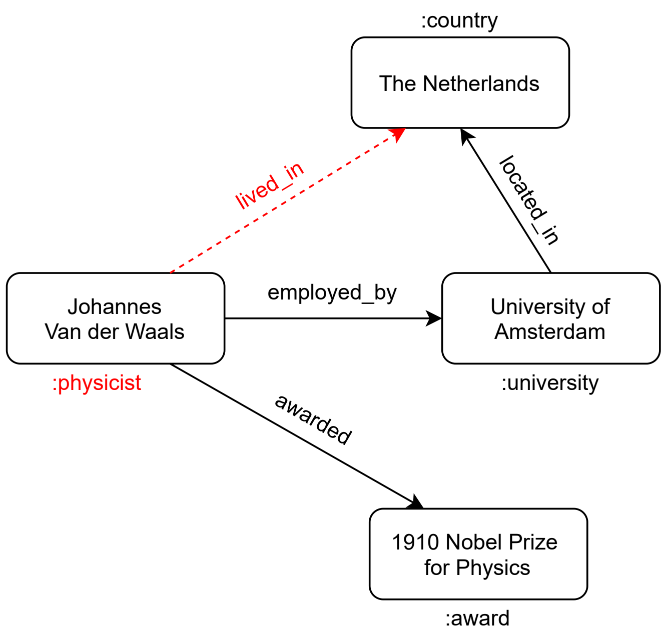


Figure 3.1: A fragment of a knowledge base. The nodes are entities, the edges are relations labeled with relation types. Similarly, the nodes are labeled with entity types (e.g. *university*). The edge and the entity type shown in red represent missing information to be inferred.

any particular task. An important step towards enabling modern NLP systems to better use the information stored in knowledge bases is therefore to develop stronger methods for modelling this information.

Graph neural networks (see our review in Section 2.2) represent a promising approach to encoding vertices. Through learned weights and normalisations, GNNs can ensure that the *right* information is encoded in each vertex. By relying on neighbourhood structure rather than static embeddings, GNNs can also transfer to parts of the graph unseen during training. This is important for real-world cases of massive graphs, where training can only realistically happen on a subset of the vertices and edges (e.g. Freebase, which we rely on in this chapter as well as in Chapter 4). Transferring to unseen vertices and edges is also necessary for any real-world knowledge base, since they continuously update.

Even the largest real-world knowledge bases remain incomplete, despite the massive efforts invested in their maintenance. Indeed, the constant discovery of new entities and new relational facts, along with the need for human effort to manually enter facts into knowledge bases, virtually guarantees that no such database will ever be complete. Luckily, automatic reasoning can facilitate the recovery of missing information. This is done by inferring what *should* be in the knowledge base on the basis of what *is* in the knowledge base. The prediction of such information is the main focus of *statistical relational learning*. There is a clear correspondence between automatically *using* knowledge bases and automatically *completing* knowledge bases, as both tasks require similar modelling strategies. As such, statistical relational learning is a good starting point when developing techniques to model knowledge bases.

Two fundamental tasks within the statistical relational learning setting are *link prediction*, the discovery of binary facts representing a relation between two entities, and *node classification*, the discovery of unary facts describing a single entity. In the example in Figure 3.1, the knowledge that Van der Waals worked at the University of Amsterdam could let a model infer that he likely lived in the Netherlands. Similarly, the knowledge that he won a Nobel Prize in physics could let a model infer that he likely was a physicist by profession.

The neighbourhood structure of each vertex as such contains important information for both of these tasks. In Figure 3.1, explicitly encoding the relationship between Van der Waals and the University of Amsterdam when modelling either would allow models to make inferences on this basis. In this chapter and in Schlichtkrull et al. (2018), we introduce a graph neural network method for knowledge base completion.[2] Here, we focus on the link prediction task, and we direct the reader to Schlichtkrull et al. (2018) for the node classification task. We attempt to answer the following research questions:

1. *Can graph neural network encoders improve the modelling of entities in knowledge bases for relational link prediction?*

2. *Which aspects of relational graph structure do graph neural networks capture better than baseline techniques?*

The link prediction model we introduce here consists of a GNN encoder based on Graph Convolutional Networks (Kipf & Welling, 2017) for entities, paired with a decoder that predict edges from vector representations of vertices. The original GCN framework performs well for modelling graph structure, but is not directly applicable to labelled, directed graphs like knowledge bases. We extend GCNs through the simple approach of using separate weights and normalisation constants for each direction and relation type. Since this drastically increases the number of weights and hence the complexity of the model, we also introduce a technique to ensure sparsity in the weights. This is done by only allowing non-zero weights in blocks around the diagonals of the weight matrices. Inspired by prior work on relational link prediction, we use the DistMult (Yang et al., 2015) factorization to decode potential edges. All parameters are trained using a novel graph autoencoder loss.

[2] This chapter is based on our ESWC 2018 publication, Schlichtkrull et al. (2018). The first two authors contributed equally to this publication. My contribution was the part addressing relational link prediction presented in this thesis, while Thomas Kipf contributed the corresponding part addressing node classification.

Our primary contributions in this chapter are the extension of graph neural networks to relational data for link prediction, as well as a sparsity constraint that reduces the number of parameters and allows the application of our model to large multirelational graphs. We validate our model on the FB15k-237 dataset, and report results also for the previously used FB15k and WN18 datasets. We publish the code for our experiments at https://github.com/MichSchli/RelationPrediction.

## 3.2 Background

For the purposes of relational link prediction as explored in this chapter,[3] we may think of a knowledge base as a directed, labeled multigraph $G = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ with vertices (entities) $v_i \in \mathcal{V}$ and labeled edges (relations) $(v_i, r, v_j) \in \mathcal{E}$, where $r \in \mathcal{R}$ is a relation type. We explicitly add inverse relations to the set of total relations $\mathcal{R}$, i.e. for every edge $(v_i, r, v_j)$ in the knowledge base we also include a separate edge $(v_j, r_{inverse}, v_i)$ in $\mathcal{R}$. We do not include node labels or address the problem of node classification here – we refer the reader to Schlichtkrull et al. (2018) for details on how to address that task.

Relational link prediction is the automatic discovery of new facts, e.g. new triples *(subject, relation, object)* which can be inferred to be missing from the knowledge base. That is, rather than the full set of edges $\mathcal{E}$, we are given only an incomplete subset $\hat{\mathcal{E}}$. The task is to assign scores $f(v_i, r, v_j)$ to possible edges $(v_i, r, v_j) \notin \hat{\mathcal{E}}$ in order to determine how likely those edges are to belong to $\mathcal{E}$.

Previous work on link prediction for knowledge bases has focused primarily on factorisation techniques. In these, scores of possibly missing edges are computed by decomposing into factors originating from the subject, the relation type, and the object. As we elaborate on in Section 3.3, our approach incorporates a previously existing factorization technique – DistMult (Yang et al., 2015) – as the decoder. DistMult is a special, simplified case of the earlier RESCAL factorization (Nickel et al., 2011). DistMult tends empirically to be more effective than the original RESCAL in the context of multirelational knowledge bases. RESCAL computes scores through a simple bilinear product; DistMult does the same, under the additional assumption that the weight matrix is diagonal. That is, a triple $(s, r, o)$ representing a subject $s$, a relation type $r$, and an object $o$ is scored through decomposition into individual factors originating from $s$, $r$, and $o$.

[3] It should be noted that real-world knowledge bases are somewhat more complicated, owing to the existence of so-called Compound Value Types – intermediary vertices representing n-ary relations. For further discussion, see Chapter 4.

**3.2.1.** Definition. Given a triple $(s, r, o)$, embeddings $e_v \in \mathbb{R}^d$ for every vertex $v$, and a diagonal matrix $R_r \in \mathbb{R}^{d \times d}$ for every relation $r$, the DistMult score of $(s, r, o)$ is defined as

$$f(s, r, o) = e_s^\intercal R_r e_o$$

Numerous alternative factorizations have been proposed and studied in the context of relational link prediction, including both (bi-)linear and nonlinear techniques (e.g. Bordes et al. (2013), Socher et al. (2013), Chang et al. (2014), Nickel et al. (2016), and Trouillon et al. (2016)). Many of these approaches can be regarded as modifications to or special cases of classic tensor decomposition methods such as canonical polyadic decomposition (Hitchcock, 1927), as discussed in Balazevic et al. (2019). For a comprehensive overview of tensor decomposition literature, we refer the reader to Kolda and Bader (2009).

The incorporation of paths between KB entities has previously received considerable attention. We can roughly classify the prior work into (1) methods creating auxiliary triples, which are then added to the learning objective of a factorisation model (Guu et al., 2015; Garcia-Duran et al., 2015); (2) approaches using paths (or walks) as features when predicting edges (Lin et al., 2015); and (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 similar terms to our objective (in other words, extending our decoder). The second research line is more comparable; R-GCNs provide a computationally cheaper alternative to these path-based models. Direct comparison is somewhat complicated, as path-based methods typically use different datasets (e.g. precomputed, sub-sampled sets of walks from a knowledge base).

Concurrently with our work, Marcheggiani and Titov (2017) introduced a GNN operating in syntactic dependency graphs. Their model encodes relation labels by conditioning only the bias term and a scalar gate on the relation type when computing messages. Like us, they introduce additional features $\delta \in \Delta$ where $\Delta = \{\leftarrow, \rightarrow\}$ to represent edge direction and use different weight matrices for different directions. This parameterisation is not sufficiently expressive to model highly multirelational knowledge bases.

## 3.3 Methods

As we have mentioned, relational link prediction denotes automatic likelihood evaluation for *(subject, relation, object)*-triples thought to be missing from a knowledge base. Formally, scores $f(v_i, r, v_j)$ must be assigned to new edges $(v_i, r, v_j)$ that do not – but *could* – belong to $\mathcal{E}$. Most existing approaches to link prediction (for example, tensor and neural factorization methods such as Bordes et al. (2013), Lin et al. (2015), Yang et al. (2015), and Trouillon et al. (2016)) address the problem by embedding every entity $v_i \in \mathcal{V}$ as a real-valued vector

$e_i$, and subsequently computing scores through a learned function $f(e_i, r, e_j)$. We build on this approach by *contextualizing* entity representations through a graph neural network.

## 3.3.1 Relational Graph Convolutional Networks

To model entity neighbourhoods, we extend the Graph Convolutional Networks[4] (GCNs) of Kipf and Welling (2017) to directed, labeled graphs. Recall that in GCNs, the message-propagation and message-aggregation steps of the graph neural network are jointly defined as a single matrix multiplication step:

$$h_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}_i} \frac{1}{c_i} W^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)}\right), \tag{3.1}$$

where $h_i^{(l)} \in \mathbb{R}^{d^{(l)}}$ is the hidden state of node $v_i$ in the $l$-th layer of the neural network, with $d^{(l)}$ being the dimensionality of this layer's representations. $\sigma(\cdot)$ denotes a non-linear activation function such as the linear rectifier $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$ and $c_i$ is a problem-specific normalization constant that can either be learned or chosen in advance. $\mathcal{N}_i$ denotes the set of neighbour indices of node $i$.

To extend this model to multirelational, directed graphs we instantiate separate weights and biases for every relation at every layer, along with separate normalization constants. We furthermore introduce an additional feature $\delta \in \Delta$ where $\Delta = \{\leftarrow, \rightarrow\}$ to represent the direction of edges. To send messages along both directions of edges, we add for every edge $(u, r, v, \delta) \in \mathcal{R}$ an additional edge $(u, r, v, \hat{\delta})$ to the graph where $\hat{\delta}$ is the opposite direction of $\delta$. This allows the propagation and aggregation steps to take into account the type and direction of edges. We refer to our extension as a *Relational Graph Convolutional Network*, or *R-GCN*. Formally:

**3.3.1.** DEFINITION. Let $h_i^l \in \mathbb{R}^{d^{(l)}}$ be the embedding of the vertex $v_i$ at the $l$-th layer. Then, the R-GCN step $f_l : \mathcal{V} \times \mathbb{R}^{d^{(l+1)}} \rightarrow \mathcal{V} \times \mathbb{R}^{d^{(l)}}$ at layer $l$ takes the form:

$$h_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}_i^r} \sum_{r\in\mathcal{R}} \sum_{\delta\in\Delta} \frac{1}{c_{i,r,\delta}} W_{r,\delta}^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)}\right)$$

where $\mathcal{N}_i^r$ denotes the neighbour indices of node $i$ connected to $i$ by edges of type $r$.

[4] See Section 2.2 for a thorough discussion of the graph neural network framework, including Graph Convolutional Networks.

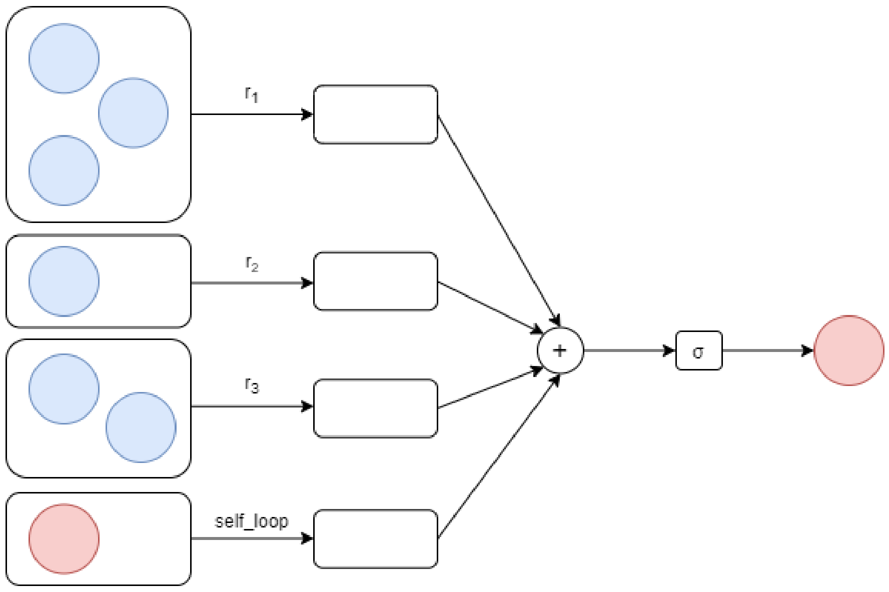


Figure 3.2: R-GCN per-layer update for a single graph node (in light red). Activations from neighboring nodes (light blue) are first summarized individually by relation type, and subsequently summarized in total. A similar operation is performed on the previous embedding of the node in question via a self-loop. The resulting sum is passed through an activation function (e.g. the ReLU) to construct a new embedding. This per-node update is computed in parallel with shared parameters across the whole graph.

By stacking layers of R-GCN updates, we can compute learned, contextualised embeddings of vertices for potentially any graph – see Figure 3.2.

We note that the R-GCN framework as expressed in Definition 3.3.1 can be efficiently implemented through the GCN-formulation discussed in Section 2.2. This is done by computing a separate propagation step for each relation type. Since the total number of edges (and thus the total number of messages over all propagation steps) is unchanged, the time complexity remains $O(|\mathcal{E}|)$. However, as an update for every vertex must be stored for every relation type, the space complexity grows to $O(|\mathcal{R}||\mathcal{V}|)$. This can be circumvented using an IMN-formulation, resulting in a space complexity of $O(|\mathcal{V}| + |\mathcal{E}|)$. If the number of relation types is large, this may be preferable.

Real-world knowledge bases are massive in size, which can make the cost of executing an R-GCN step prohibitively expensive. To alleviate this problem, we introduce a sparsity constraint for our relational graph convolutional networks reliant on restricting weights to only take non-zero values in block matrices surrounding the diagonal; that is, each $W_r^{(l)}$ is represented as a block-diagonal matrix $\text{diag}(Q_{1r}^{(l)}, \ldots, Q_{Br}^{(l)})$. In addition to reducing the computational complexity of the GNN, this parameterisation is also more compact and hence easier to learn. It can be seen as a parallel to the reduction in the decoder from full matrices in RESCAL (Nickel et al., 2011) to diagonal matrices in DistMult (Yang et al., 2015).

**3.3.2.** DEFINITION. Let $Q_{1r}^{(l)}, \ldots, Q_{Br}^{(l)}$ be a set of square matrices such that $Q_{br}^{(l)} \in \mathbb{R}^{(d^{(l+1)}/B) \times (d^{(l)}/B)}$. Then, the weight matrix $W_r^{(l)}$ corresponding to relation $r$ at layer $l$ is expressed as:

$$W_r^{(l)} = \bigoplus_{b=1}^{B} Q_{br}^{(l)}.$$

where $\bigoplus$ represents the direct sum.

Block decomposition can be seen as a sparsity constraint on the weight matrices for each relation type. The block decomposition structure implements an intuition that latent features can be grouped into sets of variables, which are more tightly coupled within groups than across groups. This reduces the number of parameters needed to learn for highly multirelational data.

Our full graph encoder consists of $L$ layers of R-GCN as defined in 3.3.1, with the output of each layer being the input to the next. We compute the initial representation $h_i^0$ of each vertex $v_i$ by passing a unique one-hot vector for each node in the graph (if no other features are present) through a single linear transformation. While we only consider the featureless approach in this work and in Schlichtkrull et al. (2018), we note that it was shown in Kipf and Welling (2017) that GCNs can also benefit from pre-defined feature vectors (e.g. a bag-of-words description of a document associated with a specific node).

To improve the training of our R-GCN encoder, we further extend the model with residual connections (He et al., 2016) between each stacked layer. Preliminary experiments suggested that this extension – although not necessary to reach strong performance – stabilised training and facilitated faster learning.

### 3.3.2 Link Prediction

We model the link prediction problem through a graph auto-encoder model, comprised of an entity encoder and a scoring function (decoder). The encoder maps each entity $v_i \in \mathcal{V}$ to a real-valued vector $e_i \in \mathbb{R}^d$. The decoder reconstructs edges of the graph relying on the vertex representations; in other words, it scores *(subject, relation, object)*-triples through a function $s : \mathbb{R}^d \times \mathcal{R} \times \mathbb{R}^d \to \mathbb{R}$. As mentioned, existing approaches from the literature often consist purely of a scoring function $s' : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \to \mathbb{R}$; these can be considered a special case of our framework where the encoder is replaced with a learnable embedding matrix.

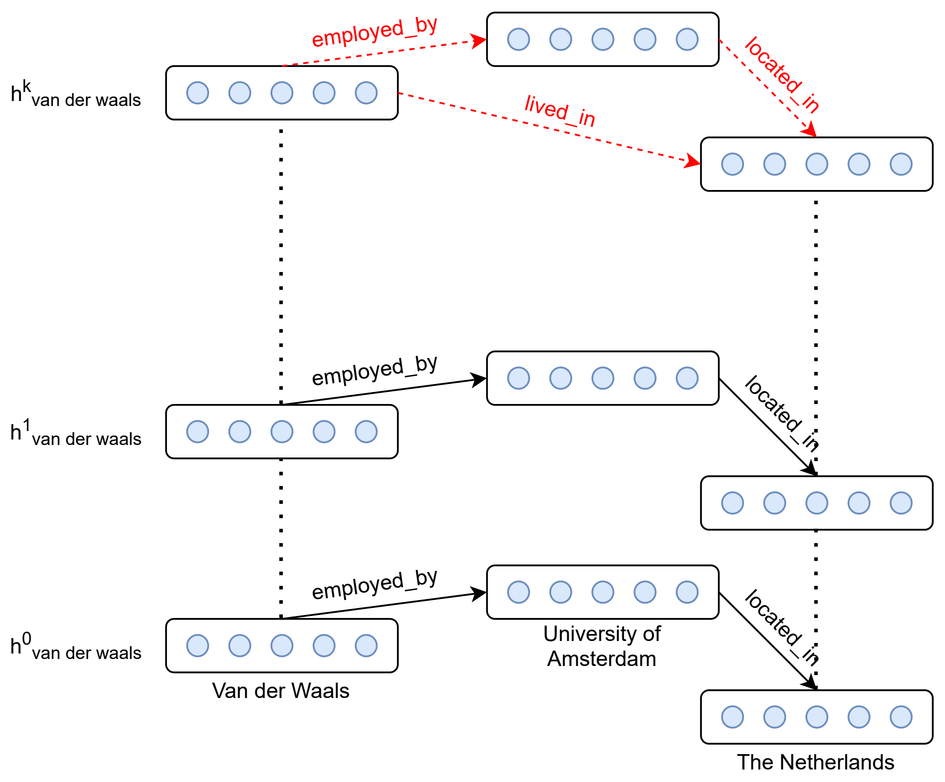


Figure 3.3: The structure of our link prediction model. Contextualized embeddings for each entity are constructed through successive R-GCN layers, and the top-layer embeddings $h^{(k)}$ are subsequently used in a factorization model to score edges. During training, the encoder sees only a partial graph (shown in black). The full graph (shown in red) is then reconstructed.

Our work distinguishes itself by parameterising the encoder in order to contextualise each entity embedding. Instead of a single, real-valued vector $e_i$ for every $v_i \in \mathcal{V}$ optimized directly in training, we compute representations through an R-GCN encoder with $e_i = h_i^{(k)}$, similar to the graph auto-encoder model introduced in Kipf and Welling (2016) for unlabeled, undirected graphs. During training, we drop edges from the underlying graph and as such learn to *reconstruct* the neighbourhood around each vertex. The structure of this model can be seen in Figure 3.3.

In our experiments, we use the DistMult factorization (Yang et al., 2015) as the scoring function. DistMult is known to perform well on standard link prediction benchmarks when used on its own, and the strict sparsity constraint in the decoder is beneficial for reducing the complexity of the model. Following Definition 3.2.1, every relation $r \in \mathcal{R}$ is associated with a diagonal matrix $R_r \in \mathbb{R}^{d \times d}$, and every vertex $v \in \mathcal{V}$ is associated with an embedding $e_v$. We simply let $e_v$ be the top-layer encoding of $v$ produced by our R-GCN. As such, a triple $(s, r, o)$ is scored as

$$f(s, r, o) = h_s^{(k)\intercal} R_r h_o^{(k)} . \tag{3.2}$$

As in previous work on factorization (Yang et al., 2015; Trouillon et al., 2016), we estimate the model through negative sampling. For each observed example, we sample $\omega = 10$ negative ones. We sample by randomly corrupting either the subject or the object of each positive example. We optimize for a cross-entropy loss to push the model to score observable triples higher than the negative ones:

$$\mathcal{L} = -\frac{1}{(1+\omega)|\hat{\mathcal{E}}|} \sum_{(s,r,o,y)\in\mathcal{T}} y \log \sigma\big(f(s,r,o)\big) + (1-y)\log\big(1-\sigma\big(f(s,r,o)\big)\big) , \tag{3.3}$$

where $\mathcal{T}$ is the total set of real and corrupted triples, $\sigma$ is the logistic sigmoid function, and $y$ is an indicator set to $y = 1$ for positive triples and $y = 0$ for negative ones.

To train our model in the aforementioned graph auto-encoder setup, we begin by sampling a large neighbourhood of $K$ to encode – this is necessary to train our model on larger datasets. In our experiments, we choose neighbourhoods of size $K = 40.000$. We perform the sampling by an iterative process. We randomly select an edge, and subsequently continue to pick edges adjacent to any that has already been selected. We stop this process when no more edges can be selected, or when $K$ edges have been selected. We include these edges in the decoder with a probability $\delta = 1.0$, and in the encoder with a probability $\tau = 0.5$ (selected on development data). This ensures that the model encounters examples at training time where edges that are present in the decoder as positive samples are not in the encoder. We then pick relations to use for Equation 3.3 from among those chosen for the decoder. Using this *edge dropout* objective makes our training regime similar to that of denoising autoencoders (Vincent et al., 2008). As we discuss in Section 3.4, the use of edge dropout is crucial for training our model.

## 3.4 Experiments

To evaluate the performance of our model and judge the effectiveness of our R-GCN encoder, we carry out experiments on several datasets from the literature. We focus primarily on the challenging FB15k-237 dataset introduced by Toutanova and Chen (2015). This is a fragment of the Freebase (Bollacker et al., 2008) knowledge base constructed for link prediction.

The train, validation, and test split in FB15k-237 are subsets of those introduced for the FB15k-dataset by Bordes et al. (2013). For completeness, we also include results for that dataset, as well as for the WordNet fragment also introduced in the same paper (WN18). However, as demonstrated by Toutanova and Chen (2015), those datasets are not realistic representations of real-world inference tasks as many test triplets $(s, r, o)$ have an inverse triplet $(o, r^{-1}, s)$ that

| Dataset | WN18 | FB15K | FB15k-237 |
|---|---|---|---|
| Entities | 40,943 | 14,951 | 14,541 |
| Relations | 18 | 1,345 | 237 |
| Train edges | 141,442 | 483,142 | 272,115 |
| Val. edges | 5,000 | 50,000 | 17,535 |
| Test edges | 5,000 | 59,071 | 20,466 |

Table 3.1: Number of entities and relations (i.e. relation types) along with the number of edges per split for the two datasets.

appears in the training data; this allows data leakage from the training to the test set. While FB15k and WN18 do allow some reasoning about the relative performance of models, we can only fully investigate whether our model yields improvements for more complex inferences by evaluating R-GCN on the FB15k-237 dataset. Characteristics of the three datasets are summarized in Table 3.1.

As is standard in the literature, we provide results using two evaluation metrics: *mean reciprocal rank* (MRR) and *Hits at n* (H@n). Both are ranking metrics that compare the rank of positive triplets with respect to potential negative triplets under a local closed-world assumption (Bordes et al., 2013). That is, they compute scores for all possible triplets, and compute the rank of the triples in the test dataset relative to all other triples.

Formally, for every test triplet $t = f(s, r, o)$, we compute the score $x$ of that triplet. Then, we find the rank of $t$ with respect to the set of scores $A = \{f(x, r, o) | x \in \mathcal{V}\}$, and compute evaluation measurements on that basis. We repeat this process as a separate measurement for the inverse set of scores $B = \{f(x, r, o) | o \in \mathcal{V}\}$. Following Bordes et al. (2013), there are two separate ways of computing these methods – the *raw* setting and the *filtered* setting. In the raw setting, all potential triplets are included in $A$ and $B$; in the filtered setting, triplets that already occur in the train, test, or validation set are excluded. We report both filtered and raw MRR (with filtered MRR typically considered more reliable), and filtered Hits at 1, 3, and 10.

### 3.4.1 Evaluation

We achieved the best results with a normalization constant defined as $c_{i,r} = c_i = \sum_r |\mathcal{N}_i^r|$ — in other words, a predefined rather than a learned normalization factor, and applied across relation types. This *relation-wise normalisation* can be seen as a middle ground between mean- and sum-pooling. We found block decomposition to perform best with blocks of dimension $5 \times 5$. For FB15k-237 where complex reasoning about neighbourhoods is more beneficial, we found a two-layer R-GCN encoder with 500-dimensional embeddings to work best. For FB15k and WN18, we achieved the best performance using a single layer and

200-dimensional embeddings.

We regularise the encoder through edge dropout as discussed in the previous section, applied before normalisation. We use an edge dropout probability of $\tau = 0.5$. In our experiments, the use of edge dropouts was crucial to obtain good performance. This is because of how the edges used for embedding vertices in the encoder and those used as positive samples in the decoder are selected at training time. If these are chosen from the same set, the model can learn to make positive predictions only for this set. If these are chosen from entirely distinct sets, the model can learn to make *negative* predictions only for this set. As the graph used by the encoder is the training set, models without edge dropout tend to learn solutions that make either entirely positive or entirely negative predictions for the test set. Using $\tau = 0.5$ strikes a good balance – half the edges used as positive examples in each batch occur in the encoder, the other half do not.

We furthermore apply regular dropouts to the self-loop connection with a probability 0.2, to encourage reliance on long-distance propagation. Following previous best practices for DistMult (Yang et al., 2015), we apply $l_2$ regularization to the decoder with a penalty of 0.01. We use the Adam optimizer (Kingma & Ba, 2015) with a learning rate of 0.01. For the baseline and the other factorisations, we found the parameters from Trouillon et al. (2016) – apart from the dimensionality on FB15k-237 – to work best. Owing to the large storage cost necessary for our graph autoencoder objective, we used CPUs rather than GPUs to train the model. The full training process took approximately 72 hours, with early stopping after 10.000 updates.

The natural baseline for our experiments is to directly optimise the DistMult decoder (Yang et al., 2015) without any encoder. This corresponds to a version of our model using fixed entity embeddings, rather than the R-GCN. We furthermore compare against several other factorization models from the literature: TransE (Bordes et al., 2013), HolE (Nickel et al., 2016), ComplEx (Trouillon et al., 2016), and CP (Hitchcock, 1927). At the time our initial paper Schlichtkrull et al. (2018) was published, TransE was the best performing model on FB15k-237, and ComplEx and HolE were the best performing models on FB15k and WN18, respectively.

TransE is a simple system wherein triples are modelled as the sum of the embeddings of the subject, the relation, and the object. ComplEx generalises DistMult to the complex domain (more appropriate for modelling asymmetric relations), while HolE replaces the vector-matrix product with circular correlation. CP is a classic benchmark that is often included for the link prediction task. It is important to note that improvements in the *decoders* employed by these three are orthogonal to our improved *encoder*, and combination systems could be constructed.

We also include results for an ensemble of our model and the DistMult baseline, observing that the strengths of the two systems may be complementary. We

| | MRR | | Hits @ | | |
|---|---|---|---|---|---|
| Model | Raw | Filtered | 1 | 3 | 10 |
| LinkFeat | - | 0.063 | - | - | 0.087 |
| DistMult | 0.100 | 0.191 | 0.106 | 0.207 | 0.376 |
| R-GCN (basis) | 0.117 | 0.194 | 0.102 | 0.208 | 0.399 |
| R-GCN (block) | **0.158** | 0.248 | **0.153** | 0.258 | 0.414 |
| R-GCN+ | 0.156 | **0.249** | 0.151 | **0.264** | **0.417** |
| CP | 0.080 | 0.182 | 0.101 | 0.197 | 0.357 |
| TransE | 0.144 | 0.233 | 0.147 | 0.263 | 0.398 |
| HolE | 0.124 | 0.222 | 0.133 | 0.253 | 0.391 |
| ComplEx | 0.109 | 0.201 | 0.112 | 0.213 | 0.388 |

Table 3.2: Results on FB15k-237, a reduced version of FB15k with infrequent and near-duplicate relations removed. CP, TransE, and ComplEx were evaluated using the code published for Trouillon et al. (2016).

define

$$f(s,r,t)_{\text{R-GCN+}} = \alpha f(s,r,t)_{\text{R-GCN}} + (1-\alpha) f(s,r,t)_{\text{DistMult}}. \tag{3.4}$$

The mixing ratio $\alpha$ was set to 0.4, chosen empirically on development data. For the ensemble components in R-GCN+, we employ half-sized models to ensure a fair comparison.

In Table 3.2, we show the results for FB15k-237.[5] The block diagonal decomposition of our R-GCN model outperforms the DistMult baseline by a large margin of 29.8%, highlighting the importance of a good encoder model. The R-GCN model also performs favourably against other factorisations, despite the comparatively low result for the DistMult decoder when used without an encoder. TransE performs surprisingly well, outperforming both CP, DistMult, and ComplEx, suggesting that the use of a task-specific decoder choice for R-GCN could, in future work, lead to improved performance.

In Schlichtkrull et al. (2018), we also introduced an alternative to block decomposition, which performed better for node classification. Instead of representing relation-specific weights through $|\mathcal{R}|$ block-diagonal sparsified matrices, that scheme uses weights composed from learned weighted sums over $\mathcal{B}$ basis matrices such that $\mathcal{B} << |\mathcal{R}|$. In Table 3.2, we compare our block decomposition against

[5] Our numbers are not directly comparable to those reported by Toutanova and Chen (2015), as they use pruning both for training and testing (see their sections 3.3.1 and 4.2). Their pruning scheme includes a relation-specific hyperparameter $t$. The values for this hyperparameter are not given in the paper. As this pruning scheme is therefore not fully specified, and as their code is not available, we could not fully replicate their set-up.

| | FB15k | | | | | WN18 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | MRR | | Hits @ | | | MRR | | Hits @ | | |
| Model | Raw | Filtered | 1 | 3 | 10 | Raw | Filtered | 1 | 3 | 10 |
| LinkFeat | - | 0.779 | - | - | 0.804 | - | 0.938 | - | - | 0.939 |
| DistMult | 0.248 | 0.634 | 0.522 | 0.718 | 0.814 | 0.526 | 0.813 | 0.701 | 0.921 | 0.943 |
| R-GCN | 0.251 | 0.651 | 0.541 | 0.736 | 0.825 | 0.553 | 0.814 | 0.686 | 0.928 | 0.955 |
| R-GCN+ | **0.262** | **0.696** | **0.601** | **0.760** | **0.842** | 0.561 | 0.819 | 0.697 | 0.929 | **0.964** |
| CP* | 0.152 | 0.326 | 0.219 | 0.376 | 0.532 | 0.075 | 0.058 | 0.049 | 0.080 | 0.125 |
| TransE* | 0.221 | 0.380 | 0.231 | 0.472 | 0.641 | 0.335 | 0.454 | 0.089 | 0.823 | 0.934 |
| HolE** | 0.232 | 0.524 | 0.402 | 0.613 | 0.739 | **0.616** | 0.938 | 0.930 | **0.945** | 0.949 |
| ComplEx* | 0.242 | 0.692 | 0.599 | 0.759 | 0.840 | 0.587 | **0.941** | **0.936** | **0.945** | 0.947 |

Table 3.3: Performance for our models and baselines on the full FB15k and WN18 datasets. Results marked (*) taken from Trouillon et al. (2016). Results marked (**) taken from Nickel et al. (2016).

this *basis decomposition* scheme. For link prediction on FB15k-237, block decomposition performs much better – basis decomposition fails to outperform the TransE baseline, while block decomposition achieved results that outperformed all contemporary models. Interestingly, this difference does not exist for the full FB15k (see Table 3.3); we theorise that block decomposition is helpful primarily when more complex relation-specific reasoning is required.

Table 3.2 includes results for the simple LinkFeat baseline introduced in Toutanova and Chen (2015). In that model, pairs of entities are modelled through $|\mathcal{R}|$ indicator features representing whether an edge of type $r$ connects those entities. Then, a simple classifier is used to score triples on that basis. For FB15k-237, where no data leakage through explicit inverse relations occurs, this baseline does not perform well; however, for FB15k and WN18 this baseline outperforms all existing methods. Despite this, these datasets continue to be used in the literature. For that reason, we include in Table 3.3 results for our methods on FB15k and WN18, along with results for the LinkFeat baseline.

As can be seen, LinkFeat performs better than any other method on FB15k, and the best alternative does not significantly outperform LinkFeat on WN18. With that said, R-GCN does present a competitive alternative to other methods on these datasets as well, especially in the R-GCN+ setting where an ensemble of a DistMult and an R-GCN model is employed. The complementary nature of R-GCN and DistMult is as such very clear in this setting.

### 3.4.2 Analysis

While we have seen that R-GCN and pure DistMult in some settings perform very different and highly complementary types of inference, it is unclear where this disparity originates from. A notable strength of R-GCN is that our relation-

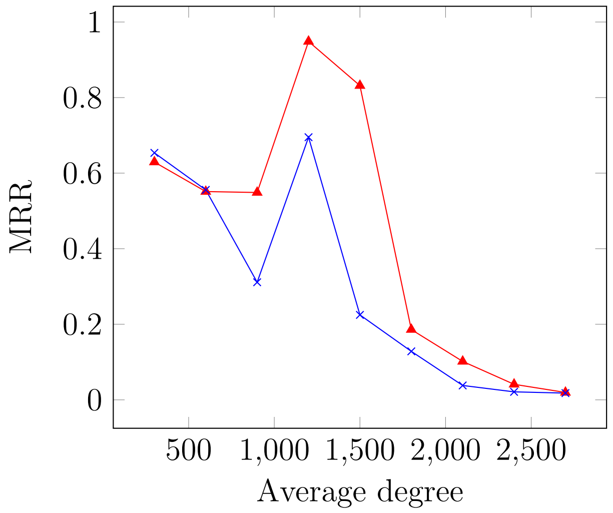


Figure 3.4: Mean reciprocal rank (MRR) for R-GCN (—▲—) and DistMult (—×—) on the FB15k-237 validation data as a function of the node degree (average of subject and object).

wise normalisation scheme equalises the importance of different relations. This can be helpful for high-degree vertices, where the contributions of a few edges with infrequent relation types can become overshadowed by the contributions of many edges with frequent relation types. In Figure 3.4, we plot the performance of DistMult and R-GCN as functions of the average degree of the subject and object in each triple. As can be seen, DistMult performs slightly better for low-degree vertices, and R-GCN performs better for high-degree vertices. This is intuitively reasonable, as DistMult must compress entire neighbourhoods into singular vectors through gradient updates with equal priority given to each relation type. In contrast, R-GCN can instead learn a graph convolutional function to do so, normalising over relation types separately so that low-frequency types are not excluded.

An interesting question is what exactly the R-GCN encodes when it constructs vertex embeddings. In Figure 3.5 we plot the relative performance of DistMult and R-GCN on FB15k-237, separated according to the minimum distance in the encoded graph (e.g. the training set) between the source and target vertices. We note that Toutanova and Chen (2015) removed from FB15k-237 all node pairs connected by a single train edge; as such, only self-connections and longer distance connections are present.

As the figure shows, the R-GCN model performs better when the subject and object are close in the encoder graph, and the DistMult model performs better when the subject and object are far apart. One explanation may be that the explicit encoding R-GCN performs allows the model to reason about the *relation* between the two vertices, whereas it does not rely as heavily on implicit similar-

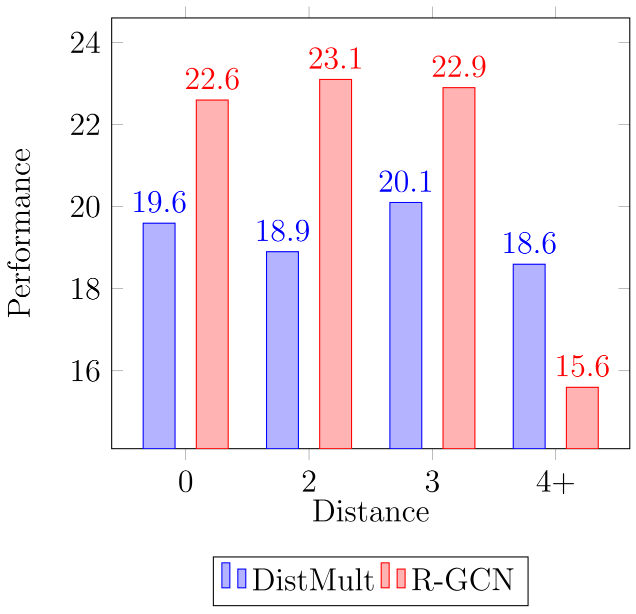


Figure 3.5: Performance on FB15k-237 of our R-GCN model compared to the baseline DistMult model, split according to minimum distance between source and target vertex.

ities between vertex neighbourhoods. Since the R-GCN model only employs two layers of convolution, long-distance relations cannot be captured through explicit reasoning, and such relations are therefore harder to model.

This observation raises the question of why the R-GCN/DistMult ensemble leads to performance improvements on FB15k, but not on FB15k-237. Toutanova and Chen (2015) suggested that two separate types of one-hop relations were removed from FB15k during their process of creating FB15k-237: Implicit inverse relations that result in data leakage, and legitimate one-hop connections. One simple explanation for this observation could be that the decoder-only model is better at exploiting the data leakage.

There is a large disparity between the frequency with which common and rare relations appear in the dataset. This can present a problem for models, as it is difficult to learn inference patterns for rare relations. Our R-GCN encoder allows for an increased capacity for reasoning about the relationship between vertices, and could as such alleviate this problem. In Figure 3.6, we plot the performance of our model versus the DistMult baseline as a function of the rank of the relation being queried. For DistMult, performance degrades when modelling rarer relations; R-GCN mitigates this issue, and in fact performs slightly *better* for rarer relations.

Taken together, R-GCN as such excels for high-degree vertices, to encode relations between entities that are close (but not directly connected), and to memorise predictive information for low-frequency relations. R-GCN learns to

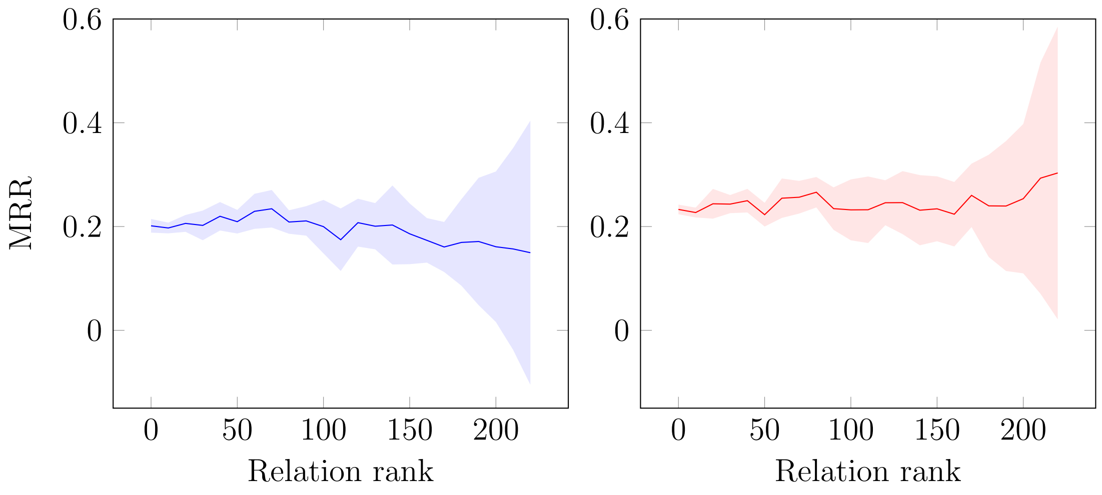


Figure 3.6: Mean reciprocal rank (MRR) for DistMult (left) and R-GCN (right) on the FB15k-237 dataset as a function of the rank of the relation. While performance degrades for DistMult on rare relations, R-GCN performs well across the entire dataset. Performance smoothed over bags of size 10.

collect and summarise information from the local neighbourhood around a vertex, and to filter relevant and irrelevant information when the neighbourhood is large. Because the model relies strongly on information and relations present in the local neighbourhood, it struggles to capture relations exactly when the two vertices being encoded are further apart than the few hops allowed by two layers of R-GCN; further layers of R-GCN may alleviate this problem, but as with other neural networks deeper models can be harder to train.

## 3.5 Subsequent Work

Since our initial publication on R-GCNs (Schlichtkrull et al., 2018), variants have found application to many different domains. Closest to our work is Berg et al. (2017), wherein an encoder based on our R-GCN was directly combined with a bilinear scoring function to model and complete recommendation graphs. Many subsequent applications similar to our technique combine GNN encoders with other neural network components for different tasks, whereas the entire setup combining GNN encoders and factorisation decoders for relational link prediction has not seen the same degree of application. In NLP, models similar to R-GCN have been employed to encode syntactic graphs for machine translation (Bastings et al., 2017), extending the concurrent GNN for semantic role labelling by Marcheggiani and Titov (2017). GNNs have also been applied to model coreference- and cooccurrence graphs for a variety of question answering tasks (Sun et al., 2018; De Cao et al., 2019). GNNs combining relation-based mod-

| Model | MRR | Hits@1 | Hits@3 | Hits@10 |
|---|---|---|---|---|
| Ours | 0.249 | 0.151 | 0.264 | 0.417 |
| ConvE (Dettmers et al., 2018) | 0.325 | 0.237 | 0.356 | 0.501 |
| RotatE (Sun et al., 2019d) | 0.297 | 0.205 | 0.328 | 0.480 |
| TuckER (Balazevic et al., 2019) | 0.358 | 0.266 | 0.394 | 0.544 |
| A2N (Bansal et al., 2019) | 0.317 | 0.232 | 0.348 | 0.486 |
| DAGN (Wang et al., 2020b) | 0.369 | 0.275 | 0.409 | 0.563 |

Table 3.4: Results on FB15k-237 for several works subsequent to our publication (Schlichtkrull et al., 2018).

elling with gates (Li et al., 2016b) or attention functions (Veličković et al., 2018) have become increasingly popular as a means of representing graph-structured data, with the use of relation-specific weights for gates being a common approach.

Relational link prediction has also seen strong improvements since our publication of Schlichtkrull et al. (2018), both on the decoder-side and the encoder-side. In Table 3.4, we report the results of several subsequent papers. Convolutional models as introduced in Dettmers et al. (2018) have proven especially performant, with stable results across many datasets. Recent papers have produced even stronger decoders, such as TuckER (Balazevic et al., 2019). Fundamentally, these improvements are orthogonal to the results we have presented – our R-GCN encoder could be combined with any decoder. However, newer models often rely on very high-dimensional embeddings. This can represent a challenge for GNN-based models, because the message passing step has a memory footprint of $O(|\mathcal{V}|+|\mathcal{E}|)$ (see Definition 2.2.2) – at least for the popular IMN-implementations. That is, the memory footprint is proportional to the number of edges. Factorisation algorithms only require memory proportional to the number of vertices, which is often much smaller.

Nevertheless, several recent papers have demonstrated strong improvements from the use of encoders. A2N (Bansal et al., 2019) employs a single layer of attention-based aggregation to construct a contextualized embedding for the source vertex $s$, given a query $(s, r, ?)$. Their approach is comparable to a single layer of Veličković et al.'s (2018) attention-based GNN, although crucially their attention function is conditional on the target relation $r$ in addition to the graph structure. Wang et al. (2020b) introduce DAGN, a more complex graph model which constructs attention-based embeddings of vertices not only conditional on the neighbour set, but also on other vertices in the graph. As in our work, A2N and DAGN pair the graph encoder with DistMult acting as a decoder.

## 3.6 Conclusion

We have introduced Relational Graph Convolutional Network (R-GCN) encoders for link prediction, a family of graph neural networks which incorporate relation type and direction when computing entity representations. By encoding neighbourhoods with R-GCN, we have demonstrated strong performance improvements on relational link prediction, specifically for the FB15k-237 dataset. Along with our model, we have introduced a block decomposition scheme, which allows our technique to be applied also to large graphs with many different relation types.

Reflecting on the two research questions we set out to explore in this chapter, it is clear that benefits can be gained from explicitly modelling entities with graph neural networks. Performance gains primarily occur for complex inference problems, and when very specific information is needed to make predictions – high-degree vertices and infrequent relations, where the necessary information can be drowned out. The use of a GNN encoder does come with a drawback for vertices too far apart, because the GNN struggles to capture their relationship. Within the neighbourhood defined by the $k$ layers of convolution around either vertex, the performance gain from GNNs is much stronger.

The experiments we have carried out here demonstrate the value of explicitly encoding vertices in semantic graphs – especially for tasks requiring infrequently appearing information, or for gathering rich information from few-hop neighbourhoods. The strong results for modelling relational data with GNNs are promising for our intention to apply these models for question answering over knowledge graphs, which we explore in Chapter 4.

Chapter 4

# Graph Neural Networks for Factoid Question Answering

## 4.1 Introduction

One of the most fundamental problems in NLP is automatic answering of natural language questions. This problem is known as *question answering* (QA). Several distinct forms of QA have been studied in the literature, separated by the nature of the knowledge source, which is used to answer questions. Answering questions on the basis of knowledge bases represents an important setting, because the immense amounts of information curated in KBs are often highly relevant. Furthermore, because of the need to formulate queries against KBs in machine language, this information is not easily accessible to humans. Accurately retrieving and processing information from KBs to answer natural language queries opens up a host of practical applications, from virtual assistants to search algorithms.

In the previous chapter, we introduced a model for relational link prediction in knowledge bases. In that setting, we aimed to build a model which, given a potential triplet $(s, r, t)$, scored the likelihood of that triplet based on the KB. Such methods can answer queries formulated as triplets $(s, r, \cdot)$ missing a target vertex $v_t \in \mathcal{V}$ by ranking potential target vertices. It is a small step from a formalized query $(s, r, \cdot)$ to a natural language query $Q$, expressing a request for information concerning an entity $v_s \in \mathcal{V}$, to be answered with another entity $v_t \in \mathcal{V}$. Conceptually, it is tempting to think of the difference between the two settings as simply the substitution of a relational for a textual query. Were that the case, we could directly apply the model from Chapter 3 to factoid question answering.

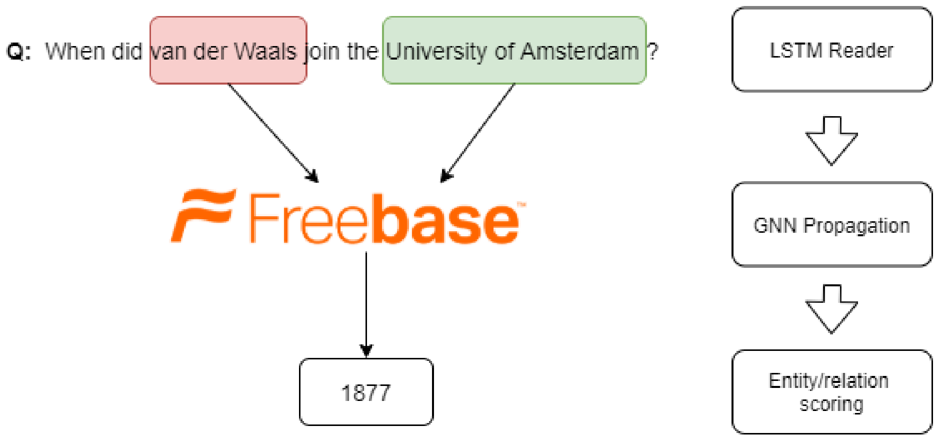


Figure 4.1: In factoid question answering, natural language questions are answered by interfacing with a knowledge base and selecting answers from among the entities. Our models use LSTMs to encode the questions, GNN-propagation to encode the relations between candidate answers and question entities, and finally entity- or relation-based scoring to choose answers.

Of course, many real-world questions are too complex for this simplistic scheme. They involve multiple entities (e.g. *"which years did Michael work at the University of Amsterdam?"*), and a single entity is often not sufficient to give a full answer; a system may need to answer with multiple entities, or with a textual answer that does not involve entities at all. The much more expansive setting where the query and the answer take the form of *sets* of entities $v_s \subset \mathcal{V}$ and $v_t \subset \mathcal{V}$ is a well-studied problem known as *factoid question answering*.

The most common strategy for tackling factoid question answering is *semantic parsing*; that is, the translation of the question into a formal query language, which is then used to access the knowledge base. Semantic parsing has led to impressive advances in factoid question answering, for example in Berant and Liang (2014), Reddy et al. (2014), and Berant and Liang (2015). However, semantic parsing is fundamentally limited by the need to either annotate datasets of valid formal queries or rely on automatically generated denotations. Furthermore, earlier models[1] have struggled to incorporate the knowledge base as contextual information for parsing decisions. Modelling the knowledge base with graph neural networks, as such, represents a promising alternative.

In this chapter, we attempt to apply graph neural networks to the factoid question answering problem. There are two immediately obvious strategies we could choose for applying GNNs to factoid QA. The simplest would be to model

[1] In the time since we conducted our research, methods have been introduced to incorporate relations from the knowledge source as features in semantic parsing models. Notably, Bogin et al. (2019) and Wang et al. (2020a) demonstrated significant gains on the Spider dataset using respectively GNNs and a relation-aware attention mechanism to model the knowledge source.

vertices as we do in Chapter 3, and classify vertices as answers on that basis. That is, we could use a GNN to obtain a contextualized vector representation of every vertex. Then, we could make an individual decision for every vertex based on the corresponding vector as to whether that vertex constitutes (part of) an answer to the query. A slightly different strategy would be to instead use the GNN to obtain contextualized vector representations of *paths* in the graph. We could then return as an answer every vertex that is reachable from mentioned entities in the query, following paths with a specific sequence of relation types. These paths can be seen as analogous to the central path discussed in Yih et al. (2015). We refer to these two strategies as *entity-based* and *relation-based* modeling.

We introduce several models for both the entity-based and the relation-based setting, and investigate their respective strengths and differences. We seek to answer the following research questions:

1. *Can graph neural network encoders yield benefits for question answering by modelling entities and relations in the knowledge base?*

2. *Is entity-based or relation-based modelling the more suitable strategy, and which graph neural network models work best in either setting?*

The models we introduce combine LSTMs for understanding the question with GNNs for understanding the knowledge base. We experiment with two different problem formulations, showing that different GNNs lead to strong performance in each setting. We furthermore explore several choices enabling GNNs to operate in large knowledge bases, including the use of gated messages. To this end, we introduce a novel GNN variant relying on gates to choose which edges to rely on and $L_1$ regularization to induce sparsity.

Our primary contribution in this chapter is the aforementioned exploration of different models and GNN encoders for factoid QA. We validate our models on the WebQuestions dataset. The experiments we present here were carried out in 2018, making ours one of the earliest attempts to address the problem. Since then, several concurrent and subsequent works have been published. In addition to our experiments, we also include a discussion of these in Section 4.5, using the nowadays more popular WebQuestionsSP dataset.

## 4.2 Background

In this chapter, we address the problem of factoid question answering. Formally, we are given a set of examples each consisting of a natural language question $Q = w_1, w_2, ..., w_n$, as well as a knowledge graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{R})$.[2] The task is to predict for each example a set of answer entities $V_a \subset \mathcal{V}$. We furthermore have

[2] Recall from Section 3.2 that a knowledge graph is defined as a directed, labeled multigraph $G = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ with vertices (entities) $v_i \in \mathcal{V}$ and labeled edges (relations) $(v_i, r, v_j) \in \mathcal{E}$,

access to a set of question entities $V_q \subset \mathcal{V}$, linking the question to the knowledge graph. Each question entity $v \in V_q$ is associated with a mention, i.e. a span of words $m_v = w_i^v, ..., w_j^v$ in the question.

Factoid question answering has been the subject of much research in the form of semantic parsing, where natural language questions are converted into logical forms to be executed against a knowledge base (Zelle & Mooney, 1996; Zettlemoyer & Collins, 2005; Wong & Mooney, 2007; Kwiatkowksi et al., 2010). Despite significant efforts to scale these methods to realistic knowledge bases (Berant et al., 2013), they remain hard to apply due to the difficulty of obtaining labelled training examples with complete parses (Berant & Liang, 2014). For this reason, recent years have seen substantial amounts of research dedicated to developing factoid question answering models that do not rely on labelled semantic parses at training time.

A significant problem is that existing labelled data only covers a small fraction of the possible questions a system might be tasked with answering; however, paraphrasing can be used to breach the gap. This was the topic of Berant and Liang (2014), which paraphrased queries to ease parsing. A similar approach was taken in Kwiatkowski et al. (2013), where malformed or under-specified queries resulting from parsing questions too far from the training domain were "paraphrased" into executable queries.

This lack of supervised training pairs was also addressed in Berant et al. (2013), wherein the authors proposed a strategy for learning from denotations (that is, answer sets in the knowledge base). In their approach, candidate logical forms for the question are sampled in the question entities' neighbourhoods. Then, a simple log-linear model scores these candidate forms. At training time, the correctness of a generated logical statement can be evaluated simply by executing it against the knowledge base to see whether the returned entities match the answer set. As such, only question-answer pairs are necessary for learning. This approach was further extended in Reddy et al. (2014), where the problem was reformulated as matching between a CCG-parse of the sentence and the candidate graphs.

Selecting a *best path* to match the question from the knowledge graph is a strategy that reappears in several highly-performant systems. Yih et al. (2015), the state-of-the-art model for the dataset we experiment on, begins from a best path (or *core inferential chain* in their terminology) and grows a semantic parse from that. Xu et al. (2016) selects a best path directly for inference, extending their classifier with additional textual information. Similarly, Dong et al. (2015) score all paths from the question entities to candidate answers with a CNN, selecting a best possible path in place of selecting an answer. This idea of making

where $r \in \mathcal{R}$ is a relation type. As mentioned, this definition does not fully encompass real-world knowledge bases, because it only covers binary relations. We address this deficiency in Section 4.3.1.

inference based on a path (rather than individually per vertex) features in our model as *relation-based prediction.*

Another approach that addresses the problem is slot-filling (Bast & Haussmann, 2015; Yao, 2015). The question entities are matched to one of several logical templates, and the template is populated using the question entities. This limits the parsing problem's scope by reducing it to a search for the best match among the patterns observed at training time, simplifying the problem.

## 4.3 Methods

As we have mentioned, factoid QA can be formalised as follows. Given is a natural language question $Q = w_1, w_2, ..., w_n$ as well as a knowledge base $G = (\mathcal{V}, \mathcal{E}, \mathcal{R})$, linked through a set question entities $V_q \subset \mathcal{V}$ each with a corresponding mention span of words $m_v = w_i^v, ..., w_j^v$. Then, the task is to produce answers in the form of sets of entities. In this chapter, we experiment with different graph neural network models (see our review in Section 2.2) for encoding knowledge bases in factoid QA. Common to our various models is a general structure wherein we encode the questions into vectors. We then use those vectors as initial embeddings for the question entities in a GNN, letting the information in the question propagate through the knowledge base via message passing. Finally, we answer the question based on the top-layer vertex representations.

To encode the question, we begin by vectorizing each word using either GloVE (Pennington et al., 2014) or ELMo (Peters et al., 2018). For every word $w_i$, we obtain a corresponding embedding $f(w_i)$. To indicate the various entities present in the question, we concatenate an indicator variable $m_i$ to each word embedding such that $m_i = 1$ if $w_i \in m_v$ for some question entity $v$, and $m_i = 0$ otherwise. We then process the concatenated question word embeddings through a bidirectional LSTM (Hochreiter & Schmidhuber, 1997), obtaining high-level embeddings $f^*(w_i)$ of each word $w_i$. Finally, we construct an embedding $s_v$ for each question entity $v$ by concatenating the high-level embeddings of the first and the last word in the corresponding mention span; that is, $s_v = [f^*(w_i^v), f^*(w_j^v)]$.

### 4.3.1 Graph Neural Networks for Freebase

In the experiments we perform in this chapter, the knowledge base used to answer questions is Freebase (Bollacker et al., 2008). To address the particularities of that knowledge base, we develop two specialised GNN variants: A simple extension of the R-GCN from Chapter 3, and a more expressive gated GNN.

The vertices of Freebase consists of *entity* vertices $e_1, ..., e_n$ and *compound value type* vertices $c_1, ..., c_n$. Compound value types represent complex, n-ary relations. Formally, given an instantiation of a complex relation $r$ with $m$ arguments, e.g. $r(e_1, ..., e_m)$, each argument $e_k$ is connected to a compound value type

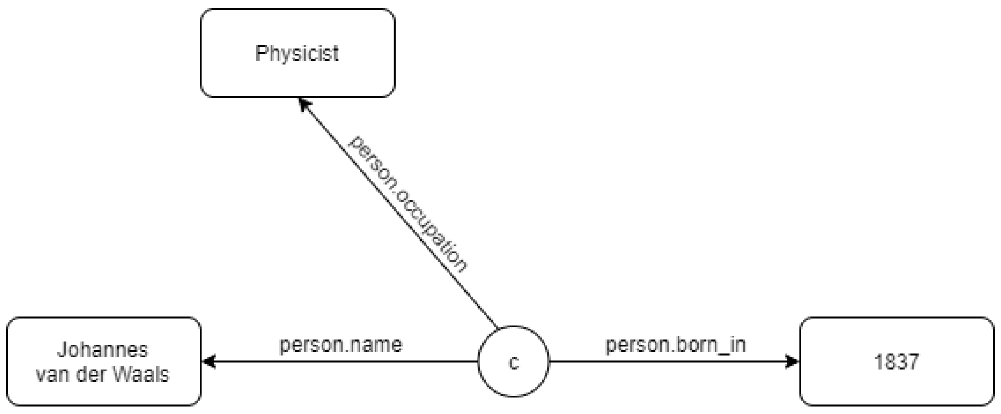


Figure 4.2: An example of a ternary relation expressed through a compound value type $c$. A GNN operating in Freebase must be designed to incorporate propagation through such relations.

$c$ through an edge of type $r.k$. An example can be seen in Figure 4.2.

Binary Freebase relations are treated simply as arcs between the two arguments. This distinction can be problematic for GNN models, as hops through compound value types require twice as many GNN layers. As such, because depth influences the difficulty of learning, naive models tend to overprioritise binary relations. To prevent this, we first modify the binary relations into artificial compound value types by replacing every relation $(e_i, r, e_j)$ between two entity vertices with two relations $(e_i, r.1, c)$ and $(c, r.2, e_j)$ such that $c$ is a novel compound value type vertex – see also the example in Figure 4.3.

With 1.9 billion entries, it is difficult to tractably learn to encode the full Freebase with a GNN. Thankfully, factoid question answering does not necessitate encoding the *full* Freebase. Because the task is to retrieve information related to the question entities, the answer is likely to be found within their immediate neighbourhood. As such, for a question $Q$ with question entities $V_q \subset \mathcal{V}$, we use only the vertices and edges within $k$ hops of any question entity $v \in V_q$. For our experiments, we chose $1 \leq k \leq 3$ empirically. We refer to this subgraph as $G_Q$.

To initialize our GNN models, we begin by defining the initial vertex embeddings $H^{(0)}$ as discussed in Section 2.2. As mentioned, we inject information from the question into the GNN through these. We use the mention embeddings obtained through the LSTM, $s_v$. All other vertices are initialized with zeros. That is:

$$h_v^{(0)} = \begin{cases} s_v & \text{if } v \in V_q. \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

Past work on factoid question answering (Reddy et al., 2014) has shown that most questions can be answered with a small fragment of the knowledge base – a few edges connecting the question to the answers, and a few auxiliary edges as extra evidence. As such, many of the edges in the knowledge base will be

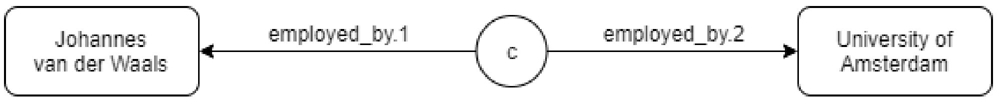


Figure 4.3: An example of a binary relation transformed to be expressed through a compound value type. We preprocess the Freebase subgraphs our models operate in such that all direct relations between entities become compound value types.

superfluous when attempting to answer a given question. Indeed, the information they encode will often be unrelated to the question. In addition to the R-GCN introduced in Chapter 3, we therefore experiment with a *gated* model that can learn to ignore edges.

We refer to this model as Gated Relational Graph Neural Network (GR-GNN). We note that our GR-GNN differs from popular prior work on gated GNNs (Li et al., 2016b; Beck et al., 2018), as the primary gating mechanism is applied to messages during propagation rather than to vertex embeddings after aggregation. This *edge-wise* gating strategy was previously applied for semantic role labeling in Marcheggiani and Titov (2017). As opposed to R-GCN, we do not use individual weight matrices for each relation type. Instead, we embed relation types and direction as additional feature vectors when constructing messages. We furthermore condition each message on a question embedding $S$, obtained as the output of the final state of the LSTM. While increasing performance, this formulation prevents us from implementing the model as a GCN (see Section 2.2.1).

**4.3.1.** Definition. A Gated Relational Graph Neural Network (GR-GNN) is a graph neural network parameterized through a set of layerwise weights $W_g^{(l)} \in \mathbb{R}^{3d \times 1}$ and $W_m^{(l)} \in \mathbb{R}^{3d \times d}$ at layer $l$, as well as an embedding function $E : \mathcal{R} \times \{\leftarrow, \rightarrow\} \rightarrow \mathbb{R}^d$.

We express the GR-GNN through a gated instantiation of the message function (see Definition 2.2.1). While we experiment with various aggregation functions, sum-pooling is the most natural (and the best performing) for GR-GNN.

**4.3.2.** Definition. Let $h_v^{(l)} \in \mathbb{R}^d$ be the embedding of the vertex $v$ at layer $l$, and $r$ and $\delta$ be the relation type and direction of an edge $e = (u, r, v, \delta)$. Furthermore, let $E_{(r,\delta)} \in \mathbb{R}^d$ be the embedding vector of $r$ and $\delta$. Then, the GR-GNN message $m_e^{(l)}$ associated with $e$ is defined as:

$$m_e^{(l)} = \sigma(W_g^{(l)}[h_u^{(l)}, E_{(r,\delta)}, S]) \cdot \text{ReLU}(W_m^{(l)}[h_u^{(l)}, E_{(r,\delta)}, S])$$

where $\sigma$ is the sigmoid function.

As we also did for R-GCN, we connect layers with residual connections (He et al., 2016). Similarly to our previous experience with R-GCN, such connections did not directly improve performance, but stabilised training and strongly decreased the number of training updates necessary to reach an optimum. GR-GNN can be efficiently implemented as an incidence matrix network as described in Section 2.2.2, keeping the complexity low.

As mentioned, a benefit of using gates is the ability for the model to choose not to use superfluous edges. It can be beneficial to directly encourage the model to close gates. We introduce an $L_1$-penalty term for doing so. As we later show (see Section 4.4), this penalty is crucial for reaching high performance.

**4.3.3.** Definition. Let $h_v^{(l)} \in \mathbb{R}^d$ be the embedding of the vertex $v$ at layer $l$, and $r$ and $\delta$ be the relation type and direction of an edge $e = (u, r, v, \delta)$. Furthermore, let $E_{(r,\delta)} \in \mathbb{R}^d$ be the embedding vector of $r$ and $\delta$. Then the $L_1$-penalty of the GR-GNN at layer is defined as:

$$L_1^{(l)} = \sum_{(u,r,v,\delta)\in\mathcal{E}} \sigma(W_g^{(l)}[h_u^{(l)}, E_{(r,\delta)}, S])$$

where $\sigma$ is the sigmoid function.

## 4.3.2 Entity-based models

An intuitively simple strategy for choosing a set of answer vertices is to score each vertex individually, and then choose all vertices with scores above a certain threshold. This parallels our approach to modelling relational link prediction in Chapter 3, where we learned a score for each vertex in order to select the *most* suitable answer. Here, however, a question can have potentially many answers.

**4.3.4.** Definition. Entity-based factoid question answering is the task of computing, given a question $Q$ with an associated graph $G_Q = \mathcal{V}, \mathcal{E}, \mathcal{R}$, the probability $p_v(a|Q, G_Q)$ for every vertex $v \in \mathcal{V}$ of $v$ being an answer to $Q$.

We address this by learning individual binary probability distributions $p_v(a|Q, G_Q)$ for each vertex $v$, representing the probability of $v$ being a member of the answer set $V_a$. We compute this distribution on the basis of the GNN-encoding of $v$ at the top layer of the GNN, $h_v^{top}$, as well as the sentence embedding $S$:

$$p_v(a|Q, G_Q) = \sigma(f([h_v^{top}, S])) \tag{4.2}$$

where $f$ is an MLP. We then predict an answer set containing every entity with a probability above some threshold chosen on validation data.

As we did for R-GCN (see Equation 3.3), we learn the vertex scores using negative sampling. For every question $Q$, we sample $\omega = 10$ potential answers $V_a^-$ from the immediate neighbourhood $G_Q$ in addition to the set of actual answers $V_a$. We combine these into a set of evaluated vertices $V_l = V_a \cup V_a^-$. We introduce an indicator $y_v$ such that $y_v = 1$ iff $v \in V_a$. For each question $Q$, we then compute a loss term as:

$$\mathcal{L} = -\frac{1}{(1+\omega)} \sum_{v \in V_l} y_v \log p_v(a|Q, G_Q) + (1 - y_v) \log\big(1 - p_v(a|Q, G_Q)\big) \qquad (4.3)$$

### 4.3.3 Relation-based models

In Yih et al. (2015), strong performance was achieved through the assumption that there exists a *core inferential chain* connecting the question to the answer in at most two hops (or one hop, if the middle vertex is not a CVT). This core inferential chain connects most potential answers to the question. A decent strategy is accordingly to assume that *all* such entities and *only* those entities should be returned as answers.

Following this idea, we experiment with a model that performs predictions based on relations rather than individual entities. We predict a triple containing a mentioned entity and two relation types (e.g. *person.name* and *person.occupation*, in Figure 4.2), and return all entities from the database matching this pattern as answers. We enumerate for each question sentence $Q$ a set of triples on the form $C_Q = (m, r_1, r_2)$, each consisting of a mention $m$ and a hyperpath $r_1, r_2$ (e.g. a compound pass through an artificial or natural CVT-vertex). We then choose from among these to answer the question.

**4.3.5.** Definition. Relation-based factoid question answering is the task of computing, given a question $Q$ with mentions $M$ and an associated graph $G_Q = \mathcal{V}, \mathcal{E}, \mathcal{R}$, the probability distribution $p(C = C_Q|Q, G_Q)$. Here, $C_Q = (m, r_1, r_2)$ is a precomputed optimal hyperpath from a mentioned entity $m \in M$ to an answer to $Q$.

We encode each possible mention $m$ simply by reusing the embedding $s_m$ obtained through the LSTM. To encode the relation types $r_1$ and $r_2$, we employ the GNN. We encode $r_1$ as the mean embedding $e(r_1)$ of all entities $v_1$ such that $(m, r_1, v_1) \in \mathcal{E}$, and $r_2$ as the mean embedding $e(r_2)$ of all entities $v_2$ such that $(m, r_1, k) \in \mathcal{E}$ and $(k, r_2, v_2) \in \mathcal{E}$ for some entity $k$. We then learn a probability distribution over these triples:

$$p(C = C_Q|Q, G_Q) = \sigma(f([s_m, e(r_1), e(r_2), S])) \qquad (4.4)$$

where $\sigma$ is a the softmax function and $f$ is an MLP.

| Method | Mean F1 |
|---|---|
| Berant and Liang (2014) | 39.9 |
| Dong et al. (2015) | 40.8 |
| Reddy et al. (2014) | 41.3 |
| Yao (2015) | 44.3 |
| Bast and Haussmann (2015) | 49.4 |
| Berant and Liang (2015) | 49.7 |
| Yih et al. (2015) | **52.5** |
| Reddy et al. (2016) | 50.3 |
| Xu et al. (2016) | 47.1 |
| Entity-based baseline | 34.1 |
| Entity-based GR-GNN | 45.6 |
| Relation-based baseline | 44.8 |
| Relation-based GR-GNN | 47.4 |

Table 4.1: Results from the literature plus results for some of our various systems. First section contains comparable background results, the second section contains our results with entity-prediction models, and the third section our results with relation-prediction models. For both types of models, we also include results for a baseline that leaves out the GNN; equivalent to a zero-layer GR-GNN model.

We again employ negative sampling, selecting a set $C^-$ of negative triples for each question. We then use a categorical cross entropy loss over this set to learn to score triples:

$$\mathcal{L}_r = \sum_{C \in C^- \cup \{C_Q\}} y(C_Q) \log p(C = C_Q | Q, G_Q) \tag{4.5}$$

## 4.4 Experiments

We test our models on the WebQuestions dataset introduced in Berant et al. (2013). WebQuestions is a dataset of 3,778 training and 2,032 test questions to be answered using Freebase as a knowledge source. The questions were not originally tagged with named entities linked directly to the knowledge base, but it has become standard practise to reuse the excellent entity linking provided in Yih et al. (2015); we do so as well. For comparability, we use the same Freebase dump as in Reddy et al. (2016). We report results in terms of mean per-example F1-score, as in prior work. That is, for each example, we compute the F1-score between the generated answer set and the gold answer set. We then compute the dataset-level mean as the evaluation metric.

| Model | Mean F1 |
|---|---|
| Full entity-based model | 45.6 |
| - Gates | 41.3 |
| - L1 | 45.0 |
| - ELMo | 44.7 |

Table 4.2: Ablation test for our entity-based model. We report results disabling the gates from the GNN-model introduced in Section 4.3, as well as the gated GNN with and without L1-regularization. We furthermore report results using GloVe-embeddings rather than contextualized ELMo-embeddings for modeling the question.

In Table 4.1, we compare the respective performance of our strongest entity-prediction and relation-prediction models to several works from the literature. As discussed in Section 4.3, our formulation relies on ELMo-embeddings for encoding the question, followed by several layers of graph neural networks. In both cases, two-layer models yielded the highest performance. For the entity-based model, the gated relational GNN introduced in Definitions 4.3.1 and 4.3.2 with L1-regularization performed the best; for the relation-based model, we achieved the highest performance with a simple R-GCN as discussed in Chapter 3. We scaled the L1-term by a factor 0.01, chosen as a hyperparameter on a held-out portion of the training set (as the original WebQuestions dataset does not contain a development set). Our models were trained using Adam (Kingma & Ba, 2015) with a learning rate of 0.001. Our models were trained on a single Titan X GPU, requiring approximately 48 hours of computation for the entity-based model, and 36 hours for the relation-based model.

With a mean F1-score of 47.4, our strongest model does not quite reach the state-of-the-art semantic parsing result of Yih et al. (2015). Nevertheless, our GNN-based approach represents a competitive alternative architecture. We achieve the best results with the relation-based strategy, gaining 1.8 points F1-score compared to the best entity-based system. In conjunction with the strong performance of Yih et al. (2015) – which as discussed relies on a "core inferential chain" not dissimilar from our relation-based approach – this suggests that the idea of answering through such a structure is a good way of modelling the problem.

Interestingly, the best-performing graph encoder for the relation-based strategy (e.g. R-GCN) is much simpler than for the entity-based strategy (e.g. GR-GNN) – there are no gates, and no L1-regularization. In Tables 4.2 and 4.3, we perform ablation tests to further investigate which components are necessary to achieve high performance.

For the entity-based model, Table 4.2 contains results dropping respectively the gates or the sparsity-inducing L1 regularization. We also include results

| Model | Mean F1 |
|---|---|
| Full relation-based model | 47.4 |
| GR-GNN | 41.8 |
| GR-GNN + L1 | 43.1 |
| - ELMo | 46.5 |

Table 4.3: Ablation test for our relation-based model. As the gated GNN in this setting actually hampers performance, we compare the simple R-GCN used in Table 4.1 to versions using the GR-GNN-model introduced in Section 4.3 with and without L1-regularization. We furthermore report results using GloVe-embeddings rather than contextualized ELMo-embeddings for modeling the question.

replacing the contextualized ELMo-embeddings in the encoder with GloVe embeddings (Pennington et al., 2014). We find that gates are highly beneficial for the model, especially when accompanied by the sparsity-inducing penalty. Allowing the model to choose selectively which edges to send messages along as such appears to be crucial. We furthermore find that the ELMo-embeddings add a small improvement to the model.

For the relation-based model, the use of gates does not seem to give the same benefits. Indeed, as the ablation tests in Table 4.3 show, the simpler R-GCN without gates actually performs much better than the more complex gated version. As with the entity-based model, L1-regularization does seem to improve the performance of the gated model – this further supports our hypothesis that sparsity is advantageous for GNNs operating on knowledge bases. However, the simpler model here yields even greater performance. Together with the marginal benefits of the second layer (see Figure 4.4), this may suggest that the relation-based approach for this dataset requires less complexity to model. Another explanation could be that the GR-GNN gates in the entity-based model capture essentially the same information as the hyperpath for the relation-based model – a central path from the mentioned entities to the answer.

An important question is the number of GNN-layers required to model factoid question answering. For both the entity-based and the relation-based models, we achieved the best performance (e.g. the performance reported in Table 4.1) with two layers; and there was only a 1.2-point performance difference between the two models. Interestingly, this difference increases drastically with fewer layers. In Figure 4.4 we plot the performance of the two models as a function of the number of GNN-layers. While both models benefit from additional layers (up to a point), the gain from adding multiple layers in the entity-based model is massive. This is not entirely unsurprising in the case of the 0-layer baseline, as the entity-based model without layers cannot situate unseen entities in relation to the question entities at all. Therefore, it is only for entities that occur during the training set

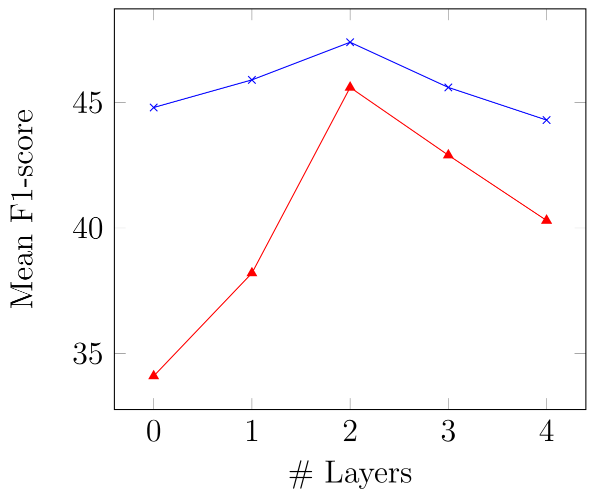


Figure 4.4: Mean F1-score for our best-performing entity-based (—▲—) and relation-based (—×—) factoid question answering models on the WebQuestions test data as a function of the number of GNN-layers used.

| Model | Mean F1 |
|---|---|
| Entity-based, mean-pooling | 42.3 |
| Entity-based, relation-wise normalization | 43.7 |
| Entity-based, sum-pooling | 45.6 |
| Relation-based, mean-pooling | 46.1 |
| Relation-based, relation-wise normalization | 47.4 |
| Relation-based, sum-pooling | 44.6 |

Table 4.4: Performance of our entity-based (top) and relation-based (bottom) models with mean-pooling, relation-wise normalization, and sum-pooling.

that performance above random can be expected. However, the gap between the one- and two-layer cases can only be explained by the model successfully using two-hop information.

In Chapter 3, we introduced a relation-wise normalization scheme for our R-GCN model. For the experiments in that chapter, relation-wise normalization was necessary to achieve strong performance. Since our work on R-GCN, subsequent research (Xu et al., 2019) has demonstrated that all forms of mean- and max-pooling for GNNs limit the expressivity of the GNN; instead, their results suggest the use of sum-pooling. Sum-pooling is also problematic, as it may result in embeddings of high-degree vertices having substantial variance. We mentioned in Section 3.4 that our relation-wise normalization can be seen as a middle ground between mean- and sum-pooling. However, it still fundamentally suffers from the same issues as mean-pooling when it comes to expressivity.

In Table 4.4, we perform experiments with the various models introduced in this chapter under different normalization schemes. As previously, we use the GR-GNN for the entity-based approach and the R-GCN for the relation-based approach. We found it necessary to use relation-wise normalization for the relation-based model, and do away entirely with normalization (e.g. use sum-pooling) for the entity-based model. Interestingly, when training an entity-based model *without* gates, sum-pooling and relation-wise normalization perform on par (42.3% and 42.4%, respectively). This suggests that normalization and gating can perform the same variance-reducing function, ensuring that the embeddings of high-degree vertices do not explode. This is intuitively reasonable, as messages with closed gates do not contribute to the variance of the target vertex' embedding. In contrast, messages with open gates make the same contribution as messages with *no* gates.

## 4.5 Subsequent Work

Since our experiments, factoid QA has seen several important developments. Focus has shifted from WebQuestions to the WebQuestionsSP subset, developed in Yih et al. (2016). WebQuestionsSP provides a SPARQL-query exactly answering each question, and filters out any questions not answerable through SPARQL. This stronger supervisory setting can be used to achieve higher performance for the STAGG-model (Yih et al., 2015), which held the state-of-the-art result on the original WebQuestions.

Concurrently with our efforts to develop GNN-models for factoid QA, Sun et al. (2018) introduced GRAFT-Net, a GNN-based system for WebQuestionsSP. GRAFT-Net was developed to combine information from knowledge graphs and text, and also functions well in settings where a large part of the knowledge graph has been replaced with textual documents. GRAFT-Net was further developed into PullNet (Sun et al., 2019b). In Table 4.5, we compare several subsequent results, including GRAFT-Net and PullNet. We also include results with the REINFORCE-based Neural Symbolic Machines (Liang et al., 2017). Unfortunately, two disparate evaluation metrics are in use for WebQuestionsSP, and several papers report only one of the two – this complicates comparisons.

In our experiments in this chapter, inducing sparsity through gates yielded strong improvements for the entity-based approach. An unfortunate consequence of relying on gates to reduce the number of edges is that the training complexity is unaffected. As the Freebase fragments we rely on here are large, this slows down training – indeed, our best-performing models require several days of GPU-accelerated training. GRAFT-Net and PullNet address this problem by pruning the graph *before* training through two different strategies. GRAFT-Net prunes the graph with Personalized PageRank (Haveliwala, 2003), retaining only the top $E$ entities reachable from the question entities $V_q$. PullNet takes this one step

| Model | Hits@1 | Mean F1 |
|---|---|---|
| STAGG | - | 66.8 |
| GRAFT-Net only KB | 66.7 | 62.4 |
| GRAFT-Net | 67.8 | 60.4 |
| PullNet | 68.1 | - |
| NSM | - | 69.0 |
| T5 (text + tables + KB) | 79.1 | - |
| STAGG (w/ strong supervision) | - | 71.7 |

Table 4.5: Results on WebQuestionsSP subsequent to our work. We include results for STAGG (Yih et al., 2015), the state-of-the-art model for WebQuestions at the time of our experiments. We then report results for GRAFT-Net with and without extra textual data (Sun et al., 2018), the followup work PullNet (Sun et al., 2019b), as well as Neural Symbolic Machines (Liang et al., 2017). We furthermore include results from T5 pretrained for QA over text, tables, and knowledge bases (Oğuz et al., 2020), as well as a version of STAGG trained with annotated semantic parse labels (Yih et al., 2016).

further, *learning* which entities to prune. PullNet iteratively grows the graph through a process where a frontier is scored using a GNN. The highest-valued entities are added to a set of retained entities, and a new frontier is constructed. The model is trained to find entities on the shortest path between the question and the answer. In both cases, this pruning increases performance and decreases complexity drastically.

GRAFT-Net and PullNet both rely on GNNs similar to our R-GCN to encode graph fragments. In the case of GRAFT-Net, the encoder is further extended to also include a term conditioned on a PageRank score. The line of work wherein GNNs are used to model graph fragments for factoid QA was also explored in Sorokin and Gurevych (2018) using a novel version of WebQuestionsSP relying on WikiData rather than Freebase as the knowledge source. Their experiments relied on a gated GNN, which demonstrated strong performance in comparison to baselines.

A promising line of related research involves transforming other question answering problems into factoid QA by constructing artificial graphs between potential answer candidates (De Cao et al., 2019; Tu et al., 2019). This allows the use of GNN models to very effectively solve multi-hop question answering problems. The graphs used are typically coreference- and cooccurrence-graphs, or knowledge base fragments used to enrich textual documents.

Very recently, Oğuz et al. (2020) have demonstrated that large pretrained language models (see our discussion in Section 2.3.1) can be applied to linearised representations of knowledge graph fragments, enabling unified models to learn from and perform question answering over many sources. Combining text, tables, and knowledge base triples, strong performance is reached on many datasets, including state of the art results on WebQuestions and WebQuestionsSP. Their results indicate that the inclusion of structured knowledge sources – that is, tables and knowledge bases – can complement textual knowledge sources and lead to improvements also for more traditional textual question answering scenarios.

## 4.6 Conclusion

We have introduced two GNN-based models for factoid question answering, relying either on choosing individual vertices or choosing a best path to answer natural language questions. Although our models do not surpass the strong semantic parsing baseline of Yih et al. (2015), we do demonstrate performance close to contemporary models from the literature. Our models rely on a gated graph neural network with a sparsity-inducing $L_1$-penalty. We demonstrate that the addition of the penalty term improves performance, showing the importance of reducing the number of edges used for decisionmaking in GNN-based QA systems. This finding is further backed by the results of e.g. Sun et al. (2018), Sun et al. (2019b), wherein pruning through either PageRank or a secondary model is used to achieve high performance.

Regarding the research questions we set out to answer, we see clear benefits from the application of GNNs to factoid question answering, although the end result of our experiments is a promising alternative with performance close to or similar to existing techniques rather than an improvement. Relation-based modelling seems to lead to superior results, perhaps due to the implicit modelling of a "central path" as also used by Yih et al. (2015). Interestingly, concurrent and subsequent work (Sun et al., 2018; Sun et al., 2019b) relies on entity-based modelling, suggesting an avenue for future improvement on those techniques.

In the process of developing GNNs for factoid QA, we encountered several obstacles. A major problem was the difficulty in understanding why certain variants outperformed others, and which aspects of the problem a given solution addressed. To a large extent, this was due to the nature of GNNs as highly complex, nonlinear, "black-box"-models (Belinkov & Glass, 2019). In Chapter 5, we focus on developing a technique for understanding and interpreting the predictions of trained GNN models, inspired in part by our experiences with the development of models for factoid QA.

Chapter 5

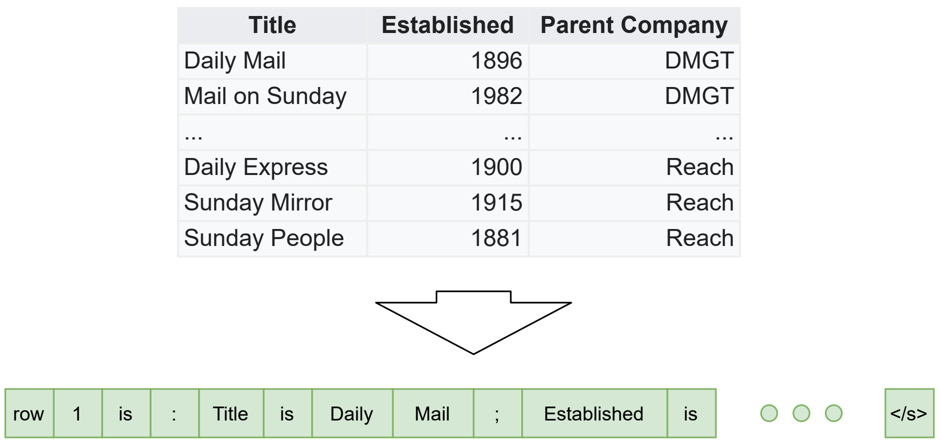
# Open Fact Verification Over Tables

## 5.1 Introduction

Verifying whether statements are true or false based on a trusted knowledge source is a fundamental NLP problem, closely related to question answering. This task has important applications to automated fact checking (Vlachos & Riedel, 2014) and other tasks in computational journalism (Cohen et al., 2011; Flew et al., 2012). The problem has been extensively investigated under different conditions including entailment and natural language inference (Dagan et al., 2005; Bowman et al., 2015) as well as claim verification (Vlachos & Riedel, 2014; Alhindi et al., 2018; Thorne & Vlachos, 2018). Despite this, little attention has been devoted to the setting where the trusted body of evidence is structured – that is, where it consists of tabular or graph-structured data.

In previous chapters, we have focused on modelling structured data as graphs with GNNs. For many forms of such data – knowledge bases, social networks, geographical maps – modelling the structure as a relational graph and encoding that through a GNN is a very natural approach. Indeed, relational graphs are a very general approach to modelling structure, applicable to most situations. In certain special cases, the elements of the structure can be assigned a linear order. This enables the use of an alternative modelling scheme commonly referred to as *linearisation*.

Linearisation works by defining a heuristic for mapping from structure to sequence (see the example in Figure 5.1). Then, one of the many high-performing sequence models from the literature can be applied. In the case of tables, it is simple to order the elements of the structure linearly: rows can be enumerated, columns can be enumerated, and cells accordingly can also be enumerated. Recent work on table modelling has demonstrated very strong results using pretrained



| Title | Established | Parent Company |
|---|---|---|
| Daily Mail | 1896 | DMGT |
| Mail on Sunday | 1982 | DMGT |
| ... | ... | ... |
| Daily Express | 1900 | Reach |
| Sunday Mirror | 1915 | Reach |
| Sunday People | 1881 | Reach |

Figure 5.1: An example of *linearisation*. Structured data (a table, in this case) is converted to an unstructured form through a predefined heuristic. Subsequently, high-performing models for unstructured data can be applied to model the information.

transformer-based[1] models to encode linearised tables (Chen et al., 2020; Yin et al., 2020). In this chapter we explore the use of such models as an alternative to GNNs for encoding structure, focusing specifically on fact verification for the open domain.

Traditionally, fact verification and question answering are defined in two different settings: the *open* and the *closed* domain (Chen et al., 2017). In the closed domain, each statement or question is associated with a single document (or a small collection of documents). This document is then guaranteed to contain information to verify the statement or answer the question. In the open domain, the system must first retrieve appropriate evidence from a larger knowledge source. While both settings are highly beneficial for the development of models, only the open domain accurately reflects the necessities of real-world fact verification tasks.

Recently, two large-scale datasets were introduced for claim verification over structured tables (Chen et al., 2020; Gupta et al., 2020). These represent the first large-scale structured fact verification environments available to the community; unfortunately, they are both designed as closed-domain tasks. At first glance, one might expect closed-domain datasets to not be easily transferable to the open setting – potentially, many queries could be too context-dependent to be answerable without reference to a specific table. For example, if the claim in Figure 5.2 had been *"some companies own multiple newspapers"* or even *"some companies occur multiple times as owners"*, retrieval would not be possible. Thankfully, in the dataset we use (Chen et al., 2020), the majority of the claims are sufficiently

[1] See also Section 2.3, where we discuss how transformers can be seen as a special case of graph neural networks operating in the fully connected graph.

context-independent to be understood without knowing which table they were constructed with reference to. The existence of a reference table for every claim does, however, present a unique opportunity to exploit an additional signal at training time, which we seek to benefit from.

In this chapter and in Schlichtkrull et al. (2020a), we investigate fact verification over tables in the open setting. We take inspiration from similar work on unstructured data (Chen et al., 2017; Nie et al., 2019; Karpukhin et al., 2020; Lewis et al., 2020), proposing a two-step model which combines non-parametric, retrieval-based memory with a neural reader operating on top of retrieved tables. Drawing on preliminary work in open question answering over tables (Sun et al., 2016), we perform retrieval based on simple heuristic modelling of individual table cells. While highly performant in terms of hits-at-N, a more sophisticated model is ultimately needed to bridge the gap between the top-rated retrieved table and the best table located in the $k$ top-rated tables. For that reason, we combine this simple heuristic with a joint reranking-and-verification model, performing fusion of $k$ evidence documents in the verification component. This is similar to the approach also suggested for question answering in Izacard and Grave (2020). We attempt to answer the following research questions:

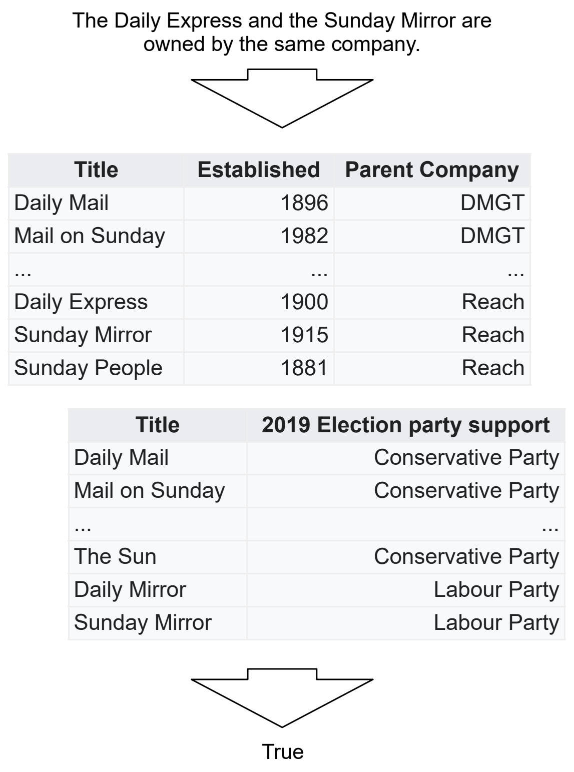
1. *Can neural models be applied to efficiently encode tables for querying in open-domain fact verification tasks?*

2. *How can the existence of large-scale closed-domain datasets be exploited to improve open-domain performance?*

3. *Will our models still yield performance improvements if applied to query against a Wikipedia-scale dataset rather than the much smaller* $16,573$ *evidence tables available in TabFact?*

We introduce a novel model for fact verification over multiple tables, consisting of a RoBERTa (Liu et al., 2019) encoder for linearised tables, and a cross-attention module fusing evidence documents. Combined with a heuristic TF-IDF retriever, our system demonstrates strong open-domain performance exceeding even the previous closed-domain state of the art (outside of Eisenschlos et al. (2020), which includes intermediary pretraining on additional synthetic data). When combined with an oracle retriever for a single table, our encoder sets a new closed-domain state of the art. We furthermore propose two strategies with corresponding loss functions for exploiting closed-domain fact verification datasets in the open setting. These respectively either increase verification accuracy, or enable the model to better identify when the TF-IDF retriever has failed to find appropriate evidence. We finally present additional results using a highly realistic setting where tables are retrieved from the full Wikipedia dump.

Our primary contribution in this chapter is the introduction of effectively the first system for open-domain table fact verification. The two loss functions



The Daily Express and the Sunday Mirror are owned by the same company.

| Title | Established | Parent Company |
|---|---|---|
| Daily Mail | 1896 | DMGT |
| Mail on Sunday | 1982 | DMGT |
| ... | ... | ... |
| Daily Express | 1900 | Reach |
| Sunday Mirror | 1915 | Reach |
| Sunday People | 1881 | Reach |

| Title | 2019 Election party support |
|---|---|
| Daily Mail | Conservative Party |
| Mail on Sunday | Conservative Party |
| ... | ... |
| The Sun | Conservative Party |
| Daily Mirror | Labour Party |
| Sunday Mirror | Labour Party |

True

Figure 5.2: Example claim to be evaluated against two retrieved tables. Named entities represent a strong baseline for retrieval, but ultimately a more complex model is required to distinguish highly similar tables.

represent a secondary contribution enabling our high performance. We test our model on TabFact dataset (Chen et al., 2020), reporting results with retrieval from the $16,573$ TabFact tables. We also compare against a version of our model using an oracle to retrieve the closed-domain gold table. Additionally, we test our system with retrieval from approximately 3 million tables automatically gathered from Wikipedia. We furthermore include a thorough ablation study.

## 5.2 Background

Fact verification is the task of automatically determining the truth value of a given claim $q$ based on some trusted body of evidence. This evidence could be a single document in the closed-domain setting, or a large collection of documents in the open setting. For our purposes, each document is a table $t$, and the collection of documents in the open setting is as such a database of tables $T$. This is the setting studied in the closed domain by the recently introduced large-scale dataset of Chen et al. (2020).

Automated fact verification as an NLP problem was first studied in Vlachos and Riedel (2014). Since then, tremendous gains have been accomplished, thanks in large part to the FEVER shared task (Thorne & Vlachos, 2018). Unfortunately, the majority of the existing literature has studied fact verification only over unstructured, textual documents. This is in contrast to question answering, where semantic parsing over tables is well-studied (e.g. Pasupat and Liang (2015), Khashabi et al. (2016), and Yu et al. (2018)).

Two recent papers have proposed large-scale table fact verification datasets to fill this niche: Chen et al. (2020) and Gupta et al. (2020). These also include neural models for encoding tables; in fact, the BERT-based encoder introduced in the former is the closest related work to the model presented in this chapter. A few subsequent papers have introduced models for these datasets. In Zhong et al. (2020), a logic-based fact verification system was introduced to improve the model presented in the initial TabFact paper (Chen et al., 2020). Yang et al. (2020) builds on the program induction model also introduced in Chen et al. (2020), using a GNN to verify generated programs. Finally, Herzig et al. (2020) and Eisenschlos et al. (2020) introduced a BERT-based model for various table semantic tasks, extending BERT with additional position embeddings denoting columns and rows.

Outside of fact verification, a few papers have proposed general-purpose neural encoders for tables. The closest to our work is Yin et al. (2020). In their paper, a pretrained BERT-based encoder for tables is introduced and demonstrated to yield strong improvements on several semantic parsing tasks. Chen et al. (2019) introduced a model to automatically predict and compare column headers for tables in order to find semantically synonymous schema attributes. Similarly, Zhang and Balog (2019) introduced an autoencoder for predicting table relatedness.

## Operating in the Open Domain

Whether for fact verification or question answering, the systems we have so far discussed operate in the closed domain. For the open domain, attention has mostly been on unstructured, textual data. Early work resulted in several highly sophisticated full pipeline systems (Brill et al., 2002; Ferrucci et al., 2010; Sun et al., 2015). These inspired the influential DrQA model (Chen et al., 2017), which like many later systems – including ours – relies on a simple heuristic retrieval model, and a complex reading model. Recent work (Karpukhin et al., 2020; Lewis et al., 2020) has built on this approach, developing fast learned retrieval models through dot-product indexing (Johnson et al., 2019), and increasingly advanced pretrained transformer-models for reading. At present, unfortunately, a technique for fast maximum inner product search over structured data is not available.

Several strategies have previously been proposed for retrieving tables in different contexts. For question answering, Sun et al. (2016) used string matching between aliases of linked entities to search millions of tables crawled from the Web. Similarly, Jauhar et al. (2016) demonstrated strong results with a Lucene index and a Markov Logic Network-based model for answering scientific questions. Recently, Chakrabarti et al. (2020a) and Chakrabarti et al. (2020b) developed an improved model for table retrieval, combining neural representations of the table and the query with a BM25 index.

Concurrently with our work, Chen et al. (2021) have introduced a BERT-based model to perform question answering over open collections of data, including tables. Like ours, their model consists of separate retriever- and reader-components. Their retriever is an iterative two-step process that first generates a candidate pool through BM-25 scores, then refines it through a BERT-based scorer. Their best-performing reader employs a long-range sparse attention transformer (Ravula et al., 2020) to jointly summarize all retrieved data. As in our case, their model demonstrates significant improvements from using multiple retrieved tables.

Retrieving tables for use by a verification component is furthermore related to table search. Here, a line of research employs various means to rerank the tables returned by search engines or SQL querying. Cafarella et al. (2008) and Cafarella et al. (2009) employed keyphrase-based table retrieval by reranking a list of tables returned by a search engine. Pimplikar and Sarawagi (2012) demonstrated strong improvements using a graphical model to perform retrieval based on co-occurrence statistics, table metadata, and column headers. In Ghasemi-Gol and Szekely (2018), non-parametric clustering was employed as a strong heuristic for table retrieval. Zhang and Balog (2018) introduced a ranking method based on mapping available features into several semantic spaces. Recently, Zhang et al. (2019) introduced a neural method for table retrieval and completion using word- and entity-embeddings of table elements.

## 5.3 Methods

Formally, the open table fact verification problem can be described as follows. Given a claim $q$ and a collection of tables $T$, the task is to determine whether $q$ is true or false. This corresponds to modeling a binary verdict variable $v$ by $p(v|q,T)$. This is in contrast to the closed setting, where a single table $t_q \in T$ is given, and the task is to model $p(v|q,t_q)$. Since there are large available datasets for the closed setting (Chen et al., 2020; Gupta et al., 2020), it is reasonable to exploit an "optimal" table $t_q$ during training; however, at test time, this information may not be available.

We follow a two-step methodology that is often adopted in the open-domain setting for unstructured data (Chen et al., 2017; Nie et al., 2019; Karpukhin et al., 2020; Lewis et al., 2020) to our setting. Namely, given a claim $q$, we retrieve a set of evidence tables $D_q \subset T$, and subsequently model $p(v|q,D_q)$ in place of $p(v|q,T)$. We use a TF-IDF based heuristic to retrieve tables, which we then linearise and subsequently encode using RoBERTa (Liu et al., 2019), a large pretrained language model of the sort we discussed in Section 2.3.1. We introduce our retrieval and verification processes in the following sections.

### 5.3.1 Entity-based Retrieval

We begin by designing a strategy for retrieving an appropriate table or subset of tables to answer a given query. For question answering over tables, Sun et al. (2016) demonstrated strong performance on retrieval of candidate tables using entity linking information, following the intuition that many table cells contain entities. We take inspiration from these results. In their setting, claim entities are linked to Freebase entities, and string matching on the alias list is used to match entities to cells. To avoid linking claim entities to a knowledge base, we instead use only textual strings from the claim to represent entities, and perform approximate matching through dot products of bi- and tri-gram TF-IDF vectors.

We first compute TF-IDF vectors $z(c_t^1), ..., z(c_t^m)$ for every table $t \in T$ with cells $c_t^1, ..., c_t^m$. Then, we identify the named entities $e_q^1, ..., e_q^n$ within the query $q$. For our experiments, Chen et al. (2020) provided named entity spans for TabFact as part of their LPA-model, and we reuse those.[2] We compute TF-IDF vectors $z(e_q^1), ..., z(e_q^n)$ for the surface forms of those entities. We then use these to construct table-level scores for every $t \in T$. Since we are approximating entity linking between claim entities and cells, we let the score between an entity and a table be the *best* match between that entity and any cell in the table, using the dot product between entity-vectors and cell-vectors to represent relatedness.

**5.3.1.** Definition. The retrieval score of a table $t$ with respect to a query $q$ is defined as the sum of the best retrieval score between each query entity and the table:

$$score(q,t) = \sum_{i=1}^{n} \max_{j=1}^{m} z(e_q^i)^\intercal \cdot z(c_t^j)$$

That is, we compute for every entity the best match in the table, and score the table as the sum over the best matches. To construct the set of evidence tables $D_q$, we then retrieve the top-$k$ highest-scoring tables. Our choice to use bi- and tri-gram TF-IDF as the retrieval strategy was determined empirically – see Section 5.4 for experimental comparisons.

[2] In the absence of named entity tags, named entity spans would first need to be found through an off-the-shelf named entity recognizer, e.g. SpaCy (Honnibal et al., 2020). A vulnerability of our approach is as such that errors can propagate from the named entity recognizer.

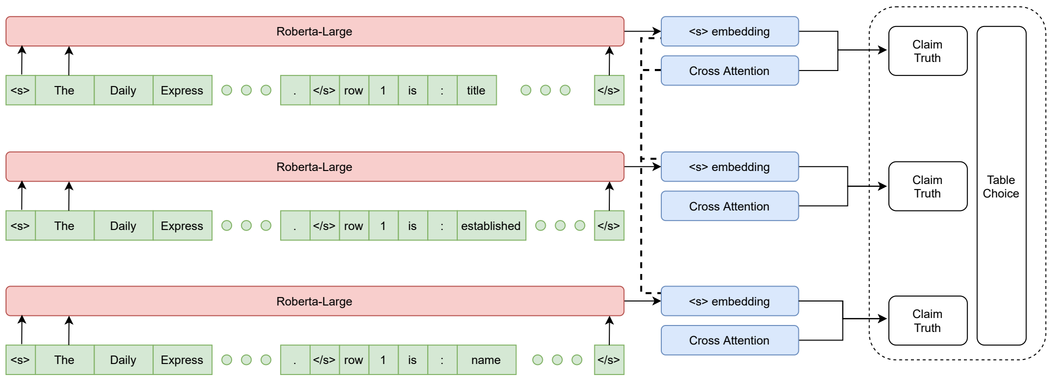


Figure 5.3: A diagram of our model, using the joint reranking- and verification approach described in Section 5.3.3. Linearised tables are encoded separately with RoBERTa. Then, cross-attention is used to contextualize each individual table with respect to the others. Finally, the model jointly predicts truth value and table selection.

## 5.3.2 Neural Verification

After we have retrieved a small set of evidence tables $D_q \subset T$, the task is to predict the truth value of the claim $q$ given this set of tables. That is, to model $p(v|q, D_q)$. To this end, we employ a late fusion strategy based on a pretrained language model (see Section 2.3.1). Given a query $q$ with a ranked list of $k$ retrieved tables $D_q = (d_q^1, ..., d_q^k)$, we begin by *linearising* each table. We then encode each linearised table as a vector with RoBERTa (Liu et al., 2019). Given a vector embedding of each table in $D_q$, we then contextualize these table embeddings through cross-attention between the representations. Finally, we predict the truth value of the claim given these contextualized table embeddings. A diagram of this process can be seen in Figure 5.3.

Our linearisation scheme follows Chen et al. (2020). We first perform sub-table selection by excluding columns not linked to entities in the query. Here, we reuse the entity linking obtained during the retrieval step, and retain only the three columns in which cells received the highest retrieval scores. We linearise each row separately, encoding entries and table headers. Suppose $r$ is a row with cell entries $c_1, c_2, ..., c_m$ in a table, where the corresponding column headers are $h_1, h_2, ..., h_m$. Row number $r$ is then mapped to *"row $r$ is : $h_1$ is $c_1$ ; $h_2$ is $c_2$; ... ; $h_m$ is $c_m$ ."* An example of this linearisation scheme can also be seen in Figure 5.1. We construct a final linearisation $L_{q,t}$ for each query-table pair $q, t$ by prepending the query to the filtered table linearisation.

**5.3.2.** DEFINITION. The linearisation $L_{q,t}$ of a table $t$ with respect to a query $q$ is the sequence of tokens defined as $[CLS], q_1, ...q_n, [SEP], t_1, ..., t_m, [EOS]$

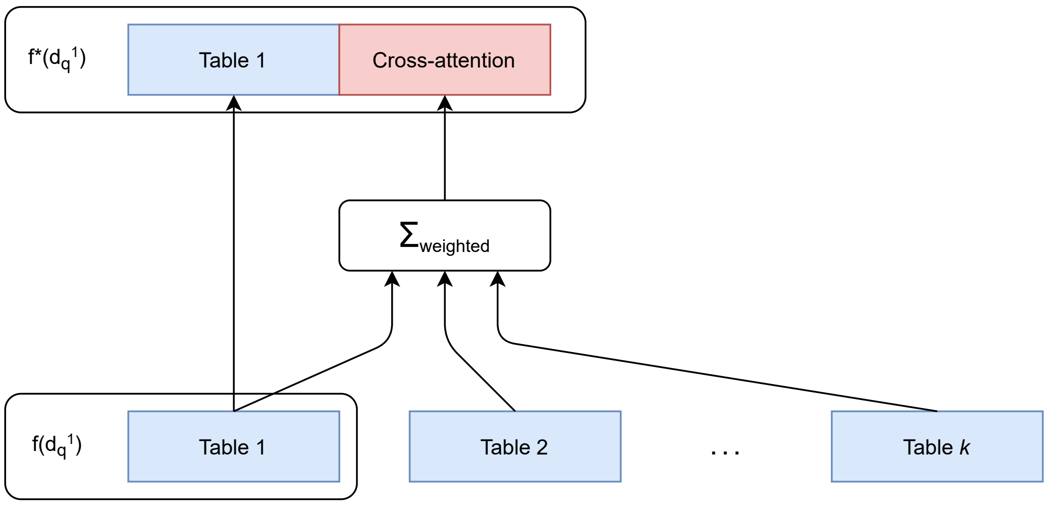


Figure 5.4: After each retrieved evidence table is embedded with RoBERTa as a table embeddings $f(d_q^k)$, our model uses a *cross-attention* to contextualize each table with respect to the others. A learned attention function constructs a weighted sum of the embeddings on the basis of $f(d_q^k)$, and the result is concatenated to $f(d_q^k)$ resulting in a contextualized table embedding $f^*(d_q^k)$.

where $q_1, ..., q_n$ is the sequence of tokens in $q$ and $t_1, ..., t_m$ is the sequence of tokens resulting from mapping the subtable consisting of the three highest-scored columns in $t$ to textual form.

We then encode each $L_{q,t}$ with RoBERTa, and obtain contextualised RoBERTa-embeddings $f(d_q^k) \in \mathbb{R}^n$ for every table as the final-layer embedding of the CLS-token (see Section 2.3.1. We construct the sequence of embeddings $f(d_q^1), ..., f(d_q^k)$ for all $k$ tables. We furthermore define a tensor $F(D_q)$ combining these.

**5.3.3.** DEFINITION. The table evidence vectors $f(d_q^1), ..., f(d_q^k)$ for a query $q$ with retrieved tables $D_q = d_q^1, ..., d_q^k$ is the RoBERTa-embedding of the CLS-tokens of $L_{q,d_q^1}, ..., L_{q,d_q^k}$. The evidence embedding tensor $F(D_q)$ is the stacking $[f(d_q^1)^\intercal, ..., f(d_q^k)^\intercal]^\intercal$ of the corresponding table evidence vectors.

When the model attempts to judge whether to rely on a given table for verification, other highly-scored tables represent useful contextual information. For example, in Figure 5.2, newspapers may be more likely to share political leanings if they also share an owner. Nevertheless, each table embedding $f(d_q^k)$ is constructed independently of the other retrieved tables in $D_q$. As such, the model cannot take these contextual clues into account.

To remedy this, we introduce a cross-attention layer between all tables corresponding to the same query (see Figure 5.4). We apply a single multi-head attention transformation[3] to the evidence embedding tensor $F(D_q)$, performing cross-attention between the rows corresponding to individual tables. We subsequently concatenate the result to the original evidence embedding tensor. That is, we compute an attention score for head $h$ from table $i$ to table $j$ with query $q$ as:

$$\alpha_{ij}^h = \sigma\left(\frac{W_Q^h f(d_q^i)(W_K^h f(d_q^j))^T}{\sqrt{dim(K)}}\right) \tag{5.1}$$

where $\sigma$ is the softmax function, and $W_Q$ and $W_K$ represent linear transformations to respectively queries and keys. We then compute an attention vector for that head as:

$$A_i^h = \sum_{j \in D_q} \alpha_{ij} W_V^h f(d_q^j) \tag{5.2}$$

We finally construct contextualized table representations through concatenation as:

**5.3.4.** Definition. Given a table embedding tensor $F(D_q)$, the contextualized table embedding $f^*(d_q^k)$ of table $k$ is defined as:

$$f^*(d_q^k) = [f(d_q^k), A_k^1, ..., A_k^h]$$

We subsequently use $F^*(D_q)$, e.g. the evidence tensor corresponding to $f^*(d_q^1)$, ..., $f^*(d_q^k)$, for downstream predictions. We note that our approach can be viewed as an extension of the Table-BERT algorithm introduced in Chen et al. (2020) to the multi-table setting, using an attention function to fuse the information from different tables.

### 5.3.3 Training & Testing

At training time, relying on a closed-setting dataset allows us to identify which tables contain appropriate information for answering each query (e.g., the table against which the claim is to be checked in the closed setting). Although this information is not available at test time, we can construct a training regime that allows us to exploit it to improve model performance, as well as obtain a test-time indicator that an appropriate table has not been retrieved. We identify and experiment with two different approaches to modelling this problem. The first option is to jointly model the choice of table and the truth value of the claim. We refer to this as *joint reranking and verification*. The second option is to model for each table a choice between indicating that the claim is true, that the claim is

[3] See Section 2.3 for our introduction to attention. We also refer the reader to Vaswani et al. (2017) for an extended discussion of the mechanism. As with the transformer, this attention layer can be seen within the GNN framework as a single step of GAT-propagation (Veličković et al., 2018) in the fully connected graph between all tables.

false, or that no information about the claim is given. We refer to this as *ternary verification*. In Section 5.4, we demonstrate how the former leads to increased performance on verification, while the latter gives access to a strong predictor for cases where no appropriate table has been retrieved to verify the query.

**Joint reranking and verification** For the joint reranking and verification approach, we assume that a single *best* table $t_s$ for answering each query exists. This best table can then be used to learn a ranking function. We model this as choosing the right table from $D_q$, e.g. through a categorical variable $s$ that indicates which table should be selected. We then learn a joint probability of $s$ and the truth value of the claim $v$ over the tables. Assuming that $s$ and $v$ are independent, $p(s, v|q, D_q)$ is also a categorical distribution with one correct outcome that can be optimized for (that is, one correct pair of table and truth value).

**5.3.5.** Definition. With the joint reranking and verification objective, the joint probability of choosing the most appropriate table $s$ and the truth value of the claim $v$ is defined as

$$p(s, v|q, D_q) = \sigma(W(F^*(D_q)_s)_v)$$

where $W : \mathbb{R}^{2n} \rightarrow \mathbb{R}^2$ is an MLP and $\sigma$ is the softmax function.

At training time, we obtain one cross-entropy term corresponding to $p(s, v|q, D_q)$ for each query. At test time, we can marginalize over $s$ to obtain a final truth value:

$$p_v(v|q, D_q) = \sum_{t \in D_q} p(v, s = t|q, D_q) \tag{5.3}$$

This formulation has the additional benefit of also allowing us to make a prediction on which table matches the query. We can do so by marginalizing over $v$:

$$p_s(s|q, D_q) = \sum_{v_q \in \{true, false\}} p(s, v = v_q|q, D_q) \tag{5.4}$$

This loss is only well-defined so long as the gold table $t_s$ appears in $D_q$. As such, when we train the model, we replace $D_q$ with an alternative set $D_q^*$. If $t_s \in D_q$, we simply let $D_q^* = D_q$. If $t_s \notin D_q$, we construct $D_q^*$ by substituting the lowest-scored table in $D_q$ according to our retrieval system with $t_s$.

**Ternary verification** At test time, there may be cases where a table refuting or verifying the fact has not been retrieved. That is, it may be the case that no relevant table exists in $D_q$. For some applications, it could be useful to identify these cases. We therefore design an alternative variant of our system better suited for this scenario. Intuitively, each table can represent three outcomes – the query is true, the query is false, or the table is irrelevant. We can model this through a ternary variable $i$.

**5.3.6.** Definition. With the ternary verification objective, the probability $i$ of choosing whether a given table $t$ indicates that the query is true, indicates that the query is false, or gives no indication either way is defined as:

$$p(i|q, t, D_q) = \sigma(W'(F^*(D_q)_t)_i)$$

where $W' : \mathbb{R}^{2n} \to \mathbb{R}^3$ is an MLP and $\sigma$ is the softmax function.

During training, we then assign *true* or *false* to the gold table depending on the truth of the query, and *irrelevant* to every other table. We then use the mean cross-entropy over the tables associated with each query as the loss for each example. At test time, we compute the truth value $v$ of each query as:

$$\sum_{t \in D_q} p(i = true|q, t) > \sum_{t \in D_q} p(i = false|q, t) \tag{5.5}$$

## 5.4 Experiments

We apply our model to the TabFact dataset (Chen et al., 2020), which consists of 92,283 training, 12,792 validation, and 12,792 test queries to be validated against 16,573 tables. The task is to classify, for each claim, whether it is true or false given the knowledge source. To benchmark our open-domain models, we begin by performing evaluation in the closed domain. This enables us to construct upper and lower bounds for the performance of our open-domain model. As an upper bound, we can compare to using a single table retrieved through an oracle – that is, the equivalent of the closed-domain setting. As a lower bound, we can use the highest-ranked table according to our TF-IDF retriever, with no reranking. The evaluation metric is simply prediction accuracy.

### 5.4.1 Evaluating Retrieval

We choose the retrieval strategy empirically, settling on bi- and tri-gram TF-IDF as the best-performing option among those we tested. To address the comparative performance of this choice, we compute and rank in Table 5.1 the retrieval scores obtained through our strategy on the TabFact validation set, using several alternative strategies. In addition to entity-level bi- and tri-gram TF-IDF, we try using bi- and trigram TF-IDF vectors for all words in the claim (rather than just the entities), word-level TF-IDF vectors for entities, and entity-level exact matching. Our bi- and tri-gram TF-IDF strategy yields by far the strongest

| Dataset | H@1 | H@3 | H@5 | H@10 |
|---|---|---|---|---|
| Claim-level word TF-IDF | 41.7 | 54.2 | 59.0 | 65.3 |
| Claim-level (2,3)-gram TF-IDF | 34.7 | 45.5 | 50.2 | 56.8 |
| Entity-level exact match | 48.2 | 57.9 | 64.2 | 67.3 |
| Entity-level word TF-IDF | 56.0 | 65.6 | 74.1 | 81.2 |
| Entity-level (1,2,3)-gram TF-IDF | 62.3 | 75.2 | 80.1 | 86.1 |
| Entity-level (2,3)-gram TF-IDF | 69.6 | 78.8 | 82.3 | 86.6 |

Table 5.1: Retrieval accuracy for our entity-based TF-IDF retrieval along with several baselines for the TabFact validation set, computed using all 16,573 TabFact tables.

performance. Interestingly, the exclusion of unigrams from the TF-IDF vectors slightly increases performance.

### 5.4.2 Evaluating Verification

In Table 5.2, we compare our approach to the closed-setting system from Chen et al. (2020), as well as to several recent models from the literature (Zhong et al., 2020; Yang et al., 2020; Eisenschlos et al., 2020). We include results with both losses as discussed in Section 5.3, using varying numbers of tables.

With an accuracy of 75.1%, we obtain the best open-domain results with our model using the joint reranking-and-verification loss and five tables. We see performance improvements when increasing the number of tables, both from one to three and from three to five, although increasing the number of retrieved tables to ten decreases performance. In the closed domain, the 77.6% accuracy our model achieves is a significant improvement over the 74.4% the strongest comparable baseline reached. This may be due to our use of RoBERTa, which has previously been found to yield improvements for linearised tables (Gupta et al., 2020). As expected, the joint loss, which trains the model to focus on one particular table, performs better than the ternary loss (see Section 5.3.3).

Relying purely on TF-IDF for retrieval – that is, using our system with only one retrieved table – yields a performance of 73.2%. This represents a surprisingly small decrease of 4.6% accuracy compared to the closed setting, given that an incorrect table is provided in approximately a third of all cases (see Table 5.1). We suspect that many of the claims which the closed-setting model fails to correctly classify are difficult cases, for which the retriever also fails. To make sure we are not seeing the effect of false negatives (e.g. tables that are not the gold table, but which nevertheless have the information to verify the claim), we train and test the model in a setting where one retrieved table is used, but the gold table is removed from the retrieval results. Here, the model achieves an accuracy of

| Model | Test | Simple | Complex | Small |
|---|---|---|---|---|
| Chen et al. (2020) | 65.1 | 79.1 | 58.2 | 68.1 |
| Zhong et al. (2020) | 71.7 | 85.4 | 65.1 | 74.3 |
| Yang et al. (2020) | 74.4 | 88.3 | 67.6 | 76.2 |
| Eisenschlos et al. (2020)* | 81.0 | 92.3 | 75.6 | 83.9 |
| Ours (Oracle retrieval) | 77.6 | 88.9 | 72.1 | 79.4 |
| Ours (1 retrieved table) | 73.2 | 86.7 | 67.8 | 76.6 |
| Ours (Ternary loss, 3 tables) | 73.5 | 86.9 | 68.1 | 76.9 |
| Ours (Ternary loss, 5 tables) | 73.7 | 87.1 | 67.9 | 76.5 |
| Ours (Ternary loss, 10 tables) | 73.1 | 86.5 | 67.9 | 77.3 |
| Ours (Joint loss, 3 tables) | 73.8 | 87.0 | 68.3 | 78.1 |
| Ours (Joint loss, 5 tables) | **75.1** | **87.8** | **69.5** | **77.8** |
| Ours (Joint loss, 10 tables) | 73.8 | 86.9 | 68.1 | 76.9 |

Table 5.2: Prediction accuracy of our RoBERTa-based model on the official splits from the TabFact dataset. We include closed-domain performance of several models from the literature, as well as the performance of our model in both the closed and the open domain, using both proposed loss functions. The first section of the table contains closed-domain results, the second open-domain. * employs intermediary pretraining on additional synthetic data.

only 56.2%. We furthermore experiment with a system trained and tested using a random table rather than a retrieved table; with a performance drop to 53.1, we find that the information in the retrieved table is indeed what drives the model (rather than e.g. RoBERTa weights).

**Hyperparameters** Our model uses RoBERTa (Liu et al., 2019) to encode each table into vectors. For the cross-attention function, we choose empirically to use two attention heads. We use an MLP consisting of a linear transformation to $h = 3072$ hidden units, followed by *tanh*-activation and linear projection to the output space. During training, we employ dropouts with probability 0.1 before each linear transformation in the MLP.

We train the model using Adam (Kingma & Ba, 2015) with a learning rate of 5e−6. We use a linear learning rate schedule, warming up over the first 30000 batches. We use a batch size of 32. Training was done on 8 NVIDIA Tesla V100 Volta GPUs (with 32GB of memory) and completed in approximately 36 hours.

### 5.4.3 Analysis

To understand how our model derives improvement from the addition of more tables, we compute in Table 5.3 the performance of our reranking-and-verification

| Model | R@1 | R@2-3 | R@4-5 |
|---|---|---|---|
| Oracle retrieval | 80.6 | 74.1 | 75.0 |
| 1 table | 80.6 | 55.6 | 53.9 |
| 3 tables | 78.8 | 66.7 | 58.2 |
| 5 tables | 79.4 | 73.1 | 71.7 |

Table 5.3: Peformance of our RoBERTa-based model on the parts of the TabFact test set where our TF-IDF retriever assigns the gold table rank respectively 1, 2-3, or 4-5.

model when TF-IDF returns the correct table at rank 1, rank 2-3, or rank 4-5. Immediately, we notice a much stronger improvement from using multiple tables when TF-IDF fails to identify the gold table correctly. This is natural, as those are exactly the cases where our model (as opposed to the baseline) has access to the appropriate information to verify or refute the claim.

Interestingly, using three tables improves on using one table even when the gold table is *not* included among the top three (from 53.9% to 58.2%), and using five tables improves on using three tables also when the gold table *is* included among the top three (from 66.7% to 73.1%). Manual inspection reveals than our model in many cases can rely on correlations between tables, even when TF-IDF has not retrieved the gold table – if a sports team loses games in three tables, then that may give a higher probability of that team also losing in an unretrieved, hypothetical fourth table. To test this, we apply the model in a setting where we retrieve the top five tables *excluding* the gold table, and a setting where we use five random tables. Using highly scored (but wrong) tables, we achieve a performance of 59.4%, a significant improvement on the 53.1% we achieve using random tables. This supports our hypothesis that other *good* tables can provide useful background context for verification.

It is worth noting that reliance on such information, while increasing model performance, may also increase the degree to which the model makes inference based on inexactness or bias. Depending on the application, correlations between information in various tables may not be a desirable basis for verifying facts. Returning to the example in Figure 5.2, inferring ownership on the basis of political affiliation when no other information is available may increase accuracy *on average*, but it can also lead to erroneous or biased decisions (indeed, for the claim in the example, the prediction would be wrong).

Our best-performing model from Table 5.2 relies on two innovations: the cross-attention function, which contextualizes each table in relation to the other retrieved table for that claim, and the joint reranking-and-verification loss. In Table 5.4, we evaluate the model without either of these. Leaving the attention function out is simple – we use the RoBERTa-embeddings $f(d_q^k)$ for each table

| Model | Accuracy |
|---|---|
| Full model | 75.1 |
| - Attention | 73.6 |
| - Joint objective | 72.9 |
| - Both | 71.2 |

Table 5.4: Ablation study for our model, performing verification with the five-table version on the TabFact test set. We remove respectively our cross-attention function, the reranking component in the loss, and both.

| Model | H@1 | H@3 | H@5 |
|---|---|---|---|
| TF-IDF | 69.6 | 78.8 | 82.3 |
| Reranking only | 69.9 | 78.9 | 82.3 |
| Ours (no attention) | 67.4 | 78.3 | 82.3 |
| Ours (attention) | 70.9 | 79.4 | 82.3 |

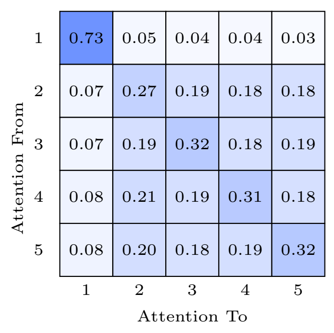
Table 5.5: Ranking performance on the TabFact validation set, using either our TF-IDF retriever alone or reranking with our model. We test a version of our model using only a reranking loss, as well as joint-loss model with and without attention.

directly for predictions. For testing our loss without reranking, we assume a uniform distribution over the tables. As can be seen, the combination of both is strictly necessary to obtain strong performance. Indeed, without our joint objective, the model performs worse than simply applying the baseline model to the top table returned by TF-IDF as in Table 5.2.

In Section 5.3, we introduced our model as a joint system for fact verification and evidence reranking. A benefit of our formulation is the ability to reason about the ability of our model to rerank by marginalizing over the truth value of the claim, following Equation 5.4. In Table 5.5, we compare the table retrieval ranking performance of our joint model to a model only trained for reranking, as well as to the TF-IDF baseline.

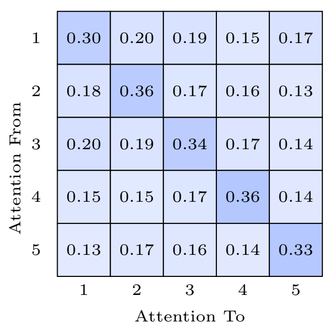
As can be seen, our joint loss provides a slight performance improvement when the attention component is included. Interestingly, the joint-loss model performs better than a system trained purely for reranking — this highlights the complementary nature of the reranking and verification tasks.

An interesting question is which role attention plays in our model. As can be seen from Tables 5.4 and 5.5, our cross-attention module is necessary to achieve high performance – without it, the model struggles to identify which table should be used for verification. To investigate the function of attention, we plot in Figure 5.5 the strength of the cross-attention between each table for our five-table model.



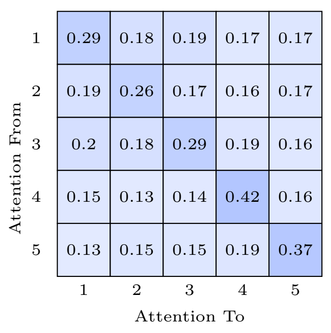
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0.73 | 0.05 | 0.04 | 0.04 | 0.03 |
| 2 | 0.07 | 0.27 | 0.19 | 0.18 | 0.18 |
| 3 | 0.07 | 0.19 | 0.32 | 0.18 | 0.19 |
| 4 | 0.08 | 0.21 | 0.19 | 0.31 | 0.18 |
| 5 | 0.08 | 0.20 | 0.18 | 0.19 | 0.32 |

(a) Head 1, R@1



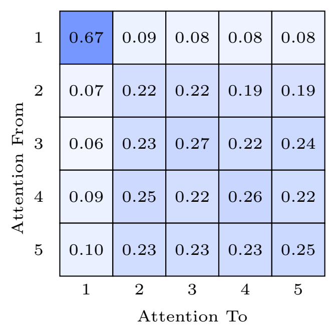
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0.30 | 0.20 | 0.19 | 0.15 | 0.17 |
| 2 | 0.18 | 0.36 | 0.17 | 0.16 | 0.13 |
| 3 | 0.20 | 0.19 | 0.34 | 0.17 | 0.14 |
| 4 | 0.15 | 0.15 | 0.17 | 0.36 | 0.14 |
| 5 | 0.13 | 0.17 | 0.16 | 0.14 | 0.33 |

(b) Head 1, R@2-3



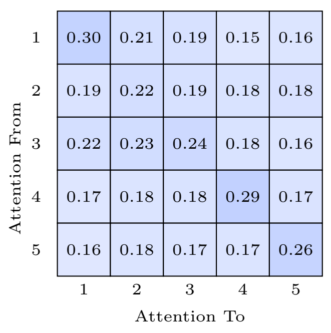
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0.29 | 0.18 | 0.19 | 0.17 | 0.17 |
| 2 | 0.19 | 0.26 | 0.17 | 0.16 | 0.17 |
| 3 | 0.2 | 0.18 | 0.29 | 0.19 | 0.16 |
| 4 | 0.15 | 0.13 | 0.14 | 0.42 | 0.16 |
| 5 | 0.13 | 0.15 | 0.15 | 0.19 | 0.37 |

(c) Head 1, R@4-5



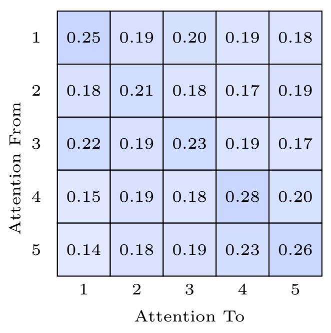
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0.67 | 0.09 | 0.08 | 0.08 | 0.08 |
| 2 | 0.07 | 0.22 | 0.22 | 0.19 | 0.19 |
| 3 | 0.06 | 0.23 | 0.27 | 0.22 | 0.24 |
| 4 | 0.09 | 0.25 | 0.22 | 0.26 | 0.22 |
| 5 | 0.10 | 0.23 | 0.23 | 0.23 | 0.25 |

(d) Head 2, R@1



| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0.30 | 0.21 | 0.19 | 0.15 | 0.16 |
| 2 | 0.19 | 0.22 | 0.19 | 0.18 | 0.18 |
| 3 | 0.22 | 0.23 | 0.24 | 0.18 | 0.16 |
| 4 | 0.17 | 0.18 | 0.18 | 0.29 | 0.17 |
| 5 | 0.16 | 0.18 | 0.17 | 0.17 | 0.26 |

(e) Head 2, R@2-3



| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0.25 | 0.19 | 0.20 | 0.19 | 0.18 |
| 2 | 0.18 | 0.21 | 0.18 | 0.17 | 0.19 |
| 3 | 0.22 | 0.19 | 0.23 | 0.19 | 0.17 |
| 4 | 0.15 | 0.19 | 0.18 | 0.28 | 0.20 |
| 5 | 0.14 | 0.18 | 0.19 | 0.23 | 0.26 |

(f) Head 2, R@4-5

Figure 5.5: Confusion matrices for the cross-attention between each pair of tables for the five-table version of our model. Each head is represented separately, and individual figures are included for the parts of the dataset where our TF-IDF retriever assigns the gold table rank respectively 1, 2-3, or 4-5.

We produce separate plots for the two attention heads, as well as for each of the splits used in Table 5.3 representing the parts of the dataset where our TF-IDF retriever assigns the gold table rank respectively 1, 2-3, or 4-5.

For both attention heads, the attention function has clearly distinct behaviour when the gold table is retrieved as top 1; the degree to which that table attends to itself is much greater. We suspect that this is because of "easy" cases, where the attention function is used to separate a clearly identifiable "appropriate" table from the other tables. In harder cases, the model uses the attention focus to compare information across tables. To test this, we run the model in a setting where four random tables are used along with the gold table. In that setting, the division is even clearer. For the gold table, respectively 86 and 82 percent of the attention for the two heads is on average focused on itself; for the four random tables, the attention is evenly distributed over all tables except the gold table.

To distinguish the two heads, we in general see the first head exhibit a pattern of behaviour where each table assigns the majority of attention to itself — especially when that table is the gold table. The second head seemingly encodes a

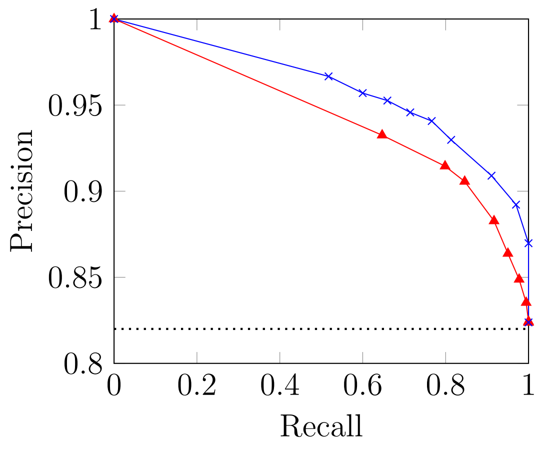


Figure 5.6: Precision-recall curve for determining whether a set of five retrieved tables in the TabFact validation set contains the gold table, using respectively entropy of the reranking scores with our joint loss (▲) or the maximum probability of some table being the gold table with our ternary loss, (×). We also include a most frequent class baseline (⋯).

slightly more even spread over the retrieved tables, perhaps representing general context more than an attempt to identify the gold table.

In realistic settings, some claims will not be directly answerable from any retrieved table. This could be because of inexactness in the retrieval algorithm, or because the trusted corpus of tables does not contain the information that a user is querying for. In such cases, it can be valuable to inform the user explicitly – giving false verifications or refutations when sufficient information is not available may mislead users and decrease trust in the verification system. To model a scenario where information may not always be present, we create a classification task wherein the model must predict for all examples whether the gold table is among the $k$ documents in $D_q$.

Using the ternary loss, we can directly obtain the probability of each table containing appropriate information to verify the claim from our model. We do so by computing $(1-p(I_t = irrelevant|q,t))$. We can estimate the suitability of the best retrieved table for verifying the claim as $\max_t(1-p(I_t = irrelevant|q,t))$, and apply a threshold $\tau_1$ to classify $D_q$ as suitable or unsuitable. For the joint loss, a more indirect approach is necessary. Intuitively, if our model is too uncertain about which retrieved table contains the information to answer the query, there is a high likelihood that no suitable table has been retrieved. For our joint objective, we can model this directly – this case occurs exactly when the entropy of the reranking component $H_s(s|q,D_q)$ after marginalizing over the truth value of the claim exceeds some threshold $\tau_2$.

We compare these strategies in Figure 5.6, obtaining Precision-Recall curves by measuring at varying $\tau_1$ and $\tau_2$. We find that while both approaches out-

| Model | Accuracy |
|---|---:|
| RoBERTa only | 52.1 |
| Ours (1 table) | 53.6 |
| Ternary loss, 3 tables | 55.8 |
| Ternary loss, 5 tables | 57.5 |
| Joint loss, 3 tables | 56.1 |
| Joint loss, 5 tables | **58.1** |

Table 5.6: Performance of our RoBERTa-based model on the TabFact test set, using all Wikipedia tables rather than just the TabFact tables as a backend.

perform the most frequent class baseline by a significant margin, the ternary loss performs better than the joint loss. As such, the choice between the two losses represents a tradeoff between raw performance and the ability to identify verifications made based on missing or incomplete information.

### 5.4.4 Wikipedia-scale Verification

In our experiments so far, we have relied on the 16,573 TabFact tables as a knowledge source. The tables selected for TabFact were taken from WikiTables (Bhagavatula et al., 2013), and filtered to exclude "overly complicated and huge tables" (Chen et al., 2020). Moving beyond the scope of that dataset, a fully open fact verification system should be able to verify claims over even larger collections of tables — for example, the full set of tables available from Wikipedia. To make a preliminary exploration of that larger-scale setting, we include in Table 5.6 the performance of our approach evaluated using roughly 3 million tables automatically extracted from Wikipedia.

As can be seen, our approach improves on the naive strategy of using a single table and a closed-domain verification component also in this more complex setting. To verify that the inference happens based on the retrieved tables and not simply the RoBERTa-weights, we include the performance of a model that simply uses classification on top of a RoBERTa-encoding of the claim. As previously, our joint-loss model with five retrieved tables performs the strongest. We note that it is unclear whether the performance we observe here originates from correlations obtained through background information (as we saw earlier when the retriever failed to find the appropriate TabFact table), or due to verification against a single entirely appropriate table happening, but at a lower rate than when using TabFact.

## 5.5 Conclusion

We have introduced an attention-based model for fact verification over open collections of tables, along with two separate strategies for handling reranking and verification through the same model. Our approach achieves performance on par with the current state of the art for the closed setting on the TabFact dataset, with larger performance gains the more tables we include. When using an oracle to retrieve a reference table, our RoBERTa-based approach to fact verification also represents a new state of the art for the closed setting. Finally, we have made an initial foray into Wikipedia-scale open domain table fact verification, demonstrating improvements from using multiple tables also when evaluating with a full set of Wikipedia tables as a knowledge source.

Our results indicate that the use of multiple tables can provide contextual clues to the model even when those tables do not explicitly verify or refute the claim, because they can provide evidence for the *probability* of the claim. This is a double-edged sword, as reliance on such clues can increase model performance while also induce faulty claims of truthfulness due to inaccuracy or bias. As such, care will be needed to disentangle the positive and negative aspects of this phenomenon in future work.

Reflecting on our research questions, it is clear that linearisation followed by language modelling represents a highly effective means of encoding tables for fact verification, including for settings with multiple retrieved tables. The existence of datasets labelled with "correct" tables to judge claims are highly beneficial for training open-domain models, as we have demonstrated; introducing a reranking objective into the loss significantly improves performance. The combination of attention over multiple tables and our novel training objective enables verification in the difficult setting wherein the entirety of Wikipedia is used as a knowledge source, even without retraining.

Through the use of large pretrained language models, linearisation remains an effective alternative to semantic parsing strategies as well as to the GNN-based models we have explored in previous chapters. Interestingly, the class of models which enables this development, transformers, can (as we discuss in Section 2.3) themselves be seen as a form of GNNs operating in the fully connected graph between all tokens in the analysed linearisation.

Chapter 6

# Interpreting Graph Neural Networks

## 6.1 Introduction

In the previous chapters, we have explored how graph neural networks can be included as part of NLP models to process graph-structured data. In parallel with our work, GNNs have been applied with great results to many NLP tasks, including relation extraction (Zhang et al., 2018; Zhu et al., 2019; Sun et al., 2019a; Guo et al., 2019), question answering (Sorokin & Gurevych, 2018; Sun et al., 2018; De Cao et al., 2019), syntactic and semantic parsing (Marcheggiani & Titov, 2017; Bogin et al., 2019; Ji et al., 2019), summarisation (Fernandes et al., 2019), machine translation (Bastings et al., 2017) and abusive language detection (Mishra et al., 2019). While graph neural networks have yielded strong improvements in all these cases, the complexity of real-world applications often makes it difficult to understand on what basis predictions are made.

Although stronger performance is in itself a useful aim, the inability for human analysts to understand why predictions are made is a problematic side-effect. Such opaqueness prevents users from trusting model predictions (Kim, 2015; Ribeiro et al., 2016a), makes it hard to determine if models exhibit harmful biases (Sun et al., 2019c; Holstein et al., 2019), and prevents researchers from detecting model or data deficiencies, as well as from performing error analysis (Gururangan et al., 2018; Kaushik & Lipton, 2018). The latter is especially important for GNNs, where seemingly small implementation differences have been shown to make or break models (Zaheer et al., 2017; Xu et al., 2019).

In NLP it is furthermore often desirable to know which linguistic information a given model encodes, and how that encoding happens (Jumelet & Hupkes, 2018;

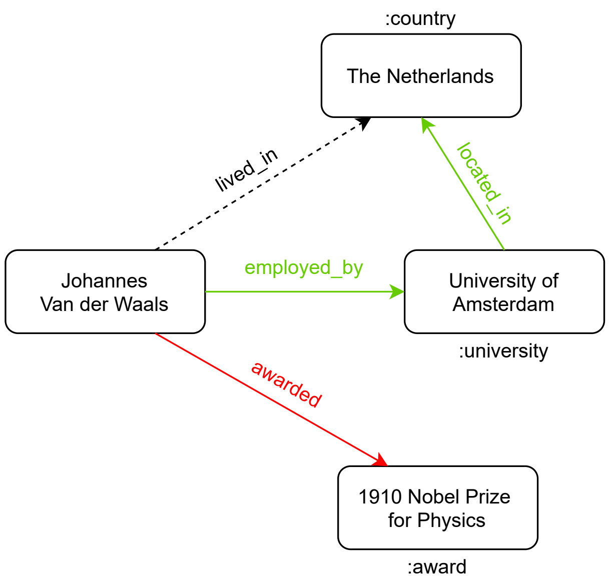


Figure 6.1: Likely useful and superfluous edges from the example in Figure 3.1. When predicting whether Van der Waals lived in the Netherlands (dotted, black edge), knowing that he won a Nobel prize (red edge) is unlikely to be informative, whereas knowing that he worked at the University of Amsterdam (green edges) is. A well-trained GNN might as such ignore the former while relying heavily on the latter.

Giulianelli et al., 2018; Goldberg, 2019). Since GNNs are often used exactly to encode linguistic structures, the difficulty in understanding their predictions can impede such analysis. As such, it is highly desirable to develop *interpretability* techniques for graph neural networks.

In this chapter and in Schlichtkrull et al. (2020b), we develop an interpretability technique[1] for GNNs. We are interested especially in determining which parts of the input graph a given GNN relies on – see for example Figure 6.1. We are furthermore interested in developing such analysis quantitatively on the level of an entire dataset. We therefore focus on developing a *post-hoc analysis* method for GNNs. To give satisfying answers to common questions about the behaviour of GNNs, we believe our technique should:

1. be able to identify relevant *paths* across layers, as paths are one of the most natural ways of presenting GNN reasoning patterns to users;
2. be sufficiently *tractable* to apply to modern GNN-based NLP models;
3. be as *faithful* (Jacovi & Goldberg, 2020) as possible, providing insights into how the model truly arrives at the prediction.

[1] The concept of interpretability has varying definitions – we refer the reader to Section 6.2 for an outline of how we understand the concept, and to Lipton (2016) for further discussion.

There are different aspects of graph neural networks which a researcher might be interested in interpreting. Depending on the circumstances, the most relevant aspect to investigate might be how vertices are used, which roles different weights play, or which edges are used to propagate information across the graph. Here, we focus primarily on understanding how *edges* are used. We do so under the condition that it should also be possible to extract relevant *paths*. We attempt to answer the following research questions:

1. *How can we tractably provide rationales for the predictions of a given graph neural network?*

2. *How can we ensure that our technique is faithful, and measure the relative faithfulness of different techniques?*

A straightforward way to determine which parts of the input (for example, a set of edges) play a role in predictions is to use *erasure search* (Li et al., 2016a; Feng et al., 2018). In that approach, attribution happens by searching for a maximal subset of input elements that can be entirely removed without affecting model predictions. Complete removal guarantees that the model ignores all information about the discarded elements, and elements that can safely be removed can be said to be *superfluous.*

This contrasts with approaches which use heuristics to define feature importance, for example attention-based techniques (Serrano & Smith, 2019; Jain & Wallace, 2019) or back-propagation techniques (Bach et al., 2015; Sundararajan et al., 2017). These techniques do not guarantee that the model completely ignores low-scored features, attracting criticism in recent years (Nie et al., 2018; Sixt et al., 2019; Jain & Wallace, 2019). The trust placed by the community in erasure search is reflected in the literature, where other methods are often motivated as approximations of erasure (Baehrens et al., 2010; Simonyan et al., 2014), or through new attribution techniques being evaluated using erasure search as ground truth (Serrano & Smith, 2019; Jain & Wallace, 2019).

Applied to GNNs, erasure search would involve searching for the largest subgraph, which can be completely discarded. In addition to a strong measure of faithfulness and the benefit of conceptual simplicity, discrete attributions would also simplify the comparison of relevance between paths – the important paths are those for which all edges are included. This contrasts with continuous attribution to edges, where it is not straightforward to extract and visualize important paths. Furthermore, as opposed to techniques based on artificial gradients (Pope et al., 2019; Xie & Lu, 2019; Schwarzenberg et al., 2019), erasure search would provide implementation invariance (Sundararajan et al., 2017). This is important in NLP, as models commonly use highly parametrized decoders on top of GNNs, e.g. Koncel-Kedziorski et al. (2019).

While arguably satisfying criteria (1) and (3) in our desiderata, erasure search unfortunately fails on tractability as every single combination of removed features must be evaluated. In practical scenarios, this is infeasible. Even approximations, which remove one feature at a time (Zintgraf et al., 2017) and underestimate their contribution due to saturation (Shrikumar et al., 2017), remain prohibitively expensive.

Our GraphMask aims at meeting the above desiderata by achieving the same benefits as erasure search in a scalable manner. That is, our method makes easily interpretable hard choices on whether to retain or discard edges. GraphMask finds edges that have no relevance to model predictions, while remaining tractable and model-agnostic (Ribeiro et al., 2016b). GraphMask can be understood as a differentiable form of subset erasure. Instead of finding an optimal subset to erase for every given example, we learn an erasure function that predicts for every edge $(u, v, r, \delta)$ at every layer $k$ whether that edge should be retained. Given an example graph $\mathcal{G}$, our method returns for each layer $k$ a subgraph $\mathcal{G}_S^{(k)}$ such that we can faithfully claim that no edges outside $\mathcal{G}_S^{(k)}$ influence the predictions of the model. To enable gradient-based optimization for our erasure function, we rely on sparse stochastic gates (Louizos et al., 2018; Bastings et al., 2019).

In erasure search, optimization happens individually for each example. This can result in a form of overfitting where even non-superfluous edges are aggressively pruned, because a similar prediction could be made using an alternative smaller subgraph; we refer to this problem as *hindsight bias*. This flaw is shared with the interpretability technique for GNNs closest to ours, GNNExplainer (Ying et al., 2019). Because our model relies on a parametrized erasure function rather than an individual per-edge choice, we can address this issue by *amortizing* parameter learning over a training dataset through a process similar to the readout bottleneck introduced in (Schulz et al., 2020). As we demonstrate in Section 6.4, this strategy avoids hindsight bias.

Our primary contribution in this chapter is the introduction of GraphMask, a novel interpretation technique for GNNs, potentially applicable to any end-to-end neural model with a GNN as a component. We use experiments on synthetic data to demonstrate the shortcomings of the closest existing methods, and we show how our method addresses those shortcomings and improves faithfulness. We furthermore use GraphMask to analyse GNN models for two NLP tasks: semantic role labeling (Marcheggiani & Titov, 2017) and multi-hop question answering (De Cao et al., 2019). We publish the code for our experiments at https://github.com/MichSchli/GraphMask.

## 6.2 Background

In recent years, the high performance of neural network models has resulted in applications to a wide variety of different domains. While the technological artefacts resulting from this process have enabled a variety of beneficial systems, they have also, in many cases, created unintended, often harmful, and sometimes very surprising consequences (O'Neil, 2016). A potential remedy to this problem is *model interpretability* – that is, the deliberate crafting of techniques that enable human analysts to understand the behaviour of models.

Model interpretability is a diverse concept with several, distinct meanings. Different papers have suggested different conceptions of what makes machine learning artefacts interpretable. Suggestions include models which users can trust to follow the expectations of some underlying, well-understood paradigm (Ribeiro et al., 2016b), models whose underlying mechanistic function can be grasped fully by human experts (Lou et al., 2013), models which uncover and present causal relationships in data (Athey & Imbens, 2015), or models which – like human experts – can convincingly and faithfully explain their behaviour (Ridgeway et al., 1998).[2] In this paper, we focus on explanations; as such, we use the concepts *interpretability* and *explainability* interchangeably. We refer the reader to Lipton (2016) for a more thorough discussion.

Following Lipton (2016), there are two different schools of thought regarding how explanations for model behaviour should come about. Models can be constructed so that they are inherently *transparent*, or models can be treated as black-box systems and paired with secondary techniques to generate *post-hoc explanations*. For GNNs, there are several popular attempts to employ various gate- or attention-functions to build inherently interpretable models (Veličković et al., 2018; Neil et al., 2018; Xie & Grossman, 2018). Unfortunately, the performance of these models is often subpar compared to models developed purely for performance, and they are therefore seldomly employed in real-world situations where the need for interpretability is the greatest.

The concept of an *explanation* is somewhat nebulous, and there is no answer which is satisfactory for every task. Should an explanation describe the relative importance of weights? Of input features? Should it give causal relations between phenomena in the modelled domain, or is it enough to point towards *which* input elements result in a given decision?

Prior work on interpretability for GNNs focuses on one particular component of the input space – the graph itself. The interpretable GNN variants presented in e.g. Veličković et al. (2018), Neil et al. (2018), and Xie and Grossman (2018) all return explanations where scalar gates $0 \leq \sigma \leq 1$ represents the relative importance of messages. Previous approaches to post-hoc interpretability similarly focus on assigning continuous scores to measure the relative importance of vertex features and messages (Pope et al., 2019; Xie & Lu, 2019; Schwarzenberg et al., 2019; Baldassarre & Azizpour, 2019; Schnake et al., 2020). We choose to follow this line of thought, and focus on determining the relative importance of each message sent for a given example. However, as previously discussed, we restrict

[2] Note that these concepts overlap – models with mechanistic functions that are simple enough that humans can directly follow their reasoning do inherently also present *explanations* for their behaviour, and the presence of faithful explanations engenders user trust.

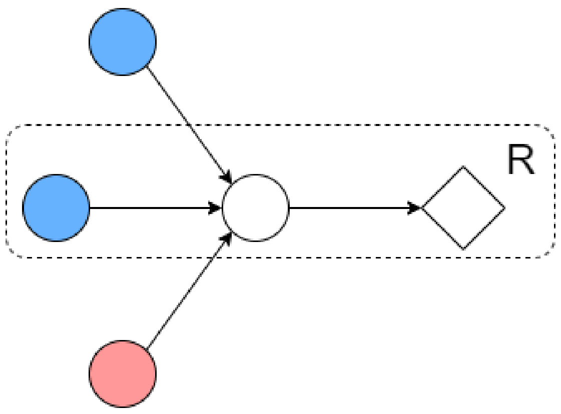


Figure 6.2: A degenerate rationale for the task of counting whether the graph contains more red or blue vertices, with predictions being made on the basis of a GNN encoding of the square vertex. The explanation $R$ contains a single blue vertex and no red vertices. A perfectly optimized underlying model, however, would count *all* red and blue vertices to make predictions.

ourselves to generating binary rather than continuous scores for edges – this more restricted form of the task is also known as *rationale generation*. We focus on generating *message-wise explanations*, which we define as follows:

**6.2.1.** Definition. A *message-wise explanation* of a prediction $\hat{y}$ for a graph neural network over a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{R}\}$ is a function $f_{\hat{y}} : \mathcal{E} \times K \to \mathbb{R}$ where $f(e, k)$ represents the relative importance of the edge $e$ at layer $k$.

Given a technique for generating explanations, it is important to understand whether the importance scores assigned to each edge actually reflect the behaviour of the analysed model. The degree to which an explanation reflects the underlying behaviour is referred to as *faithfulness*. Explanations that match the underlying behaviour are accordingly termed *faithful*, while explanations that do not accurately represent model behaviour are termed *degenerate*. Faithfulness has been the topic of several recent papers, with multiple definitions of the concept (Yu et al., 2019; Jacovi & Goldberg, 2020). We refer to Jacovi and Goldberg (2020) for a detailed discussion. It is difficult to quantifiably measure the faithfulness of any given technique. However, researchers can rely on counterexamples to test whether explanations produced for well-known problems are faithful (see e.g. Wiegreffe and Pinter (2019)).

The central problem of faithfulness is that real-world models are complex – we seek to design interpretability techniques to understand their behaviour, but without understanding their behaviour, we cannot evaluate and compare the relative performance of any given technique. In the work which this chapter is based on (Schlichtkrull et al., 2020b) as well as in De Cao et al. (2020), we attempt to address this problem by introducing simple, synthetic tasks for which there is only one possible way for perfectly optimised models to behave. As such, if a model perfectly solves the task, it must follow a predefined gold standard. We test our technique following this approach in Section 6.4. We discuss faithfulness and our approach to the concept in further depth in Section 6.8, relating our experiments to the literature.

## 6.2.1 The Hard Concrete Distribution

Our proposed interpretability technique represents a differentiable form of rationale generation, inspired by subset erasure. To ensure binary attribution scores, we need a probability distribution that allows us to learn to sample exact values of either 0 or 1. At the same time, this distribution must be differentiable to avoid training with REINFORCE (Williams, 1992) or biased straight-through estimators (Maddison et al., 2017; Jang et al., 2017).

To this end, we employ the hard concrete distribution introduced in Louizos et al. (2018). This distribution assigns density to continuous outcomes in the open interval $(0, 1)$, and non-zero mass to exactly 0 and 1. As this can be done via differentiable reparameterization (Kingma & Welling, 2014; Rezende et al., 2014), we can employ gradient optimisation to learn the parameters of the hard concrete distribution.

The concrete distribution was initially introduced to facilitate regularisation of sampled values with the $L_0$ loss, an intuitively simple strategy for prioritising the sampling of exact zeros.

**6.2.2.** Definition. Given a hypothesis defined through a set of parameters $\theta$, the $L_0$ loss of that hypothesis is

$$\|\theta_0\| = \sum_{j=1}^{|\theta|} \mathbf{1}_{[\mathbb{R}\neq 0]}(\theta_j)$$

Through the reparametrization trick, the hard concrete distribution allows us to express the $L_0$ loss as an expectation

$$\sum_{j=1}^{|\theta|} \mathbf{1}_{[\mathbb{R}\neq 0]}(\theta_j) = \sum_{j=1}^{|\theta|} \mathbb{E}_{p_\pi(\theta_j)}\left[\theta_j \neq 0\right] \ , \tag{6.1}$$

where $\pi$ are the parameters of the hard concrete distribution. The gradient for this expectation can then be estimated via Monte Carlo sampling without the need for REINFORCE and without introducing biases.

A Hard Concrete distribution (also known as a stretched and rectified Binary Concrete) distribution is obtained by applying an affine transformation to the Binary Concrete distribution (Maddison et al., 2017; Jang et al., 2017) and rectifying its samples in the interval $[0, 1]$. Like the binary concrete, the hard concrete is parameterised by a location parameter $\gamma \in \mathbb{R}$ and temperature parameter $\tau \in \mathbb{R}_{>0}$. The distribution is then stretched, and samples are rectified within the interval $[0, 1]$.

**6.2.3.** Definition. The Hard Concrete distribution is a distribution parameterised by a location parameter $\gamma \in \mathbb{R}$ and temperature parameter $\tau \in \mathbb{R}_{>0}$. Given a uniformly distributed random variable $u \sim \mathcal{U}(0, 1)$, samples from a Hard Concrete distribution can be obtained as

$$\begin{aligned} s &= \sigma\left(\left(\log u - \log(1-u) + \gamma\right)/\tau\right) \\ z &= \min\left(1, \max\left(0, s \cdot (l - r) + r\right)\right) \end{aligned}$$

where $\sigma$ is the Sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$.

We refer the reader to Louizos et al. (2018) for further discussion of this distribution, and specifically to Appendix B for information about the density of the resulting distribution and its cumulative density function.

## 6.3 Methods

To ensure that our interpretation framework is generally applicable to any graph neural network, we base it on the full graph message passing definition introduced in Section 2.2 (see Definitions 2.2.2 and 2.2.5). Recall that the propagation of information through a graph neural network – regardless of the choice of aggregation function – can be expressed through a message passing step, describing information flow from vertex along an edge $e = (u, r, v, \delta)$ at layer $k$:

$$m_e^{(k)} = M_\theta^{(k)}\left(h_u^{(k-1)}, h_v^{(k-1)}, r, \delta\right).$$

Our goal is to detect which messages $m_e^{(k)}$ at layer $k$ can be ignored without affecting model predictions. We refer to these messages and the corresponding edges (at that specific layer) as *superfluous*. Conceptually, our aim is then to identify these superfluous edges given a trained graph neural network.

GNNs can be highly sensitive to changes in the graph structure. A GNN trained on graphs where all vertices $v$ have degree $d(v) \gg n$ for some integer $n$ may become unstable if applied to a graph where some vertices have degree $d(v) \ll n$. This is because the model has not seen examples with low-degree vertices during training, and as such may give strange or nonsensical predictions for these examples. Hence, dropping edges without affecting predictions can be difficult.

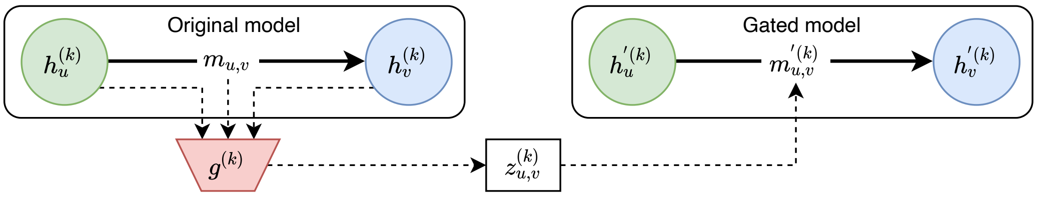


Figure 6.3: GraphMask: Vertex hidden states and messages at layer $k$ (left) are fed to a classifier $g$ that predicts a mask $z^{(\ell)}$. We use this to mask the messages of the $k$th layer and re-compute the forward pass with modified node states (right). The classifier $g$ is trained to mask as many hidden states as possible without changing the output of the gated model.

Nevertheless, many edges in that graph may be superfluous for all purposes other than maintaining neighbourhood statistics such as degree or local linearity. For that reason, it is not enough to search for edges which can be *dropped* – instead, we search for edges which, through a binary choice $z_e^{(k)}$, can be replaced with a learned baseline $b^{(k)}$.

**6.3.1.** Definition. A mock-message $\tilde{m}_e^{(k)}$ for the edge $e = (u, r, v, \delta)$ is obtained by replacing the original message $m_e^{(k)}$ by a learned baseline $b^{(k)}$ through some binary indicator $z_e^{(k)} \in \{0, 1\}$:

$$\tilde{m}_e^{(k)} = z_e^{(k)} \cdot m_e^{(k)} + b^{(k)} \cdot (1 - z_e^{(k)})$$

By learning a constant baseline for all edges within a given layer, our method can mimic neighbourhood statistics without actually propagating information. We use superfluity to denote messages and edges which can be replaced with such “mock-messages”, rather than the less informative set of edges that can be dropped entirely.

Conceptually, the search for a subset that generates the same prediction can be understood as a form of subset erasure (Li et al., 2016a; Feng et al., 2018). Unfortunately, erasure breaks with the desiderata we proposed in two important ways. First, since it involves searching over all the possible candidates that could be dropped, it is not *tractable*. Second, since the search happens individually for each example, there is a danger of hindsight bias. That is, the search algorithm finds a minimal set of features that could produce the given prediction, but which is not *faithful* to how the model originally behaved (as confirmed in our experiments, Section 6.4). To overcome those issues, we compute $z_e^{(k)}$ through a simple function, learned once for every task across data points:

**6.3.2.** Definition. Given vertex embeddings $h_u^{(k-1)}, h_v^{(k-1)} \in \mathbb{R}^d$ of vertices

$u$ and $v$ along with a message $m_e^{(k)} \in \mathbb{R}^d$ for an edge $e = (u, r, v, \delta)$, the GraphMask indicator function is a function $g_\pi : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d \rightarrow \{0, 1\}$ with learned parameters $\pi$:

$$z_e^{(k)} = g_\pi(h_u^{(k-1)}, h_v^{(k-1)}, m_e^{(k)})$$

Learning one set of parameters $\pi$ for an entire dataset is helpful in two ways. First, it allows us to use the training set for the original model to choose $\pi$, ensuring that the examples we test GraphMask on are not seen during training. Second, each $z_e^{(k)}$ for a given example is computed relying only on information also available to the original model when computing the corresponding message $m_e^{(k)}$. Crucially, neither the original prediction nor the label for that example can be used to compute $z_e^{(k)}$; the explainer has no lookahead.[3] These two aspects, by design, work to prevent hindsight bias. We refer to this strategy as *amortization*.

The alternative to this strategy is to choose the parameters $\pi$ independently for each gate, without any parameter sharing across gates. In that case, optimization would be performed directly on the analyzed (i.e. test) example. We refer to this strategy as the *non-amortized* version of GraphMask. In that case it would be wasteful to use a neural network $g_\pi(h_u^{(k)}, h_v^{(k)}, m_e^{(k)})$ to compute the gate for $m_e^{(k)}$. Instead, we directly optimize the parameters of the Hard Concrete relaxation, discussed in Section 6.2.1. We demonstrate in Section 6.4 that this version of GraphMask, unlike the amortized approach, is susceptible to hindsight bias.

We compute the parameters $\pi$ through a simple multilayer perceptron. We first derive a representation $q_e^{(k)}$ of an edge $e$ at layer $k$ simply through concatenation:

$$q_e^{(k)} = [h_u^{(k)}, h_v^{(k)}, m_e^{(k)}] \tag{6.2}$$

We then compute the scalar location parameters $\gamma_e^{(k)}$ for the hard concrete distribution based on $q_e^{(k)}$:

$$\gamma_e^{(k)} = W_2^{(k)} \text{ReLU}(\text{LN}(W_1^{(k)} q_e^{(k)})) \tag{6.3}$$

where LN represents Layer Normalization.

As we have previously discussed (see Section 2.2), it is common[4] to replace the full graph message passing framework with the faster – but slightly less expressive – Graph Convolutional Networks (GCNs). In that case, aggregation and message passing is done through matrix multiplication between the vertex embeddings matrix $H^{(k)}$ and a (normalized, relation-specific) adjacency matrix $\hat{A}_r$:

$$H^{(k)} = \hat{A}_r H^{(k-1)} W^{(k)}$$

[3] The readout function in Schulz et al. (2020) violates this constraint.

[4] An example of this is the question answering model by De Cao et al. (2019), which we analyse with GraphMask in Section 6.5.

Applying the computation of $q_e^{(k)}$ from Equation 6.2 within that scheme would be prohibitively expensive, as $q_e^{(k)}$ would need to be computed for every possible combination of $u$ and $v$ rather than just those actually connected by edges. The complexity would as such rise to $O(|\mathcal{V}|^2)$ rather than $O(|\mathcal{V}|+|\mathcal{E}|)$, which for large graphs can be problematic.

To apply our method in such cases, we also develop a faster alternative computation of $\gamma_e^{(k)}$ based on a bilinear product. In this case, rather than enumerating all possible messages, we rely purely on the source and target vertex embeddings $h_u^{(k)}$ and $h_v^{(k)}$. Taking inspiration from our previous work on R-GCN (see Chapter 3), we compute an alternative matrix-form $\hat{\gamma}^{(k)}$ as:

$$\hat{\gamma}^{(k)} = \hat{W}_r^{(k)} \mathrm{ReLU}(\mathrm{LN}(\hat{W}_1^{(k)} H^{(k)})) H^{(k)\ \top} \tag{6.4}$$

where $\hat{W}_r^{(k)}$ is unique to the relation $r$. We sample relation-specific matrix-form gates $\hat{Z}_r^{(k)}$, and apply these using an alternate – but equivalent – version of Definition 6.3.1 to derive a representation matrix $\widetilde{H}^{(k)}$ for the vertices in the masked model:

$$\sum_r (\hat{Z}_r^{(k)} \hat{A}_r) H^{(k-1)} W^{(k)} + ((J - \hat{Z}_r^{(k)})\hat{A}_r) B^{(k)} \tag{6.5}$$

where $J$ represents the all-one matrix. In our experiments, we rely on the adjacency-list formulation for SRL in Section 6.6 and the adjacency-matrix formulation for QA in Section 6.5.

After $g$ is trained, to analyse a data point with GraphMask, we first execute the original model over that data point to obtain $h_u^{(k)}$, $h_v^{(k)}$, and $m_e^{(k)}$. We then compute gates for every edge at every layer, and execute a masked version of the model as shown in Figure 6.3. For the first layer, the messages of the original model are gated according to Definition 6.3.1 to obtain mock-messages. For subsequent layers, we aggregate mock-messages using Definition 2.2.5 to obtain vertex embeddings $h_v'^{(k)}$, which we then use to obtain the next set of mock-messages.

Note that the only learned parameters of GraphMask are the parameters $\pi$ of the erasure function and the learned baseline vectors $b^{(1)}, \ldots, b^{(k)}$ – the parameters of the original model are kept constant. As long as the prediction relying on the sparsified graph is the same as when using the original one, we can interpret masked messages as superfluous.

## 6.3.1 Parameter estimation

Given a GNN $f$ of $L$ layers, a graph $\mathcal{G}$, and input embeddings $\mathcal{X}$ (e.g., initial node vectors or additional inputs), our task is to identify a set $\mathcal{G}_S = \{\mathcal{G}_S^{(1)}, \ldots, \mathcal{G}_S^{(L)}\}$ of informative sub-graphs such that $\mathcal{G}_S^{(k)} \subseteq \mathcal{G}\ \forall k \in 1, \ldots, L$. We search for a graph with the minimal number of edges while maintaining $f(\mathcal{G}_S, \mathcal{X}) \approx f(\mathcal{G}, \mathcal{X})$.[5] We

[5] With $f(\mathcal{G}_S, \mathcal{X})$ we denote a forward pass where for each layer the graph may vary where for $f(\mathcal{G}, \mathcal{X})$ the graph $\mathcal{G}$ is the same across layers.

can cast this, quite naturally, in the language of constrained optimization and employ a method that enables gradient descent such as Lagrangian relaxation. In general, however, it is not possible to guarantee equality between $f(\mathcal{G},\mathcal{X})$ and $f(\mathcal{G}_S,\mathcal{X})$ since $f$ is a smooth function, and as therefore a minimal change in its input cannot produce the exact same output. As such, we introduce i) a divergence $\mathrm{D}_\star[f(\mathcal{G},\mathcal{X})\|f(\mathcal{G}_S,\mathcal{X})]$ to measure how much the two outputs differ, and ii) a tolerance level $\beta \in \mathbb{R}_{>0}$ within which differences are regarded as acceptable. The choice of $\mathrm{D}_\star$ depends on the structure of the output of the original model. A practical way to minimize the number of non-zeros predicted by $g$ is minimizing the $L_0$ 'norm' (i.e., the total number of edges that are not masked). Accordingly, we formally define our objective function as:

**6.3.3.** Definition. The GraphMask objective is defined as:

$$\max_{\lambda}\min_{\pi,b}\sum_{\mathcal{G},\mathcal{X}\in\mathcal{D}}\left(\sum_{k=1}^{L}\sum_{(u,r,v,\delta)\in\mathcal{E}}\mathbf{1}_{[\mathbb{R}\neq 0]}(z_e^{(k)})\right)+\lambda\left(\mathrm{D}_\star[f(\mathcal{G},\mathcal{X})\|f(\mathcal{G}_S,\mathcal{X})]-\beta\right)$$

where $\mathbf{1}$ is the indicator function and $\lambda \in \mathbb{R}_{\geq 0}$ denotes the Lagrange multiplier.

Unfortunately, this objective is not differentiable, since i) $L_0$ is discontinuous and has zero derivatives almost everywhere and ii) outputting a binary value needs a discontinuous activation, e.g. the step function. This is problematic for gradient-based optimization. A solution is to address the objective in expectation and employ either score function estimation i.e. REINFORCE (Williams, 1992), biased straight-through estimators (Maddison et al., 2017; Jang et al., 2017), or sparse relaxation to binary variables (Louizos et al., 2018; Bastings et al., 2019). We choose the latter since it exhibits low variance compared to REINFORCE and is an unbiased estimator.

As we have mentioned in Section 6.2.1, we use the Hard Concrete distribution, a mixed discrete-continuous distribution on the closed interval $[0,1]$. This distribution assigns a non-zero probability to exactly zero while it also admits continuous outcomes in the unit interval. An unbiased and low variance gradient can be computed via the *reparameterization trick* (Kingma & Welling, 2014). We refer to Section 6.2.1 and further to Louizos et al. (2018) for details. Attribution scores correspond to the expectation of sampling non-zero masks, since any non-zero value can leak information. In our experiments GraphMask converges to a distribution where scores in expectation assume near-binary values. The $L_0$ loss in Equation 6.3.3 as such becomes an expectation:

$$\sum_{k=1}^{L}\sum_{(u,r,v,\delta)\in\mathcal{E}}\mathbf{1}_{[\mathbb{R}\neq 0]}(z_e^{(k)})=\sum_{k=1}^{L}\sum_{(u,r,v,\delta)\in\mathcal{E}}\mathbb{E}_{p_\pi(z_e^{(k)}|\mathcal{G},\mathcal{X})}\left[z_e^{(k)}\neq 0\right]\ , \tag{6.6}$$

In our experiments, we found a constant temperature $\tau = 1/3$ to work well.

Message specific location parameters $\gamma_e^{(k)}$ are computed as specified in the previous section. We found it practical to shift the initial location using a bias $c = 2$, e.g. rather than directly using $\gamma_e^{(k)}$ in Definition 6.2.3 we substitute $\gamma_e^{(k)} + c$. This places the model in an initial state where all gates are open, which is essential for learning.

When training GraphMask, we found it helpful to employ a regime wherein gates are progressively added to layers, starting from the top. For a model with $K$ layers, we begin by adding gates only for layer $k$, and train the parameters for these gates for $\delta$ iterations. We then add gates for the next layer $k-1$, train all sets of gates for another $\delta$ iterations, and continue downwards in this manner.

We found it necessary to use separate optimizers and learning for the Lagrangian $\lambda$ parameter and for the parameters of GraphMask. Thus, we employ Adam (Kingma & Ba, 2015) with initial learning rate $1e-4$ for GraphMask, and RMSProp (Tieleman & Hinton, 2012) with learning rate $1e-2$ for $\lambda$. For the tolerance parameter $\beta$, we found $\beta = 0.03$ to perform well for all tasks.

As GraphMask executes the model which it analyses, training- and runtime depends on the complexity of that model. At training time, GraphMask requires a single forward pass to compute gate values, followed by a backward pass through the sparsified model. Thus, every iteration requires at most twice the computation time as an equivalent iteration using the investigated model.

### 6.3.2 Integrated Gradients for Graphs

In our experiments in subsequent sections, we compare GraphMask to several different techniques from the literature. Among these, we include a variant of integrated gradients (Sundararajan et al., 2017). Applying this interpretability technique to identify relevant edges in graphs is nontrivial, and as such we introduce our approach here. We refer the reader to Sundararajan et al. (2017) for a detailed discussion of the original approach. Integrated gradients considers the straight-line path between a feature $x$ and a baseline $x'$, and assigns scores by accumulating gradients at all points along that path.

**6.3.4.** Definition. The attribution score of a feature $x_i$ with baseline $x'_i$ for a neural network $F$ is defined as:

$$\text{IntegratedGradients}_i(x) = (x_i - x'_i) \int\limits_{\alpha=0}^{1} \frac{\delta F(x' + \alpha(x - x'))}{\delta x_i}$$

For graphs, we take the simplistic approach of defining scalar variables $\hat{z}_e^k$ by which the message on the edge $e = (u, r, v, \delta)$ at layer $k$ is multiplied, and interpolating between $\hat{z}_e^k = 1$ and $\hat{z}_e^k = 0$. We compute attribution scores through integrated gradients for each $\hat{z}_e^k$, using 0 as a baseline. The attribution score of $e$

is then the integrated gradient score of the pseudo-gate $\hat{z}_e^k$. That is, we assume that the problem can be modelled as interpolating between edges being "fully present" and "fully absent" through "partially present" states.

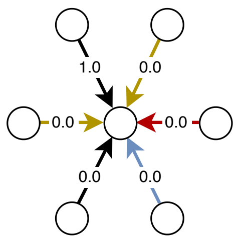
We note that it is nontrivial to extend this approach to multi-layer GNNs, since "partially present" edges in upper layers affect gradient flow and thus attribution to lower layers during interpolation. For the simple single-layer model we apply in Section 6.4 this does not present any complications. However, this flaw greatly hinders interpretability for multi-layer graph neural networks.

Furthermore, as we have previously discussed, the zero-vector may not be an appropriate baseline for general GNNs as it changes the degree statistics of the underlying graph. This could harm the performance of integrated gradients (Sturmfels et al., 2020). For the experiment in Sections 6.4, we have constructed the synthetic dataset such that any highly performant model *must* develop robustness with regards to changing degree statistics, and as such we avoid this complication. For real-world data, this would not necessarily be the case.

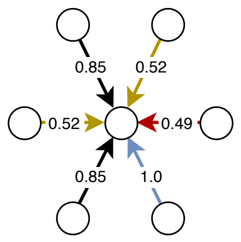
### 6.3.3 Information Bottleneck for Graphs

A related attribution technique to ours is the information bottleneck approach proposed by Schulz et al. (2020). Their technique involves computing individual soft sigmoidal gates $\xi_{hi}$ for each dimension $i$ of the hidden state $h$ of a CNN to attribute importance. Gated vectors are replaced with sampled values from a Gaussian distribution with mean and variance computed over all examples in the dataset for each hidden state. The KL divergence between the dataset distribution and the distribution obtained by interpolating between that distribution and the observed value through the gate is used as regularization to promote sparsity.

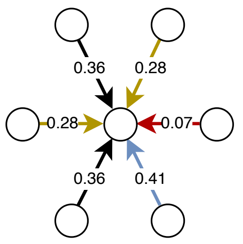
In Section 6.4, we include also results for an adaptation of Schulz et al. (2020) to the problem of attributing importance to messages in a graph neural network. To apply the information bottleneck approach for our setting, we do the following. First, instead of individual gates $\xi_{mi}$ for each dimension of the message $m$, we use a single gate $\xi_m$. We use their Readout Bottleneck approach, which can be seen as a parallel to our amortization strategy (that is, our strategy of learning the parameters $\pi$ of our erasure function from Equation 6.3.2 on the entire training set rather than individually for each test example). We predict logits for each gate by conditioning on the source and target embeddings, as well as on the message itself, similar to how we compute parameters for GraphMask. This contrasts with the original approach of using $1x1$-convolutions over the depth dimension – conditioning on "downstream" messages in the GNN could cause hindsight bias. Training is done with the KL-divergence based loss introduced in Schulz et al. (2020). Finally, we compute the mean and variance of the Gaussian noise used as a baseline (and for the loss) in their approach using all messages in the same layer over the entire training dataset. We found using the entire training data to collect statistics to work better than collecting statistics individually per example.



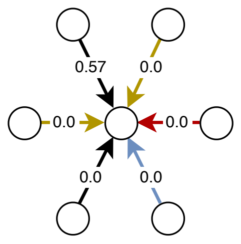
(a) Erasure search



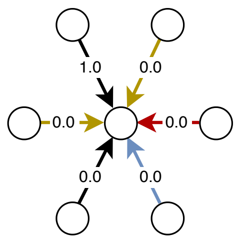
(b) Integrated Gradients



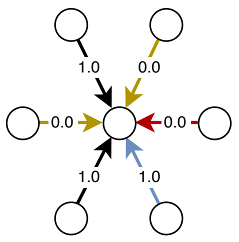
(c) Information Bottleneck



(d) GNNExplainer



(e) Ours (non-amortized)



(f) Ours (amortized)

Figure 6.4: In our synthetic task, a model predicts whether there are more black edges (→) than blue edges. (→). Erasure search, GNNExplainer, and non-amortized GraphMask overfit by retaining only a single black edge (top left). Integrated gradients and the information bottleneck approach give unsatisfying results as all edges have attribution. Only amortized GraphMask correctly assigns attribution to and only to black and blue edges.

## 6.4 Avoiding Degeneration

Interpretability techniques searching for minimal sets of useful features are typically optimized to find the smallest such set for every example (Li et al., 2016a; Ying et al., 2019). However, the smallest set of features that *could* lead a model to produce a certain prediction is not necessarily that which the model *actually* relied on. That solution may be a degenerate – that is, not faithful – explanation. If the desired result is known and an extensive search is employed, the optimizer can exploit model behaviour to find minimal (degenerate) explanations through a process similar to an adversarial attack (Kurakin et al., 2016). Knowing the desired outcome, the optimizer can pick very specific combinations of features to produce it. We refer to this problem as *hindsight bias*.

For the real NLP problems we address in Sections 6.5 and 6.6, the models and data are too complex for a human gold standard for faithfulness to be defined (Jacovi & Goldberg, 2020). To illustrate the hindsight bias problem and demonstrate how GraphMask overcomes it, we therefore introduce a synthetic task for which a clearly defined ground-truth attribution is known.

The task is defined as follows: a star graph $\mathcal{G}$ with a single centroid vertex $v_0$, leaf vertices $v_1, ..., v_n$, and edges $(v_1, v_0), ..., (v_n, v_0)$ is given, and every edge $(u, v)$ is assigned one of several colours $c_{u,v} \in C$. Then, given a query $\langle x, y \rangle \in C \times C$, the task is to predict whether the number of edges assigned $x$ is greater than number of edges assigned $y$. We generate a total of 6.000 examples randomly with 6 to 12 leaves, using 5.000 for training and 500 respectively to validate and test. We solve the task with a simple one-layer R-GCN (see Chapter 3) with vertex embeddings $h^{(0)}$ initialized as the concatenation of a one-hot-encoding of $x$ and a one-hot-encoding of $y$.

The trained model perfectly classifies every example. We know precisely which edges are useful for a given example – those which match the two colours being counted in that example. The GNN *must* count all instances of both to compute the maximum, and no other edges should affect the prediction. We can therefore define a gold standard for faithfulness on this basis: for $x > y$, all edges of type $x$ and $y$ should be retained, and all others should be discarded.

Since the task is simple counting, the GNN can be made to predict that $x > y$ by dropping everything other than a single edge of type $x$ (since $1 > 0$). Conversely, the model can be forced to predict $x \not> y$ by dropping *every* edge (since $0 \not> 0$). If the optimizer is simply searching for a minimal number of edges to produce a desired outcome, it can therefore always find a solution with at most one edge. However, that solution would not be *faithful*.

For this simple task, we can compute empirical estimates of faithfulness using the aforementioned gold standard. In Table 6.1, we compare GraphMask to four baselines: subset erasure search (Li et al., 2016a), integrated gradients (Sundararajan et al., 2017), an information bottleneck approach (Schulz et al., 2020), and GNNExplainer (Ying et al., 2019). Neither integrated gradients nor the information bottleneck approach were designed for graphs, and as such we adapt them for this setting (see Sections 6.3.2 and 6.3.3 for details). Since GNNExplainer and the information bottleneck strategy do not make hard predictions, we define any gate $\sigma_i$ where $\sigma_i > t$ for some threshold $t$ as open, and all other gates as closed. Similarly, for integrated gradients, we normalize attributions to the interval $[-1; 1]$, take the absolute value, and apply a threshold $t$. We select $t \in \{0.1, ..., 0.9\}$ to maximize $\mathrm{F}_1$ score on validation data.[6]

Only the amortized version of our method approximately replicates the gold

[6] It is important to note that selecting a threshold $t$ to distinguish useful and superfluous edges sharply is only possible because our synthetic task provides a gold standard. Otherwise, a validation set for faithfulness could not be constructed. Choosing an appropriate threshold to maximize faithfulness on the validation set would therefore not be possible on real data.

| **Method** | **Prec.** | **Recall** | **F$_1$** |
|---|---|---|---|
| Erasure search* | 100.0 | 16.7 | 28.6 |
| Integrated Gradients | 88.3 | 93.5 | 90.8 |
| Information Bottleneck | 55.3 | 51.5 | 52.6 |
| GNNExplainer | 100.0 | 16.8 | 28.7 |
| Ours (non-amortized) | 96.7 | 26.2 | 41.2 |
| Ours (amortized) | 98.8 | 100.0 | **99.4** |

Table 6.1: Comparison using the faithfulness gold standard on the toy task. *as in Li et al. (2016a).

standard. In fact, erasure search, GNNExplainer, and non-amortized GraphMask recall only a fraction of the non-superfluous edges. Visually inspecting the assigned scores (see Figure 6.4), we see that erasure search, GNNExplainer, and non-amortized GraphMask all reach the same low-penalty solution with perfect model performance – when $x > y$ a single edge of type $x$ is preserved, and when $x \not> y$ all edges are dropped. This is a degenerate interpretation, and it occurs due to hindsight bias.

With amortization, GraphMask must find a set of parameters that produces good solutions purely based on the sent messages; it is impossible to "look ahead" to determine whether a message in the future will be useful. As such, overfitting due to hindsight bias does not occur – decisions are made irrespective of hindsight, and degeneration is prevented.

The example also demonstrates a flaw of integrated gradients and the information bottleneck approach on this task. Edges in relevant colours are often the highest scored, but the magnitudes of the scalar attribution scores vary greatly across examples with different numbers of edges. As such, a single threshold $t$ cannot be defined to always distinguish between useful and superfluous edges even on this simple task. On real data, where a development set for selecting $t$ would not be available, it would be extremely difficult to interpret these continuous attribution scores.

## 6.5 Question Answering

We now apply GraphMask (amortized) to analyze predictions for a real model. In this setting, due to the complexity of real tasks, no human gold standard for attribution can be constructed (Jacovi & Goldberg, 2020). We choose the GNN-based model for multi-hop QA presented in De Cao et al. (2019), evaluated on WikiHop (Welbl et al., 2018). The task is, given a question and a set of context documents, to find the entity within the context that best answers the question. Nodes in the GNN graph correspond to mentions of entities within the question

| Retained edges | Acc. |
|---|---|
| 100% (Orig. model) | 59.0 |
| 27% (GraphMask) | 58.6 |
| 20.25% | 55.2 |
| 13.5% | 52.8 |
| 6.25% | 47.7 |
| 0% | 45.2 |

Table 6.2: Performance of the question answering model using the original input graphs, using the subgraphs retained after masking with GraphMask, and using only a randomly selected 0/25/50/75/100% of the edges retained after masking with GraphMask. Dropping the edges marked superfluous by our technique does not impact performance; dropping the remaining edges, even if only a randomly selected 25% of them, significantly hurts the model.

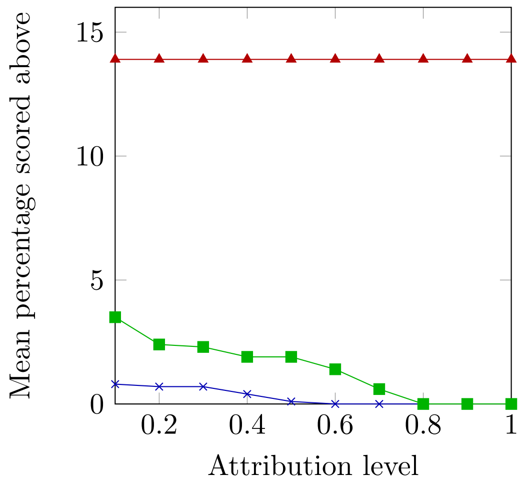
and context, and four types of edges between those are introduced: string match (MATCH), document-level co-occurrence (DOC-BASED), coreference resolution (COREF), and, finally, the absence of any other edge (COMPLEMENT).

The model consists of a two-layer BiLSTM reading the question, and three layers of R-GCN (Schlichtkrull et al., 2018) with shared parameters. Node representations at the bottom layer are obtained by concatenating the question representation to embeddings for the mention in question. Here, we focus on their GloVe-based model. Finally, the mention representations are combined into entity representations through max-pooling.

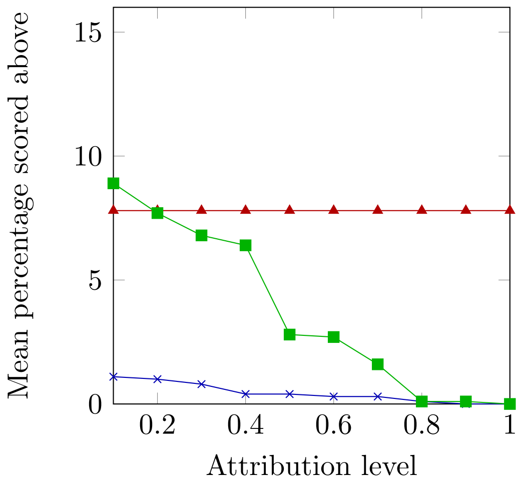
GraphMask replicates the performance of the original model with a performance change of $-0.4\%$ accuracy. 27% of edges are retained, with the majority occurring in the bottom layer (see also Table 6.5). To ensure that the choice of superfluous edges is not just a consequence of the random seed, i.e. to verify the stability of our method, we compute Fleiss' Kappa scores between each individual measurement of $z_{u,v}^{(k)}$ across 5 different seeds. We find high agreement with $\kappa = 0.65$. Dropping just a random 25% of these retained edges greatly harms performance (see Table 6.2).

For comparison, if we do not amortize to provide resilience against hindsight bias, the retained edges are different. In that case, only 0.4% of retained edges occur in the bottom layer, compared to 91.0% in the top layer. Similarly, GNNExplainer and Integrated Gradients also assign only a fraction of their total attribution score to the bottom layer, respectively 4.3% and 11.3% of their total attribution score – see Figure 6.5.

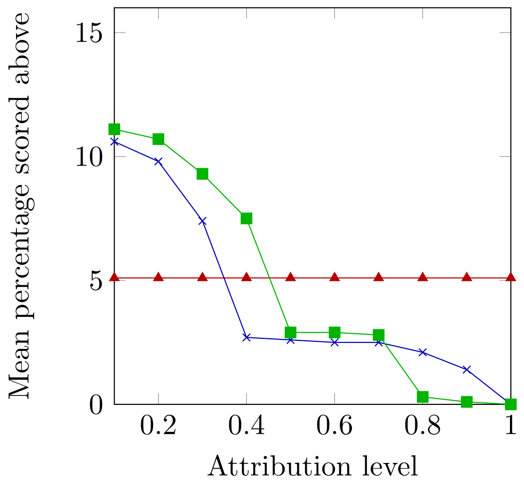
As we see in Table 6.3, dropping the bottom layer yields a much larger performance decrease (-26%) than dropping the top layer (-7%). This is at odds with the predicted attributions. For GNNExplainer, manual inspection reveals this to be a product of hindsight bias. Very specific configurations of top-layer



(a) Layer 0



(b) Layer 1



(c) Layer 2

Figure 6.5: Mean percentage of messages assigned attribution scores above a certain level in the question answering model, separated by layer. We report scores for GNNExplainer (—×—), Integrated Gradients (—■—), and GraphMask (—▲—).

| Layers discarded | Acc. |
|---|---|
| Full model | 59.0 |
| - layer 0 | 33.1 |
| - layer 1 | 41.6 |
| - layer 2 | 52.0 |

Table 6.3: Performance of the question answering model with all edges in each individual GNN layer dropped.

| Model | $k = 0$ | $k = 1$ | $k = 2$ |
|---|---|---|---|
| GNNExplainer | 4.3 | 11.9 | 83.8 |
| Int. Grad. | 11.3 | 33.0 | 55.7 |
| GraphMask | 51.6 | 28.8 | 19.6 |

Table 6.4: Mean percentage of the total attribution score allocated to each layer for the question answering model, according to GNNExplainer, Integrated Gradients, and GraphMask.

edges (in most cases, retaining only edges where the predicted answer is the target) generate the same predictions as the original model. This mirrors a common pathology of erasure search for QA over text, where the answer span and nothing else is selected as an explanation (Feng et al., 2018).

For Integrated Gradients, the low scoring of the bottom layer is a result of long-distance information (e.g. information from edges and vertices far from the predicted answer, which must travel through half-open pseudo-gates in the other layers to reach the predicted answer) being systematically underestimated as we discussed in Section 6.3. This prevents meaningful comparisons of attribution scores between layers.

Another approach is to compare the proportion of the total attribution score that different techniques assign to each layer; ideally, this should reflect that layer's importance. In Table 6.4, we compute the mean percentage of the total score assigned to messages in each layer. As in Figure 6.5, we see GNNExplainer and Integrated Gradients assign low levels of attribution to the bottom layer, at odds with the empirical performance loss from excluding that layer. This again indicates that the baselines are unlikely to be faithful.

### 6.5.1 Analysis

In Table 6.5, we investigate which edge types are used across the three layers of the model. De Cao et al.'s (2019) ablation test suggested that COREF edges provide marginal benefit to the model; our analysis does not entirely agree. Investigating

| **Edge Type** | **k = 0** | **k = 1** | **k = 2** |
|---|---|---|---|
| MATCH (8.1%) | 9.4% | 11.1% | 8.9% |
| DOC-BASED (13.2%) | 5.9% | 17.7% | 10.7% |
| COREF (4.2%) | 4.4% | 0% | 0% |
| COMPLEMENT (73.5%) | 31.9% | 0% | 0% |
| Total (100%) | 51.6% | 28.8% | 19.6% |

Table 6.5: Retained edges for De Cao et al.'s (2019) question answering GNN by layer ($k$) and type.

further, we see that only 2.3% of the retained COREF edges overlap with MATCH edges (compared to 32.4% for the entire dataset). In other words, the system relies on COREF edges only in harder cases not handled by the surface MATCH heuristic. The role COMPLEMENT edges play is interesting as well: this class represents the majority of non-superfluous edges in the bottom layer, but is always superfluous in subsequent layers. The model relies on an initial propagation-step across these edges, perhaps to pool context before any other inference.

De Cao et al.'s (2019) model concatenates a representation of the question to every node in the graph before running GNN. As such, one might expect edges connecting mentions of the question entity to the rest of the graph to be superfluous. This, however, is not the case – at least one such edge is retained in 92.7% of all cases, and in 84.1% of cases that edge occurs in the bottom layer. We hypothesize that the model relies on GNN to see whether other mentions share a surface form or co-occur with mentions of the question entity, and, if not, how they otherwise connect to those. To investigate this, we measure the percentage of retained edges at each layer that occur on paths originating from question entities.

We find that the proportion of non-superfluous edges which occur on paths from mentions of the question increases drastically by layer, from 11.8% at layer 0, to 42.7% at layer 1, and culminating in 73.8% in the top layer. A mention corresponding to the predicted answer is in 99.7% of examples the target of *some* retained edge. However, the chance that the predicted entity is connected to the question (72.1%) is near-identical to that of the average candidate entity (69.2%). As such, the GNN is responsible not only for propagating evidence to the predicted answer through the graph, but also for propagating evidence to alternate candidates. The majority of paths take one of two forms – a COMPLEMENT edge followed by either a MATCH or a DOC-BASED edge (22%), or a COMPLEMENT edge followed by two MATCH or DOC-BASED edges (52%). MATCH and DOC-BASED edges in the bottom layer tend to represent self-contained one-hop paths rather than being the first edge on a longer, non-superfluous path.

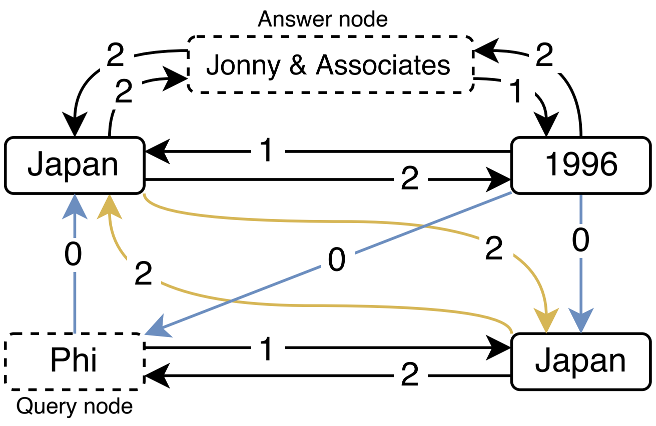


Figure 6.6: Subgraph of retained edges (21% of the original) for the question *"record_label Phi"*. → is DOC-BASED, → is COMPLEMENT, and → is MATCH where edge labels indicate in which layer GraphMask retains such edge.

Relations used by De Cao et al. (2019) are symmetric (e.g., a coreference works in both directions). A distinct feature of the subgraphs retained by GraphMask for this model is that pairs of an edge and its inverse are *both* judged to be either superfluous or non-superfluous (individually in each layer). In Figure 6.6, this can be seen for the DOC-BASED edges in layer 2 between *Japan* and *Johnny & Associates*. Indeed, 49% of retained edges in layer 0, 98% of retained edges in layer 1, and 79% of retained edges in layer 2 have their inverses also retained. In other words, "undirected" message exchange between mentions, resulting in enriched mention representations, appears crucial.

## 6.6 Semantic Role Labeling

The second model we analyse is the GNN-based SRL system of Marcheggiani and Titov (2017). The task here is to identify arguments of a given predicate and assign them to semantic roles; see the labels below the sentence in Figure 6.9. Their GNN operates over automatically predicted syntactic dependency trees, allowing for information flow in both directions between syntactic dependents and their heads. Their model encodes sentences through a stacked BiLSTM, upon which the encoded tokens are passed through two GNN-layers in the corresponding syntactic dependency tree. By analysing which dependency arcs play a role in the decision-making of the model, we can investigate which syntactic features encode semantic information not derivable by the LSTM.

We investigate both their best-performing model, which includes a BiLSTM and one layer of a GNN, and their GNN-only model.[7] For LSTM+GNN, the

[7] In Marcheggiani and Titov (2017), the best GNN-only model used three layers of GNN; with our reimplementation, a two-layer GNN performed better. Our reimplementation performed

| Retained edges | F1 |
|---|---|
| 100% (Orig. model) | 87.1 |
| 4% (GraphMask) | 86.6 |
| 3% | 83.1 |
| 2% | 74.3 |
| 1% | 68.9 |
| 0% | 63.8 |

(a) SRL: LSTM+GNN

| Retained edges | F1 |
|---|---|
| 100% (Orig. model) | 83.8 |
| 16% (GraphMask) | 83.1 |
| 12% | 74.4 |
| 8% | 66.1 |
| 4% | 58.9 |
| 0% | 56.5 |

(b) SRL: GNN-Only

Table 6.6: Performance of the two SRL models using the original input graphs, using the subgraphs retained after masking with GraphMask, and using only a randomly selected 0/25/50/75/100% of the edges retained after masking with GraphMask. Dropping the edges marked superfluous by our technique does not impact performance; dropping the remaining edges, even if only a randomly selected 25% of them, significantly hurts the model.

masked model has a minuscule performance change of $-0.62\%$ $F_1$ and retains only 4% of the messages. The GNN-only model has a similarly small performance change of $-0.79\%$ $F_1$ and retains 16% of messages. Fleiss' Kappa scores between GraphMask with 5 different seeds indicate a substantial agreement of respectively $\kappa = 0.79$ and $\kappa = 0.74$ for the full and GNN-only models.

The GNN employed by Marcheggiani and Titov (2017) uses scalar, sigmoidal gates on every message. A naive method for interpreting this model could be to employ those values. However, sigmoidal gates do not necessarily reflect the importance of individual messages to the model; they may instead provide scaling as a component in the model. On development data, the mean gate takes the value 0.16, with a standard deviation of $\sigma = 0.07$. We evaluate the model with every edge where the corresponding gate value is more than one $\sigma$ below the mean dropped, and find that performance decreases by 16.1% $F_1$ score even though only 42% of edges are removed. Thus, we see that these gates act as scaling rather than reflecting the contribution of each edge to the prediction (see also Appendix B for soft gate values for the example in Figure 6.9). This matches the intuition from Jain and Wallace (2019) that gates do not necessarily indicate attribution.

### 6.6.1 Analysis

We first investigate which dependency types the GNN relies on. We show a plot that summarizes our finding in Figure 6.7. The behaviour differs strongly for nominal and verbal predicates – NMOD dominates for nominals, whereas SBJ and OBJ play the largest roles for verbal predicates. This is unsurprising,

on par with the original.

| **Type** | **Length** | **Retained edges** | | | | |
|---|---|---|---|---|---|---|
| | | **GNN-only** | | | **LSTM+GNN** | |
| | | 0 | 1 | 2 | 0 | 1 |
| V | 1 (5755) | 0.01 | 0.99 | - | 0.01 | 0.99 |
| | 2 (1104) | 0.07 | 0.74 | 0.19 | 0.10 | 0.90 |
| | $\geq 3$ (10904) | 0.74 | 0.22 | 0.04 | 0.79 | 0.21 |
| N | 1 (3336) | 0.02 | 0.98 | - | 0.01 | 0.99 |
| | 2 (2935) | 0.30 | 0.25 | 0.45 | 0.89 | 0.11 |
| | $\geq 3$ (3251) | 0.56 | 0.32 | 0.12 | 0.73 | 0.27 |

Table 6.7: Percentages of paths with either 0, 1, or 2 edges retained, split by path length and predicate type, for the two models. For the LSTM+GNN model, at most one edge can be included per path as only a single GNN layer is employed. In parenthesis we report the supporting number of edges.

because these edges often directly connect the predicate to the predicted roles. Even where this is not the case – see *rebound* in the example in Figure 6.9 – these edges connect the predicted tokens to tokens close to the predicate, which the model can easily reach with the LSTM. Interestingly, several frequent relations (occurring in $>$ 10% of examples) are entirely superfluous – these include P, NAME, COORD, CV, CONJ, HYPH, SUFFIX, and POSTHON. For the LSTM-GNN model, we find that 88% of retained edges directly target predicted roles (e.g. *rebound*). The remaining 12% almost always target tokens that function as roles for other predicates in the same sentence (e.g. *which*).[8]

Marcheggiani and Titov's (2017) original findings suggest that the GNN is especially useful for predicting roles far removed from the predicate, where the LSTM is less reliable for propagating information. One way in which the GNN could accomplish this is by using paths in the graph; either relying on the entire path, or partially relying on the last several edges in the path. This is consistent with the literature (Johansson & Nugues, 2008; Roth & Lapata, 2016), where dependency paths connecting predicate and argument represent strong features for SRL. To investigate this, we plot in Figure 6.8 the percentage of paths from predicate to a predicted argument such that a subpath (i.e. at least one edge) ending in the predicted argument was retained.

For the LSTM+GNN model, we find that the reliance on paths decreases as the distance to the predicate increases, but only for nominal predicates. For the GNN-only model, we see the opposite: reliance on paths *increases* as the distance to the predicate increases. We report in Table 6.7 the proportion of edges retained on paths of varying length between the predicate and predicted roles.

[8] Note though that Marcheggiani and Titov's (2017) GNN model 'knows' which predicate it needs to focus on, as its position is marked in the BiLSTM input.

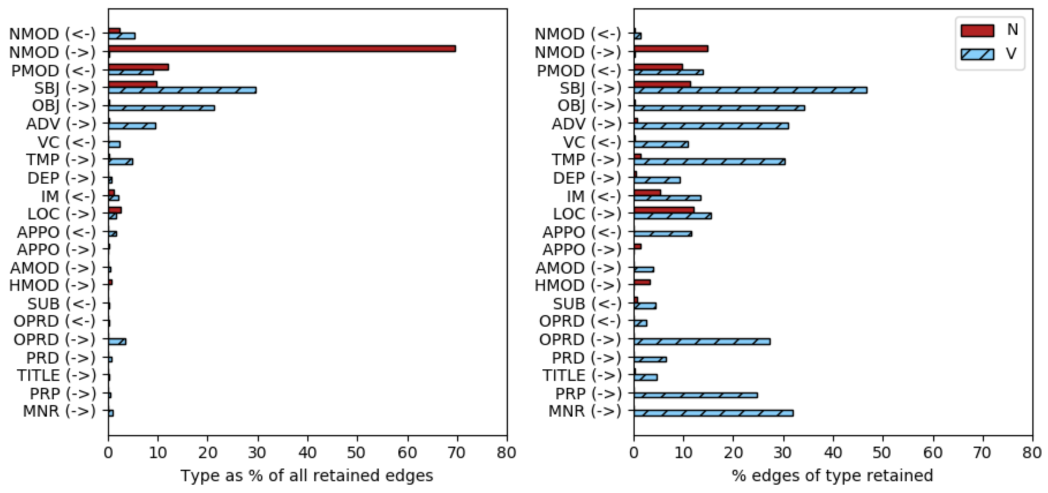


Figure 6.7: Distribution over edge types for retained edges (left) and probability of keeping each edge type (right); in both cases split by nominal (N) and verbal (V) predicates; edge types are a dependency function including computation directionality: flow from the head (–>), or flow to the head (<–). Excludes edges that occur in less than 10 % cases, and edges judged superfluous in more than 99 % cases.

Immediately noticeable is the fact that practically all direct connections between the predicate and the role are kept – this is unsurprising, as those edges constitute the most immediate indication of their syntactic relationship. Second, we note that longer paths are very often useful in both models – however, at a lower rate for nominal predicates in the LSTM+GNN model. In that particular case, the LSTM therefore seems to capture the information present in other vertices and edges on the paths. In contrast, in all other cases, the GNN complements the LSTM by connecting predicate and argument.

## 6.7 Related Work

Several recent papers have focused on developing interpretability techniques for GNNs. The closest to ours is GNNExplainer (Ying et al., 2019), wherein a soft erasure function for edges is learned individually for each example. Unlike our method (and erasure search), GNNExplainer cannot guarantee that gated edges do not affect predictions. Furthermore, as we show in our experiments (Section 6.4), separate optimisation for each example results in overfitting through hindsight bias which compromises faithfulness. Pope et al. (2019) and Xie and Lu (2019) explore gradient-based methods, including gradient heatmaps, Grad-CAM, and Excitation Backpropagation. Schwarzenberg et al. (2019), Baldassarre and Azizpour (2019), and Schnake et al. (2020) apply Layerwise Relevance Prop-

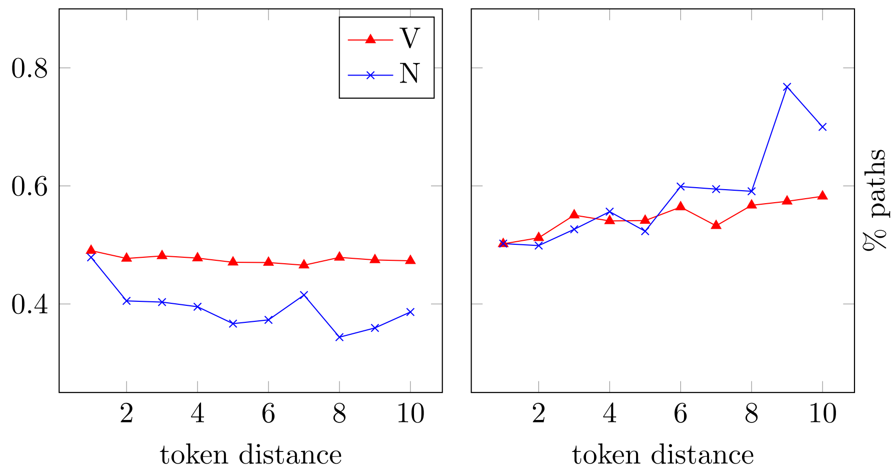


Figure 6.8: Percentage of paths used in predictions as a function of the distance between the predicate and the predicted role for the LSTM+GNN model (on the left) and the GNN only model (on the right).

agation (Bach et al., 2015) to the GNN setting, and Voita et al. (2019) employ the technique to analyse transformers.

These methods represent an alternative to GraphMask, but as we have noted their faithfulness is questionable (Nie et al., 2018; Sixt et al., 2019; Jain & Wallace, 2019), and the lack of implementation invariance (Sundararajan et al., 2017) is problematic (see Appendix A). Furthermore, significant engineering is still required to develop these techniques for certain GNNs, e.g. networks with attention as the aggregation function (Veličković et al., 2018). Another popular approach is to treat attention or gate scores as a measure of importance (Serrano & Smith, 2019). However, even leaving questionable faithfulness (Jain & Wallace, 2019) aside, many GNNs use neither gates nor attention. For those that do (Marcheggiani & Titov, 2017; Veličković et al., 2018; Neil et al., 2018; Xie & Grossman, 2018), the gates – as we demonstrate in Section 6.6 – are not necessarily informative.

Outside of graph-specific methods, one line of research involves decomposing the output into a part attributed to a specific subset of features and a part attributed to the remaining features (Shapley, 1953; Murdoch et al., 2019; Singh et al., 2019; Jin et al., 2020). For GNNs, the complexity for realistic use cases (e.g. the thousands of edges per example in De Cao et al. (2019)) is prohibitive. LIME (Ribeiro et al., 2016a) like us relies on a trained erasure model, but interprets local models instead of global models. Local models cannot trivially identify useful paths or long-distance dependent pairs of edges, and, as also pointed out in Ying et al. (2019), LIME cannot be easily applied for large general graphs. Similarly, it is unclear how to retrieve relevant paths with integrated gradients (Sundararajan et al., 2017), especially for deep GNNs and large graphs.

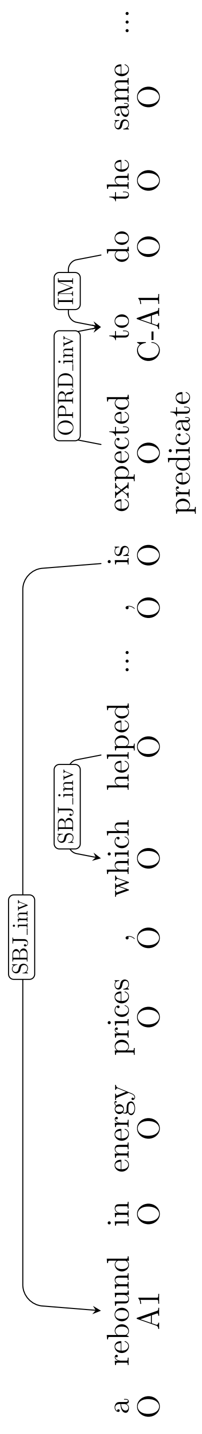


Figure 6.9: Example analysis on SRL from the GNN+LSTM model (superfluous arcs are excluded).

Masking messages in GraphMask can be equivalently thought of as adding a certain type of noise to these messages. Therefore, GraphMask can be categorised as belonging to the recently introduced class of perturbation-based methods (Guan et al., 2019; Taghanaki et al., 2019; Schulz et al., 2020) which equate feature importance with sensitivity of the prediction to the perturbations of that feature. The closest to our model is Schulz et al. (2020), wherein the authors like us apply a secondary, trained model to predict the relevancy of a feature in a given layer. Unlike us, this trained model has “lookahead”, i.e. access to layers above the studied layer, making their model vulnerable to hindsight bias. Their approach uses soft gates on individual hidden state dimensions to interpolate between hidden states and Gaussian noise, in order to detect important features for CNNs on an image processing task. They make independent Gaussian assumptions on the features to derive their objective. We adapted their method to GNNs and used it as a baseline in our experiments.

In parallel with the work which this chapter is based on (Schlichtkrull et al., 2020b), Luo et al. (2020) have developed an interpretability technique for GNNs relying on edge masking. Their approach utilises a mutual information objective like GNNExplainer (Ying et al., 2019), along with local classifiers choosing as we do for GraphMask whether to retain or discard edges on the basis of binary concrete variables. We have also introduced a similar differentiable masking approach to post-hoc analysis for transformers in De Cao et al. (2020), where we used sparse stochastic gates and $L_0$ regularisation to determine which input tokens can be dropped, conditioning on various hidden layers.

## 6.8 On Faithfulness

As we mentioned in Section 6.2, faithfulness is still not a concept with a single, well-defined meaning in the community. Jacovi and Goldberg (2020) argues that the concept, intuitively, denotes accurate representation of the reasoning process behind model predictions. Furthermore, the authors suggest that the noise inherent to machine learning methods virtually guarantees that no technique will be perfectly faithful for every example. That is, singular counterexamples to any given technique’s faithfulness can often be found via adversarial attack. As such, faithfulness should not be seen as a binary property, but rather a graded criterion by which techniques can be *more* or *less* faithful than competitors.

In this chapter, we have attempted to follow Jacovi and Goldberg’s (2020) definition. That is, we have treated faithfulness as a graded criterion measuring the degree to which techniques accurately represent the reasoning of the underlying models. We have argued for the comparatively high faithfulness of GraphMask in several ways. First, we have used a synthetic task with a single, well-defined solution that perfectly optimised models must implement. Second, we have used the degree to which real-world models’ predictions change when relying on GraphMask explanations rather than full graphs. Third, we have argued that explanations for the QA task investigated in Section 6.5 require retaining the bottom layer of the GNN, and that only GraphMask does so by a satisfying degree. Here, we shall attempt to relate our approach to the literature.

### 6.8.1 Our Investigations

Our comparison of different techniques under the controlled settings of Section 6.4 follows the trend discussed in Jacovi and Goldberg (2020) to address faithfulness through falsification by counterexample (see also Jain and Wallace (2019) and Wiegreffe and Pinter (2019)). In addition to necessitating a graded definition of faithfulness, Jacovi and Goldberg (2020) also argued that noise problematises the use of singular counterexamples. We have attempted to correct for this by testing on many generated examples rather than a single handpicked example. This allows us to reason about faithfulness on the dataset-level by measuring $F_1$-score (see Table 6.1).

We find a clear trend where GraphMask for this task produces more faithful explanations than competitors. However, with a score of 99.4%, there are a few examples for which GraphMask fails. Nevertheless, GraphMask clearly performs better than the alternatives. Similarly, integrated gradients with a score of 90.8% clearly outperforms erasure search with its score of 28.6%. However, examples could be picked where neither produces faithful explanations. This supports Jacovi and Goldberg's (2020) claims about singular counterexamples and the need to conceptualise faithfulness as graded rather than binary.

Another approach to faithfulness is to set theoretical criteria which faithful techniques must follow. Techniques can then be tested to determine whether these criteria are met. Three such criteria were given in Yu et al. (2019), while a few were discussed in Jacovi and Goldberg (2020). It is important to note that the lack of a general definition of faithfulness means that the community does not agree on any particular criterion's validity. Nevertheless, one criterion reoccurs commonly: interpretations that produce different predictions than the analysed model are not faithful (see Corollary 1.2 in Jacovi and Goldberg (2020), the Comprehensiveness Condition from Yu et al. (2019)). This is related to the line of research which measures the *fidelity* of explanations, e.g. the accuracy of predictions using interpretations rather than full inputs (Sushil et al., 2018; Lakkaraju et al., 2019).

In Sections 6.5 and 6.6, we measure the performance of the QA and SRL models with the full graphs, as well as with only the edges retained after masking with GraphMask. In both cases, dropping edges marked superfluous does not harm performance. As such, GraphMask passes this test, replicating predictions with high fidelity. We furthermore experimented with randomly dropping the *remaining* edges, in addition to those marked superfluous. For both tasks, even dropping a random 25% significantly reduces performance. This line of inquiry is related to the recently proposed strategy for measuring the faithfulness of explanations for models over text by gradually replacing the most salient words with mask tokens (DeYoung et al., 2020; Atanasova et al., 2020).

The third argument we make about faithfulness is based on observations of the average attribution assigned to entire layers in the QA task (see Figure 6.5). We note that dropping the bottom layer results in a large performance loss, but alternatives to our technique mark that layer as unimportant on average. This represents another form of dataset-level counterexample, where specific details can be inferred about the behaviour of a given model and used to reason about faithfulness. Such counterexamples can be difficult to find, and as such are not a reliable way of testing for faithfulness across different tasks – however, when they occur, they are valuable evidence.

### 6.8.2 Formal Criteria for Faithfulness

In Jacovi and Goldberg (2020), the authors called for the development of formal definitions of faithfulness. Yu et al. (2019) present an early attempt to formulate a complete, theoretical definition of the concept. They do so through the aforementioned three criteria – sufficiency, comprehensiveness, and compactness. Their definition is limited to rationale generation (e.g. techniques that produce binary attribution scores). As GraphMask falls under this category, we can compare our technique to their definition.

Their first condition, *sufficiency*, represents the simple intuition that a rationale $R$ which produces different predictions than the full set of input elements $X$ must be missing some essential information. Formally, sufficiency asserts that the probability distribution of the output space must be unchanged: $p_Y(\cdot|R) = p_Y(\cdot|X)$. This reflects the notion of fidelity discussed above, although sufficiency is much more restrictive – it is a binary rather than a graded criterion, and it measures the distribution of the output space rather than the maximum likelihood prediction. Because of this, we consider the condition too limited to serve as a test. We note, however, that the second part of our training objective (see Definition 6.3.3), the divergence $\mathrm{D}_\star[f(\mathcal{G}, \mathcal{X}) \| f(\mathcal{G}_S, \mathcal{X})]$, corresponds to optimizing for sufficiency.

Yu et al.'s (2019) second condition, *comprehensiveness*, is introduced to identify degenerate rationales. Formally, the condition states that $H(Y|R_c) \geq H(Y|R) + h$ for some constant $h$. That is, the entropy of the output space using the complement rationale (e.g. every element of the full input set $X$ *not* in $R$) must be much greater than the entropy of the output space using the rationale itself. For graph models where the presence of some input elements (e.g. the edge to the square vertex in Example 6.2) are necessary for *other* input elements to become predictive, this condition is not sufficient to avoid degeneration. In the example, trivial solutions at the second hop can encode $Y$ without violating comprehensiveness – even degenerate rationales which contain the first-hop vertex (coloured white) will be more informative than their complements, as meaningful predictions cannot be made without that vertex.

The final condition introduced by Yu et al. (2019) is *compactness*. Intuitively, faithful explanations should not contain any superfluous information. They state this condition as a requirement of sparsity, e.g. $|R| \leq s$ for some threshold $s$. Assuming that the rationale can be ordered, they furthermore include a requirement of consecutiveness as $\sum_i |R_i - Ri - 1| \leq t$ for some threshold $t$. While the latter is not relevant to graph models, we optimize for sparsity as well as sufficiency. This is done through the first part of our objective, e.g. $\sum_{k=1}^{L} \sum_{(u,v)\in\mathcal{E}} \mathbf{1}_{[\mathbb{R}\neq 0]}(z_e^{(k)})$. We furthermore measure the degree to which GraphMask produces sparse explanations for the QA and SRL tasks, finding that large parts of the input space can indeed be discarded.

## 6.9 Conclusion

We introduced GraphMask, a post-hoc interpretation method potentially applicable to any GNN model. By learning end-to-end differentiable hard gates for every message and amortizing over the training data, GraphMask is faithful to the studied model, scalable to modern GNNs, and capable of identifying both how edges and paths influence predictions. We applied our method to analyze the predictions of two NLP models from the literature – a semantic role labeling model, and a question answering model. GraphMask uncovers which edge types these models rely on, and how they employ paths when making predictions. While these findings themselves reveal interesting details of the inner workings of popular models, they also provide an illustration of types of analysis enabled by GraphMask. Here we have focused on NLP applications, where there is a strong demand for interpretability techniques applicable to graph-based models injecting linguistic and structural priors.

Returning to the research questions we set out to answer, we see that GraphMask – and, by extension, erasure search by differentiable masking – provides a tractable means of interpreting the predictions of trained GNNs. We have demonstrated the effectiveness of our technique for NLP, leaving applications to other domains for future work. To reason about the faithfulness of GraphMask, we have introduced a synthetic task with a known gold standard for faithfulness. Such tasks allow the use of both theoretical and empirical arguments to compare different interpretability techniques; in future work, batteries of such tests could be used to provide even stronger arguments.

In previous chapters, we have experimented with GNNs for various NLP tasks. Between the R-GCN of Chapter 3 and the GR-GNN of Chapter 4, we have seen very different architectures perform well for different tasks. We have previously mentioned the difficulty in understanding why specific variants perform better than others in certain situations, owing to the black-box nature of these highly complex, nonlinear models. GraphMask and other interpretability techniques represent a means for researchers to more effectively study and compare the inferences learned by different architectures.

# Chapter 7

# Conclusions

In this thesis, we set out to investigate the modelling of structured data with specialised neural network architectures for natural language processing. We conducted experiments using *graph neural networks* (GNNs) to address the need for neural models capable of incorporating graphs, focusing primarily on knowledge bases. Within that context, we experimented with relational link prediction in Chapter 3, and with factoid question answering in Chapter 4. We furthermore investigated in Chapter 5 the use of linearisation and large language models as an alternative. Finally, in Chapter 6, we developed an interpretability technique for graph neural networks. We summarise here the main findings of our investigations.

To address link prediction in knowledge bases in Chapter 3, we introduced Relational Graph Convolutional Network (R-GCN), a family of graph neural networks which incorporate relation type and direction when computing entity representations. We paired R-GCN encoders with DistMult, a decoder from the literature, constructing a novel approach to relational link prediction with strong performance. We attempted to answer the following research questions:

1. *Can graph neural network encoders improve the modelling of entities in knowledge bases for relational link prediction?*
2. *Which aspects of relational graph structure do graph neural networks capture better than baseline techniques?*

We applied our model to the challenging FB15k-237 dataset (Toutanova & Chen, 2015), demonstrating strong performance gains compared to contemporary systems. Among other baselines, we saw our system compare favourably to the pure DistMult algorithm without a GNN-encoder. As such, it is clear that the use of a GNN-encoder improved the modelling of entities. We saw these improvements primarily play a role for complex inference problems, and when very specific information was needed to make predictions — for modelling high-degree vertices, and for predicting links with infrequent relation types.

Explicitly encoding vertices with our R-GCN did present a drawback for vertices separated by many edges in the graph, as a GNN with $k$ layers can only model the $k$-neighbourhood of each vertex. For vertices separated by $k$ or fewer edges, the GNN resulted in large performance gains; for vertices further apart, the factorisation baseline performed better. As such, GNNs excel at capturing complex relationships between vertices within limited neighbourhoods defined by the number of layers in the model.

Our experiments with relational link prediction gave a clear indication of the value of explicitly encoding vertices with GNNs in semantic graphs. Furthermore, the use of models combining contextualised GNN-based vertex embeddings with a decoder represents a promising development for modelling relational link prediction. We picked up the former thread in Chapter 4, where we applied GNNs to knowledge base question answering; the latter has been picked up by several recent papers demonstrating new, promising results with encoder-decoder models for relational link prediction (Bansal et al., 2019; Wang et al., 2020b).

We experimented in Chapter 4 with various architectures and modelling choices for factoid question answering, based on the idea of modelling local neighbourhoods around mentioned entities with GNNs. We introduced systems modelling the problem as picking the right *entity* or the right *relation*, as well as a novel GNN-architecture – GR-GNN – using gates and an $L_0$ regularisation term to focus only on relevant edges. We sought to answer the following research questions:

1. *Can graph neural network encoders yield benefits for question answering by modelling entities and relations in the knowledge graph?*

2. *Is entity-based or relation-based modelling the more suitable strategy, and which graph neural network models work best in either setting?*

We tested our models on the WebQuestions dataset (Berant et al., 2013). Our models achieved performance comparable to (although not quite beating) contemporary state-of-the-art methods. Ablation tests demonstrated clear benefits from the use of GNNs and the use of multiple GNN-layers. Our experiments indicate that GNN-based encoders represent a promising alternative to semantic parsers or other similar approaches.

Relation-based modelling resulted in stronger performance than entity-based modelling, as well as faster training. We theorised that this improvement stems in part from the implicit modelling of a singular primary relation answering each question, corresponding to the "central paths" employed by Yih et al. (2015). Interestingly, very different GNN architectures performed well for relation-based and entity-based modelling – for the former, we achieved the best performance with the R-GCN of Chapter 3; for the latter, with our novel GR-GNN.

We also included an overview of several developments in factoid QA concurrent or subsequent to our own research. Notably, the line of work including GRAFT-Net (Sun et al., 2018) and PullNet (Sun et al., 2019b), which also employ graph neural network encoders. Their experiments indicate that much stronger benefits can be gained from GNN-based models if the underlying graphs are filtered to include only the most relevant edges – this matches our experience with the necessity of our sparsity-inducing $L_0$ term.

In the process of developing GNNs for factoid QA, we struggled to understand why certain architectures outperformed others, and in what the strengths and weaknesses of different models lay. This, in part, inspired our later development in Chapter 6 of a post-hoc interpretability tool for GNNs, in order to circumvent the black-box nature of these highly complex, nonlinear models.

In Chapter 5, we experimented with an alternative to GNNs for modelling structured data – linearisation and encoding with a pretrained language model. We focused on a task with little prior research, open-domain fact verification over collections of tables, and introduced a novel attention-based architecture addressing the problem. Inspired by recent work for unstructured data, we proposed a two-step model that combines non-parametric, heuristic retrieval with a neural reader fusing evidence from several retrieved tables. We attempted to answer the following research questions:

1. *Can neural models be applied to efficiently encode tables for querying in open-domain fact verification tasks?*

2. *How can the existence of large-scale closed-domain datasets be exploited to improve open-domain performance?*

3. *Will our models still yield performance improvements if applied to query against a Wikipedia-scale dataset rather than the limited knowledge sources of existing datasets?*

We conducted experiments with our system using the recently introduced TabFact dataset (Chen et al., 2020), with our open-domain model achieving performance on pair with the current state of the art for the closed domain. When applied to the closed domain, our model furthermore represents a new state of the art. As such, it is clear that linearisation and language modeling represents a highly performant approach to encoding tables for NLP tasks. This also matches the recent finding by Gupta et al. (2020), wherein linearisation and a RoBERTa-based encoder also yielded strong performance for table encoding.

We introduced two strategies for exploiting the existence of large closed-domain datasets to improve performance in the open domain, both reliant on modelling a choice of relevant tables along with the truth values of claims. In our experiments, we demonstrated that introducing a reranking objective and jointly modelling that alongside truth value significantly improves performance. We furthermore demonstrated how these strategies provide a means of determining, at test time, whether sufficient relevant information has been retrieved for any judgement of truth value to be trustworthy.

In the challenging setting where retrieval is performed against the entirety of Wikipedia rather than just the TabFact tables, our model still performed well. By combining our joint reranking- and verification-objective with a model making decisions based on multiple retrieved tables, we demonstrated significant gains. We note that these results are preliminary, owing to the novelty of the task; nevertheless, our approach improved greatly upon simple baselines such as modelling only a single retrieved table.

Through the use of large pretrained language models, linearisation remains an effective alternative to semantic parsing strategies, as well as to the GNN-based models we have explored in previous chapters. This is not a phenomenon unique to table modelling. Indeed, for factoid question answering, recent models employing similar approaches (Oğuz et al., 2020) compare favourably both to semantic parsing (Yih et al., 2015) and to the GNN-based strategies we have mentioned (Sun et al., 2018; Sun et al., 2019b).

The graph neural network models which have occupied much of our attention throughout this thesis are complex, and it is often difficult to understand the inferences that lead to specific predictions being made. In Chapter 6, we introduced GraphMask, a post-hoc interpretability technique for GNNs designed to alleviate this problem. We attempted to answer the following research questions:

1. *How can we tractably provide rationales for the predictions of a given graph neural network?*

2. *How can we ensure that our technique is faithful, and measure the relative faithfulness of different techniques?*

GraphMask produces rationales consisting of a binary choice between usefulness and superfluity for every message sent by the GNN. This allows researchers to construct arguments about which a particular GNN uses edges, vertices, and paths. The choices are predicted through local models learning end-to-end differentiable hard gates for every message, amortised over the training data; this process ensures tractability and faithfulness. We demonstrated the effectiveness of this approach by analysing the predictions of two NLP models from the literature – a semantic role labelling model, and a question answering model.

We introduced a synthetic task with a perfectly optimised model to reason about different interpretability techniques' faithfulness. Using a simple task with a predefined solution, we constructed a gold standard for faithfulness. This allowed us to present both theoretical and empirical arguments comparing GraphMask to several alternatives; for this simple task, GraphMask was the only approach to demonstrate near-perfect faithfulness. The use of small, focused synthetic tasks represents a promising direction through which the relative faithfulness of models can be compared.

In the last several years, the approach to structural modelling within NLP has changed drastically. Graph neural networks have grown from an infant state with few, specialised applications to become a regularly used architecture with many variants. During that same period, large pretrained language models based on the attention mechanism have come to dominate the field. While it is undeniable that much work is necessary to understand either in fullness – their strengths and weaknesses, their promises and prospective dangers – we hope that the experiments, discussions, and techniques presented within this thesis have contributed positively towards that end. With the ubiquity of structured data in the world, the development of processes to model such data is among the most important steps that must be taken to bring NLP beyond today's capabilities. We believe the models and techniques explored here lay the groundwork for further research into this topic.

# Appendix A

# Implementation invariance for GNNs

In Section 6.7, we mentioned Layerwise Relevance Propagation (LRP) as an alternative to our strategy for interpreting GNNs. We mentioned that – as also discussed in Sundararajan et al. (2017) – LRP can be problematic as it is not implementation invariant. Here, we include an example showing attribution scores with LRP for two functionally equivalent GNN models. The differences in learned weights result in different predictions of edge importance, despite identical inputs and predictions.

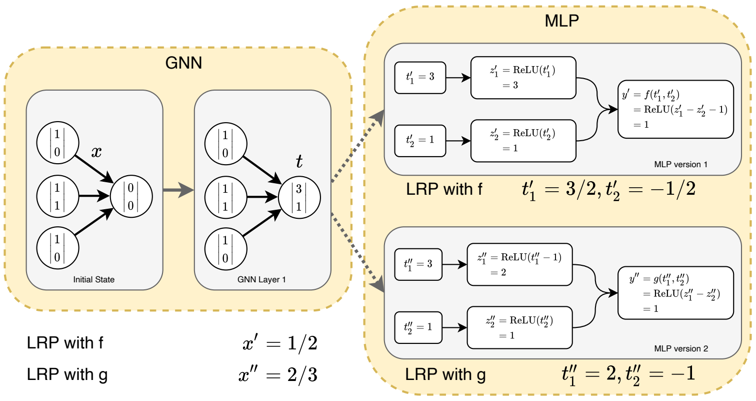
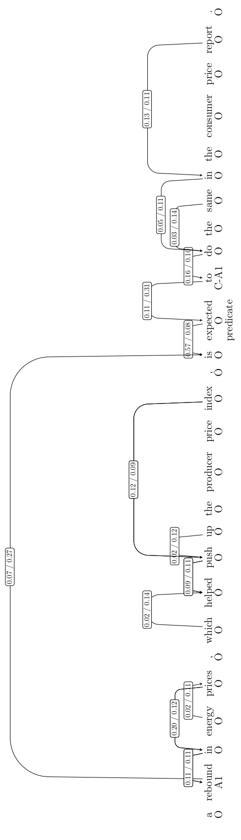

GNN
Initial State
GNN Layer 1
$x$
$t$
MLP
$t'_1 = 3$
$z'_1 = \mathrm{ReLU}(t'_1) = 3$
$t'_2 = 1$
$z'_2 = \mathrm{ReLU}(t'_2) = 1$
$y' = f(t'_1, t'_2) = \mathrm{ReLU}(z'_1 - z'_2 - 1) = 1$
MLP version 1
LRP with f $t'_1 = 3/2, t'_2 = -1/2$
$t''_1 = 3$
$z''_1 = \mathrm{ReLU}(t''_1 - 1) = 2$
$t''_2 = 1$
$z''_2 = \mathrm{ReLU}(t''_2) = 1$
$y'' = g(t''_1, t''_2) = \mathrm{ReLU}(z''_1 - z''_2) = 1$
MLP version 2
LRP with g $t''_1 = 2, t''_2 = -1$
LRP with f $x' = 1/2$
LRP with g $x'' = 2/3$

Figure A.1: Attributions for two functionally equivalent networks. We give a graph as an input to a GNN (on the left) where $x$ is the edge from the top-left node to the central right node. The GNN update rule is simply aggregation with a sum over the neighbour nodes and no activation function. After one GNN layer we apply a MLP (on the right) which is implemented with two functionally equivalent networks $f(t_1, t_2)$ and $g(t_1, t_2)$ (exactly the same as in the counterexample provided in Figure 7 in Sundararajan et al. (2017)). Since LRP is not implementation invariant, it will produce two different attributions for the node $t$ (i.e., $t'$ and $t''$), and as a consequence of the propagation rule (Schwarzenberg et al., 2019), the attribution to $x$ will also be affected (i.e., $x'$ and $x''$).

# Appendix B

# SRL Example with Soft Gates



a rebound in energy prices , which helped push up the producer price index , is expected to do the same in the consumer price report .
O A1 O O O O O O O O O O O O O O predicate C-A1 O O O O O O O O O

Figure B.1: The example analysis from Figure 6.9, using the analysis heuristic where edges with soft gate values more than one standard deviation below the mean are discarded. Directions are combined into one arc.

# Bibliography

Alhindi, Tariq, Petridis, Savvas, & Muresan, Smaranda. (2018). Where is your evidence: Improving fact-checking by justification modeling. *Proceedings of the First Workshop on Fact Extraction and VERification (FEVER)*, 85–90.

Atanasova, Pepa, Simonsen, Jakob Grue, Lioma, Christina, & Augenstein, Isabelle. (2020). A diagnostic study of explainability techniques for text classification. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 3256–3274. https://doi.org/10.18653/v1/2020.emnlp-main.263

Athey, Susan, & Imbens, Guido. (2015). Machine learning methods for estimating heterogeneous causal effects. *Research Papers 3350, Stanford University, Graduate School of Business.*

Auer, Sören, Bizer, Christian, Kobilarov, Georgi, Lehmann, Jens, Cyganiak, Richard, & Ives, Zachary. (2007). Dbpedia: A nucleus for a web of open data. *The semantic web* (pp. 722–735). Springer.

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

Bach, Sebastian, Binder, Alexander, Montavon, Grégoire, Klauschen, Frederick, Müller, Klaus-Robert, & Samek, Wojciech. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, *10*(7), e0130140.

Baehrens, David, Schroeter, Timon, Harmeling, Stefan, Kawanabe, Motoaki, Hansen, Katja, & Müller, Klaus-Robert. (2010). How to explain individual classification decisions. *Journal of Machine Learning Research*, *11*(Jun), 1803–1831.

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

Balazevic, Ivana, Allen, Carl, & Hospedales, Timothy. (2019). Tucker: Tensor factorization for knowledge graph completion. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 5188–5197.

Baldassarre, Federico, & Azizpour, Hossein. (2019). Explainability techniques for graph convolutional networks. *arXiv preprint arXiv:1905.13686*.

Bansal, Trapit, Juan, Da-Cheng, Ravi, Sujith, & McCallum, Andrew. (2019). A2N: Attending to neighbors for knowledge graph inference. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 4387–4392. https://doi.org/10.18653/v1/P19-1431

Bao, Junwei, Duan, Nan, Zhou, Ming, & Zhao, Tiejun. (2014). Knowledge-based question answering as machine translation. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 967–976.

Bast, Hannah, & Haussmann, Elmar. (2015). More accurate question answering on freebase. *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 1431–1440.

Bastings, Jasmijn, Titov, Ivan, Aziz, Wilker, Marcheggiani, Diego, & Sima'an, Khalil. (2017). Graph convolutional encoders for syntax-aware neural machine translation. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 1957–1967. https://doi.org/10.18653/v1/D17-1209

Bastings, Jasmijn, Aziz, Wilker, & Titov, Ivan. (2019). Interpretable neural predictions with differentiable binary variables. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2963–2977. https://doi.org/10.18653/v1/P19-1284

Battaglia, Peter, Pascanu, Razvan, Lai, Matthew, Rezende, Danilo Jimenez, et al. (2016). Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 4502–4510.

Beck, Daniel, Haffari, Gholamreza, & Cohn, Trevor. (2018). Graph-to-sequence learning using gated graph neural networks. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 273–283. https://doi.org/10.18653/v1/P18-1026

Belinkov, Yonatan, & Glass, James. (2019). Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics*, *7*, 49–72.

Berant, Jonathan, Chou, Andrew, Frostig, Roy, & Liang, Percy. (2013). Semantic parsing on Freebase from question-answer pairs. *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 1533–1544.

Berant, Jonathan, & Liang, Percy. (2014). Semantic parsing via paraphrasing. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, *1*, 1415–1425.

Berant, Jonathan, & Liang, Percy. (2015). Imitation learning of agenda-based semantic parsers. *Transactions of the Association for Computational Linguistics*, *3*, 545–558.

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

Bhagavatula, Chandra Sekhar, Noraset, Thanapon, & Downey, Doug. (2013). Methods for exploring and mining tables on wikipedia. *Proceedings of the ACM SIGKDD Workshop on Interactive Data Exploration and Analytics*, 18–26.

Bogin, Ben, Berant, Jonathan, & Gardner, Matt. (2019). Representing schema structure with graph neural networks for text-to-SQL parsing. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 4560–4565. https://doi.org/10.18653/v1/P19-1448

Bollacker, Kurt, Evans, Colin, Paritosh, Praveen, Sturge, Tim, & Taylor, Jamie. (2008). Freebase: A collaboratively created graph database for structuring human knowledge. *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, 1247–1250. https://doi.org/10.1145/1376616.1376746

Bordes, Antoine, Usunier, Nicolas, Garcia-Duran, Alberto, Weston, Jason, & Yakhnenko, Oksana. (2013). Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 2787–2795.

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

Bowman, Samuel R., Angeli, Gabor, Potts, Christopher, & Manning, Christopher D. (2015). A large annotated corpus for learning natural language inference. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 632–642. https://doi.org/10.18653/v1/D15-1075

Brill, Eric, Dumais, Susan, & Banko, Michele. (2002). An analysis of the askmsr question-answering system. *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, 257–264.

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

Cafarella, Michael J, Halevy, Alon, Wang, Daisy Zhe, Wu, Eugene, & Zhang, Yang. (2008). Webtables: Exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, *1*(1), 538–549.

Cafarella, Michael J, Halevy, Alon, & Khoussainova, Nodira. (2009). Data integration for the relational web. *Proceedings of the VLDB Endowment*, *2*(1), 1090–1101.

Chakrabarti, Kaushik, Chen, Zhimin, Shakeri, Siamak, & Cao, Guihong. (2020a). Open domain question answering using web tables. *arXiv preprint arXiv:2001.03272*.

Chakrabarti, Kaushik, Chen, Zhimin, Shakeri, Siamak, Cao, Guihong, & Chaudhuri, Surajit. (2020b). TableQnA: Answering list intent queries with web tables. *arXiv preprint arXiv:2001.04828*.

Chang, Kai-Wei, Yih, Wen-tau, Yang, Bishan, & Meek, Christopher. (2014). Typed tensor decomposition of knowledge bases for relation extraction. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1568–1579. https://doi.org/10.3115/v1/D14-1165

Chelba, Ciprian, Mikolov, Tomas, Schuster, Mike, Ge, Qi, Brants, Thorsten, Koehn, Phillipp, & Robinson, Tony. (2013). One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*.

Chen, Danqi, Fisch, Adam, Weston, Jason, & Bordes, Antoine. (2017). Reading Wikipedia to answer open-domain questions. *Association for Computational Linguistics (ACL)*.

Chen, Jiaoyan, Jiménez-Ruiz, Ernesto, Horrocks, Ian, & Sutton, Charles. (2019). Colnet: Embedding the semantics of web tables for column type prediction. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*, 29–36.

Chen, Wenhu, Wang, Hongmin, Chen, Jianshu, Zhang, Yunkai, Wang, Hong, Li, Shiyang, Zhou, Xiyou, & Yang Wang, William. (2020). Tabfact : A large-scale dataset for table-based fact verification. *International Conference on Learning Representations (ICLR)*.

Chen, Wenhu, Chang, Ming-Wei, Schlinger, Eva, Wang, William Yang, & Cohen, William W. (2021). Open question answering over tables and text. *International Conference on Learning Representations*.

Cohen, Sarah, Li, Chengkai, Yang, Jun, & Yu, Cong. (2011). Computational journalism: A call to arms to database researchers. *CIDR 2011 - 5th Biennial Conference on Innovative Data Systems Research, Conference Proceedings*, 148–151.

Dagan, Ido, Glickman, Oren, & Magnini, Bernardo. (2005). The PASCAL recognising textual entailment challenge. *Machine Learning Challenges Workshop*, 177–190.

Dalton, Jeffrey, Dietz, Laura, & Allan, James. (2014). Entity query feature expansion using knowledge base links. *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, 365–374.

De Cao, Nicola, Aziz, Wilker, & Titov, Ivan. (2019). Question answering by reasoning across documents with graph convolutional networks. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2306–2317. https://doi.org/10.18653/v1/N19-1240

De Cao, Nicola, Schlichtkrull, Michael, Aziz, Wilker, & Titov, Ivan. (2020). How do decisions emerge across layers in neural models? interpretation with differentiable masking. *arXiv preprint arXiv:2004.14992*.

Defferrard, Michaël, Bresson, Xavier, & Vandergheynst, Pierre. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems (NIPS)*.

Dettmers, T, Minervini, P, Stenetorp, P, & Riedel, S. (2018). Convolutional 2d knowledge graph embeddings. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, *32*, 1811–1818.

Devlin, Jacob, Chang, Ming-Wei, Lee, Kenton, & Toutanova, Kristina. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171–4186. https://doi.org/10.18653/v1/N19-1423

DeYoung, Jay, Jain, Sarthak, Rajani, Nazneen Fatema, Lehman, Eric, Xiong, Caiming, Socher, Richard, & Wallace, Byron C. (2020). ERASER: A benchmark to evaluate rationalized NLP models. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 4443–4458. https://doi.org/10.18653/v1/2020.acl-main.408

Dong, Li, Wei, Furu, Zhou, Ming, & Xu, Ke. (2015). Question answering over freebase with multi-column convolutional neural networks. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, *1*, 260–269.

Duvenaud, David K., Maclaurin, Dougal, Iparraguirre, Jorge, Bombarell, Rafael, Hirzel, Timothy, Aspuru-Guzik, Alán, & Adams, Ryan P. (2015). Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems (NIPS)*, 2224–2232.

Eisenschlos, Julian, Krichene, Syrine, & Müller, Thomas. (2020). Understanding tables with intermediate pre-training. *Findings of the Association for Computational Linguistics: EMNLP 2020*, 281–296. https://doi.org/10.18653/v1/2020.findings-emnlp.27

Feng, Shi, Wallace, Eric, Grissom II, Alvin, Iyyer, Mohit, Rodriguez, Pedro, & Boyd-Graber, Jordan. (2018). Pathologies of neural models make interpretations difficult. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 3719–3728. https://doi.org/10.18653/v1/D18-1407

Fernandes, Patrick, Allamanis, Miltiadis, & Brockschmidt, Marc. (2019). Structured neural summarization. *International Conference on Learning Representations (ICLR)*.

Ferrucci, David, Brown, Eric, Chu-Carroll, Jennifer, Fan, James, Gondek, David, Kalyanpur, Aditya A, Lally, Adam, Murdock, J William, Nyberg, Eric, Prager, John, et al. (2010). Building watson: An overview of the deepqa project. *AI magazine*, *31*(3), 59–79.

Fey, Matthias, & Lenssen, Jan E. (2019). Fast graph representation learning with PyTorch Geometric. *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Flew, Terry, Spurgeon, Christina, Daniel, Anna, & Swift, Adam. (2012). The promise of computational journalism. *Journalism Practice*, *6*(2), 157–171.

Frasconi, Paolo, Gori, Marco, & Sperduti, Alessandro. (1998). A general framework for adaptive processing of data structures. *IEEE transactions on Neural Networks*, *9*(5), 768–786.

Garcia-Duran, Alberto, Bordes, Antoine, & Usunier, Nicolas. (2015). *Composing relationships with translations* (Doctoral dissertation). CNRS, Heudiasyc.

Ghasemi-Gol, Majid, & Szekely, Pedro. (2018). TabVec: Table vectors for classification of web tables. *arXiv preprint arXiv:1802.06290*.

Gilmer, Justin, Schoenholz, Samuel S, Riley, Patrick F, Vinyals, Oriol, & Dahl, George E. (2017). Neural message passing for quantum chemistry. *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 1263–1272.

Giulianelli, Mario, Harding, Jack, Mohnert, Florian, Hupkes, Dieuwke, & Zuidema, Willem. (2018). Under the hood: Using diagnostic classifiers to investigate and improve how language models track agreement information. *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, 240–248. https://doi.org/10.18653/v1/W18-5426

Goldberg, Yoav. (2019). Assessing BERT's syntactic abilities. *arXiv preprint arXiv:1901.05287*.

Goller, Christoph, & Kuchler, Andreas. (1996). Learning task-dependent distributed representations by backpropagation through structure. *Proceedings of International Conference on Neural Networks (ICNN'96)*, *1*, 347–352.

Gori, Marco, Monfardini, Gabriele, & Scarselli, Franco. (2005). A new model for learning in graph domains. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks.*, *2*, 729–734.

Guan, Chaoyu, Wang, Xiting, Zhang, Quanshi, Chen, Runjin, He, Di, & Xie, Xing. (2019). Towards a deep and unified understanding of deep neural models in nlp. *International Conference on Machine Learning*, 2454–2463.

Guo, Zhijiang, Zhang, Yan, & Lu, Wei. (2019). Attention guided graph convolutional networks for relation extraction. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 241–251. https://doi.org/10.18653/v1/P19-1024

Gupta, Vivek, Mehta, Maitrey, Nokhiz, Pegah, & Srikumar, Vivek. (2020). INFOTABS: Inference on tables as semi-structured data. *Proceedings of the*

*58th Annual Meeting of the Association for Computational Linguistics*, 2309–2324. https://doi.org/10.18653/v1/2020.acl-main.210

Gururangan, Suchin, Swayamdipta, Swabha, Levy, Omer, Schwartz, Roy, Bowman, Samuel, & Smith, Noah A. (2018). Annotation artifacts in natural language inference data. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 107–112. https://doi.org/10.18653/v1/N18-2017

Guu, Kelvin, Miller, John, & Liang, Percy. (2015). Traversing knowledge graphs in vector space. *arXiv preprint arXiv:1506.01094*.

Hamilton, Will, Ying, Zhitao, & Leskovec, Jure. (2017). Inductive representation learning on large graphs. *Advances in neural information processing systems*, 1024–1034.

Haveliwala, Taher H. (2003). Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE transactions on knowledge and data engineering*, *15*(4), 784–796.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, & Sun, Jian. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

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

Herzig, Jonathan, Nowak, Pawel Krzysztof, Müller, Thomas, Piccinno, Francesco, & Eisenschlos, Julian. (2020). TaPas: Weakly supervised table parsing via pre-training. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 4320–4333. https://doi.org/10.18653/v1/2020.acl-main.398

Hitchcock, Frank L. (1927). The expression of a tensor or a polyadic as a sum of products. *Studies in Applied Mathematics*, *6*(1-4), 164–189.

Hixon, Ben, Clark, Peter, & Hajishirzi, Hannaneh. (2015). Learning knowledge graphs for question answering through conversational dialog. *Proceedings of NAACL HLT*, 851–861.

Hochreiter, Sepp, & Schmidhuber, Jürgen. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Holstein, Kenneth, Wortman Vaughan, Jennifer, Daumé III, Hal, Dudik, Miro, & Wallach, Hanna. (2019). Improving fairness in machine learning systems: What do industry practitioners need? *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–16.

Honnibal, Matthew, Montani, Ines, Van Landeghem, Sofie, & Boyd, Adriane. (2020). *spaCy: Industrial-strength Natural Language Processing in Python*. Zenodo. https://doi.org/10.5281/zenodo.1212303

Izacard, Gautier, & Grave, Edouard. (2020). Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282*.

Jacovi, Alon, & Goldberg, Yoav. (2020). Towards Faithfully Interpretable NLP Systems: How should we define and evaluate faithfulness? *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics.*

Jain, Sarthak, & Wallace, Byron C. (2019). Attention is not Explanation. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 3543–3556. https://doi.org/10.18653/v1/N19-1357

Jang, Eric, Gu, Shixiang, & Poole, Ben. (2017). Categorical reparameterization with Gumbel-Softmax. *International Conference on Learning Representations.*

Jauhar, Sujay Kumar, Turney, Peter, & Hovy, Eduard. (2016). Tables as semi-structured knowledge for question answering. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 474–483.

Ji, Tao, Wu, Yuanbin, & Lan, Man. (2019). Graph-based dependency parsing with graph neural networks. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2475–2485.

Jin, Xisen, Du, Junyi, Wei, Zhongyu, Xue, Xiangyang, & Ren, Xiang. (2020). Towards Hierarchical Importance Attribution: Explaining Compositional Semantics for Neural Sequence Models. *International Conference on Learning Representations.*

Johansson, Richard, & Nugues, Pierre. (2008). Dependency-based semantic role labeling of propbank. *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, 69–78.

Johnson, Jeff, Douze, Matthijs, & Jégou, Hervé. (2019). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data.*

Joshi, Mandar, Choi, Eunsol, Weld, Daniel, & Zettlemoyer, Luke. (2017). TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1601–1611. https://doi.org/10.18653/v1/P17-1147

Jumelet, Jaap, & Hupkes, Dieuwke. (2018). Do language models understand anything? on the ability of LSTMs to understand negative polarity items. *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, 222–231. https://doi.org/10.18653/v1/W18-5424

Kampffmeyer, Michael, Chen, Yinbo, Liang, Xiaodan, Wang, Hao, Zhang, Yujia, & Xing, Eric P. (2019). Rethinking knowledge graph propagation for zero-shot learning. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11487–11496.

Karpukhin, Vladimir, Oğuz, Barlas, Min, Sewon, Wu, Ledell, Edunov, Sergey, Chen, Danqi, & Yih, Wen-tau. (2020). Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906.*

Kaushik, Divyansh, & Lipton, Zachary C. (2018). How much reading does reading comprehension require? a critical investigation of popular benchmarks. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 5010–5015. https://doi.org/10.18653/v1/D18-1546

Khashabi, Daniel, Khot, Tushar, Sabharwal, Ashish, Clark, Peter, Etzioni, Oren, & Roth, Dan. (2016). Question answering via integer programming over semi-structured knowledge. *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 1145–1152.

Kim, Been. (2015). *Interactive and interpretable machine learning models for human machine collaboration* (Doctoral dissertation). Massachusetts Institute of Technology.

Kingma, Diederik P, & Welling, Max. (2014). Auto-encoding variational bayes. *Proceedings of the 2nd International Conference on Learning Representations (ICLR).*

Kingma, Diederik P, & Ba, Jimmy. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations.*

Kiperwasser, Eliyahu, & Goldberg, Yoav. (2016). Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, *4*, 313–327. https://doi.org/10.1162/tacl_a_00101

Kipf, Thomas N., & Welling, Max. (2016). Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308.*

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

Kolda, Tamara G, & Bader, Brett W. (2009). Tensor decompositions and applications. *SIAM review*, *51*(3), 455–500.

Koncel-Kedziorski, Rik, Bekal, Dhanush, Luan, Yi, Lapata, Mirella, & Hajishirzi, Hannaneh. (2019). Text Generation from Knowledge Graphs with Graph Transformers. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2284–2293. https://doi.org/10.18653/v1/N19-1238

Kotov, Alexander, & Zhai, ChengXiang. (2012). Tapping into knowledge base for concept feedback: Leveraging conceptnet to improve search results for difficult queries. *Proceedings of the fifth ACM international conference on Web search and data mining*, 403–412.

Kurakin, Alexey, Goodfellow, Ian, & Bengio, Samy. (2016). Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236.*

Kwiatkowksi, Tom, Zettlemoyer, Luke, Goldwater, Sharon, & Steedman, Mark. (2010). Inducing probabilistic ccg grammars from logical form with higher-order unification. *Proceedings of the 2010 conference on empirical methods in natural language processing*, 1223–1233.

Kwiatkowski, Tom, Choi, Eunsol, Artzi, Yoav, & Zettlemoyer, Luke. (2013). Scaling semantic parsers with on-the-fly ontology matching. *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 1545–1556.

Lakkaraju, Himabindu, Kamar, Ece, Caruana, Rich, & Leskovec, Jure. (2019). Faithful and customizable explanations of black box models. *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, 131–138.

Lewis, Patrick, Perez, Ethan, Piktus, Aleksandara, Petroni, Fabio, Karpukhin, Vladimir, Goyal, Naman, Küttler, Heinrich, Lewis, Mike, Yih, Wen-tau, Rocktäschel, Tim, et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *arXiv preprint arXiv:2005.11401.*

Li, Jiwei, Monroe, Will, & Jurafsky, Dan. (2016a). Understanding neural networks through representation erasure. *arXiv preprint arXiv:1612.08220.*

Li, Yujia, Tarlow, Daniel, Brockschmidt, Marc, & Zemel, Richard. (2016b). Gated graph sequence neural networks. *International Conference on Learning Representations (ICLR).*

Liang, Chen, Berant, Jonathan, Le, Quoc, Forbus, Kenneth D., & Lao, Ni. (2017). Neural symbolic machines: Learning semantic parsers on Freebase with weak supervision. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 23–33. https://doi.org/10.18653/v1/P17-1003

Lin, Yankai, Liu, Zhiyuan, Luan, Huanbo, Sun, Maosong, Rao, Siwei, & Liu, Song. (2015). Modeling relation paths for representation learning of knowledge bases. *arXiv preprint arXiv:1506.00379.*

Lipton, Zachary C. (2016). The mythos of model interpretability. *ICML 2016 Workshop on Human Interpretability in Machine Learning.*

Liu, Yinhan, Ott, Myle, Goyal, Naman, Du, Jingfei, Joshi, Mandar, Chen, Danqi, Levy, Omer, Lewis, Mike, Zettlemoyer, Luke, & Stoyanov, Veselin. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692.*

Lou, Yin, Caruana, Rich, Gehrke, Johannes, & Hooker, Giles. (2013). Accurate intelligible models with pairwise interactions. *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 623–631. https://doi.org/10.1145/2487575.2487579

Louizos, Christos, Welling, Max, & Kingma, Diederik P. (2018). Learning sparse neural networks through $L_0$ regularization. *International Conference on Learning Representations.*

Luo, Dongsheng, Cheng, Wei, Xu, Dongkuan, Yu, Wenchao, Zong, Bo, Chen, Haifeng, & Zhang, Xiang. (2020). Parameterized explainer for graph neural network. *Advances in Neural Information Processing Systems*, *33*.

Luong, Thang, Pham, Hieu, & Manning, Christopher D. (2015). Effective approaches to attention-based neural machine translation. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1412–1421.

Maddison, Chris J, Mnih, Andriy, & Teh, Yee Whye. (2017). The concrete distribution: A continuous relaxation of discrete random variables. *International Conference on Learning Representations (ICLR)*.

Marcheggiani, Diego, & Titov, Ivan. (2017). Encoding sentences with graph convolutional networks for semantic role labeling. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 1507–1516.

Mishra, Pushkar, Del Tredici, Marco, Yannakoudakis, Helen, & Shutova, Ekaterina. (2019). Abusive language detection with graph convolutional networks. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2145–2150.

Murdoch, W. James, Liu, Peter J, & Yu, Bin. (2019). Beyond word importance: Contextual decomposition to extract interactions from lstms. *International Conference on Learning Representations (ICLR)*.

Neelakantan, Arvind, Roth, Benjamin, & McCallum, Andrew. (2015). Compositional vector space models for knowledge base completion. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 156–166. https://doi.org/10.3115/v1/P15-1016

Neil, Daniel, Briody, Joss, Lacoste, Alix, Sim, Aaron, Creed, Paidi, & Saffari, Amir. (2018). Interpretable graph convolutional neural networks for inference on noisy knowledge graphs. *Machine Learning for Health (ML4H) Workshop at NeurIPS*.

Nickel, Maximilian, Tresp, Volker, & Kriegel, Hans-Peter. (2011). A three-way model for collective learning on multi-relational data. *Proceedings of the 28th international conference on machine learning (ICML-11)*, 809–816.

Nickel, Maximilian, Rosasco, Lorenzo, & Poggio, Tomaso. (2016). Holographic embeddings of knowledge graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, *30*(1).

Nie, Weili, Zhang, Yang, & Patel, Ankit. (2018). A theoretical explanation for perplexing behaviors of backpropagation-based visualizations. *International Conference on Machine Learning*, 3809–3818.

Nie, Yixin, Chen, Haonan, & Bansal, Mohit. (2019). Combining fact extraction and verification with neural semantic matching networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*, 6859–6866.

Niepert, Mathias, Ahmed, Mohamed, & Kutzkov, Konstantin. (2016). Learning convolutional neural networks for graphs. *Proceedings of the 33rd annual international conference on machine learning. ACM.*

Oğuz, Barlas, Chen, Xilun, Karpukhin, Vladimir, Peshterliev, Stan, Okhonko, Dmytro, Schlichtkrull, Michael, Gupta, Sonal, Mehdad, Yashar, & Yih, Scott. (2020). Unified open-domain question answering with structured and unstructured knowledge.

O'Neil, Cathy. (2016). *Weapons of math destruction: How big data increases inequality and threatens democracy.* Crown Publishing Group.

Parmar, Niki, Vaswani, Ashish, Uszkoreit, Jakob, Kaiser, Lukasz, Shazeer, Noam, Ku, Alexander, & Tran, Dustin. (2018). Image transformer. *International Conference on Machine Learning*, 4055–4064.

Pasupat, Panupong, & Liang, Percy. (2015). Compositional semantic parsing on semi-structured tables. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 1470–1480. https://doi.org/10.3115/v1/P15-1142

Pennington, Jeffrey, Socher, Richard, & Manning, Christopher D. (2014). Glove: Global vectors for word representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532–1543.

Perozzi, Bryan, Al-Rfou, Rami, & Skiena, Steven. (2014). Deepwalk: Online learning of social representations. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 701–710. https://doi.org/10.1145/2623330.2623732

Peters, Matthew, Neumann, Mark, Iyyer, Mohit, Gardner, Matt, Clark, Christopher, Lee, Kenton, & Zettlemoyer, Luke. (2018). Deep contextualized word representations. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2227–2237. https://doi.org/10.18653/v1/N18-1202

Pimplikar, Rakesh, & Sarawagi, Sunita. (2012). Answering table queries on the web using column keywords. *Proceedings of the VLDB Endowment*, *5*(10), 908–919.

Pope, Phillip E, Kolouri, Soheil, Rostami, Mohammad, Martin, Charles E, & Hoffmann, Heiko. (2019). Explainability methods for graph convolutional neural networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 10772–10781.

Ravula, Anirudh, Alberti, Chris, Ainslie, Joshua, Yang, Li, Pham, Philip Minh, Wang, Qifan, Ontanon, Santiago, Sanghai, Sumit Kumar, Cvicek, Vaclav,

& Fisher, Zach. (2020). Etc: Encoding long and structured inputs in transformers. *2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020)*.

Rebele, Thomas, Suchanek, Fabian, Hoffart, Johannes, Biega, Joanna, Kuzey, Erdal, & Weikum, Gerhard. (2016). Yago: A multilingual knowledge base from wikipedia, wordnet, and geonames. *International semantic web conference*, 177–185.

Reddy, Siva, Lapata, Mirella, & Steedman, Mark. (2014). Large-scale semantic parsing without question-answer pairs. *Transactions of the Association of Computational Linguistics*, *2*(1), 377–392.

Reddy, Siva, Täckström, Oscar, Collins, Michael, Kwiatkowski, Tom, Das, Dipanjan, Steedman, Mark, & Lapata, Mirella. (2016). Transforming dependency structures to logical forms for semantic parsing. *Transactions of the Association for Computational Linguistics*, *4*, 127–140.

Rezende, Danilo Jimenez, Mohamed, Shakir, & Wierstra, Daan. (2014). Stochastic backpropagation and approximate inference in deep generative models. *International Conference on Machine Learning (ICML)*.

Ribeiro, Marco Tulio, Singh, Sameer, & Guestrin, Carlos. (2016a). "why should i trust you?" explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 1135–1144.

Ribeiro, Marco Tulio, Singh, Sameer, & Guestrin, Carlos. (2016b). Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386*.

Ridgeway, Greg, Madigan, David, Richardson, Thomas, & O'Kane, John. (1998). Interpretable boosted naïve bayes classification. *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, 101–104.

Rogers, Anna, Kovaleva, Olga, & Rumshisky, Anna. (2021). A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, *8*, 842–866.

Roth, Michael, & Lapata, Mirella. (2016). Neural semantic role labeling with dependency path embeddings. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1192–1202. https://doi.org/10.18653/v1/P16-1113

Scarselli, Franco, Gori, Marco, Tsoi, Ah Chung, Hagenbuchner, Markus, & Monfardini, Gabriele. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, *20*(1), 61–80.

Schlichtkrull, Michael, Kipf, Thomas N, Bloem, Peter, Van Den Berg, Rianne, Titov, Ivan, & Welling, Max. (2018). Modeling relational data with graph convolutional networks. *European Semantic Web Conference*, 593–607.

Schlichtkrull, Michael, Karpukhin, Vladimir, Oğuz, Barlas, Lewis, Mike, Yih, Wen-tau, & Riedel, Sebastian. (2020a). Joint verification and reranking for open fact checking over tables. *arXiv preprint arXiv:2012.15115*.

Schlichtkrull, Michael Sejr, De Cao, Nicola, & Titov, Ivan. (2020b). Interpreting graph neural networks for nlp with differentiable edge masking. *arXiv preprint arXiv:2010.00577.*

Schnake, Thomas, Eberle, Oliver, Lederer, Jonas, Nakajima, Shinichi, Schütt, Kristof T, Müller, Klaus-Robert, & Montavon, Grégoire. (2020). XAI for graphs: Explaining graph neural network predictions by identifying relevant walks. *arXiv preprint arXiv:2006.03589.*

Schulz, Karl, Sixt, Leon, Tombari, Federico, & Landgraf, Tim. (2020). Restricting the flow: Information bottlenecks for attribution. *International Conference on Learning Representations.*

Schwarzenberg, Robert, Hübner, Marc, Harbecke, David, Alt, Christoph, & Hennig, Leonhard. (2019). Layerwise relevance visualization in convolutional text graph classifiers. *Proceedings of the Thirteenth Workshop on Graph-Based Methods for Natural Language Processing (TextGraphs-13)*, 58–62. https://doi.org/10.18653/v1/D19-5308

Serrano, Sofia, & Smith, Noah A. (2019). Is attention interpretable? *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2931–2951. https://doi.org/10.18653/v1/P19-1282

Seyler, Dominic, Yahya, Mohamed, & Berberich, Klaus. (2015). Generating quiz questions from knowledge graphs. *Proceedings of the 24th International Conference on World Wide Web*, 113–114.

Shapley, Lloyd S. (1953). A value for n-person games. *Contributions to the Theory of Games*, *2*(28), 307–317.

Shrikumar, Avanti, Greenside, Peyton, & Kundaje, Anshul. (2017). Learning important features through propagating activation differences. *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 3145–3153.

Simonyan, Karen, Vedaldi, Andrea, & Zisserman, Andrew. (2014). Deep inside convolutional networks: Visualising image classification models and saliency maps. *Workshop at International Conference on Learning Representations.*

Singh, Chandan, Murdoch, W James, & Yu, Bin. (2019). Hierarchical interpretations for neural network predictions. *International Conference on Learning Representations (ICLR).*

Sixt, Leon, Granz, Maximilian, & Landgraf, Tim. (2019). When explanations lie: Why modified BP attribution fails. *arXiv preprint arXiv:1912.09818.*

Socher, Richard, Lin, Cliff Chiung-Yu, Ng, Andrew Y, & Manning, Christopher D. (2011). Parsing natural scenes and natural language with recursive neural networks. *ICML.*

Socher, Richard, Chen, Danqi, Manning, Christopher D, & Ng, Andrew. (2013). Reasoning with neural tensor networks for knowledge base completion. *Advances in neural information processing systems*, 926–934.

Sorokin, Daniil, & Gurevych, Iryna. (2018). Modeling semantics with gated graph neural networks for knowledge base question answering. *Proceedings of the 27th International Conference on Computational Linguistics*, 3306–3317.

Sturmfels, Pascal, Lundberg, Scott, & Lee, Su-In. (2020). Visualizing the impact of feature attribution baselines. *Distill*, *5*(1), e22.

Sun, Changzhi, Gong, Yeyun, Wu, Yuanbin, Gong, Ming, Jiang, Daxin, Lan, Man, Sun, Shiliang, & Duan, Nan. (2019a). Joint type inference on entities and relations via graph convolutional networks. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 1361–1370. https://doi.org/10.18653/v1/P19-1131

Sun, Haitian, Dhingra, Bhuwan, Zaheer, Manzil, Mazaitis, Kathryn, Salakhutdinov, Ruslan, & Cohen, William. (2018). Open domain question answering using early fusion of knowledge bases and text. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 4231–4242.

Sun, Haitian, Bedrax-Weiss, Tania, & Cohen, William. (2019b). PullNet: Open domain question answering with iterative retrieval on knowledge bases and text. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2380–2390. https://doi.org/10.18653/v1/D19-1242

Sun, Huan, Ma, Hao, Yih, Wen-tau, Tsai, Chen-Tse, Liu, Jingjing, & Chang, Ming-Wei. (2015). Open domain question answering via semantic enrichment. *Proceedings of the 24th International Conference on World Wide Web*, 1045–1055.

Sun, Huan, Ma, Hao, He, Xiaodong, Yih, Wen-tau, Su, Yu, & Yan, Xifeng. (2016). Table cell search for question answering. *Proceedings of the 25th International Conference on World Wide Web*, 771–782. https://doi.org/10.1145/2872427.2883080

Sun, Tony, Gaut, Andrew, Tang, Shirlyn, Huang, Yuxin, ElSherief, Mai, Zhao, Jieyu, Mirza, Diba, Belding, Elizabeth, Chang, Kai-Wei, & Wang, William Yang. (2019c). Mitigating gender bias in natural language processing: Literature review. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 1630–1640. https://doi.org/10.18653/v1/P19-1159

Sun, Zhiqing, Deng, Zhi-Hong, Nie, Jian-Yun, & Tang, Jian. (2019d). Rotate: Knowledge graph embedding by relational rotation in complex space. *International Conference on Learning Representations*.

Sundararajan, Mukund, Taly, Ankur, & Yan, Qiqi. (2017). Axiomatic attribution for deep networks. *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 3319–3328.

Sushil, Madhumita, Šuster, Simon, & Daelemans, Walter. (2018). Rule induction for global explanation of trained models. *Proceedings of the 2018 EMNLP*

*Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, 82–97. https://doi.org/10.18653/v1/W18-5411

Sutskever, Ilya, Vinyals, Oriol, & Le, Quoc V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 3104–3112.

Taghanaki, Saeid Asgari, Havaei, Mohammad, Berthier, Tess, Dutil, Francis, Di Jorio, Lisa, Hamarneh, Ghassan, & Bengio, Yoshua. (2019). Infomask: Masked variational latent representation to localize chest disease. *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 739–747.

Tai, Kai Sheng, Socher, Richard, & Manning, Christopher D. (2015). Improved semantic representations from tree-structured long short-term memory networks. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 1556–1566. https://doi.org/10.3115/v1/P15-1150

Taylor, Wilson L. (1953). "Cloze procedure ": A new tool for measuring readability. *Journalism quarterly*, *30*(4), 415–433.

Thorne, James, & Vlachos, Andreas. (2018). Automated fact checking: Task formulations, methods and future directions. *Proceedings of the 27th International Conference on Computational Linguistics*, 3346–3359.

Tieleman, Tijmen, & Hinton, Geoffrey. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, *4*(2), 26–31.

Toutanova, Kristina, & Chen, Danqi. (2015). Observed versus latent features for knowledge base and text inference. *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, 57–66.

Toutanova, Kristina, Lin, Victoria, Yih, Wen-tau, Poon, Hoifung, & Quirk, Chris. (2016). Compositional learning of embeddings for relation paths in knowledge base and text. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 1434–1444.

Trouillon, Théo, Welbl, Johannes, Riedel, Sebastian, Gaussier, Eric, & Bouchard, Guillaume. (2016). Complex embeddings for simple link prediction. *Proceedings of the 33rd International Conference on Machine Learning*, 2071–2080.

Tu, Ming, Wang, Guangtao, Huang, Jing, Tang, Yun, He, Xiaodong, & Zhou, Bowen. (2019). Multi-hop reading comprehension across multiple documents by reasoning over heterogeneous graphs. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2704–2713. https://doi.org/10.18653/v1/P19-1260

Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N, Kaiser, Łukasz, & Polosukhin, Illia. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 5998–6008.

Veličković, Petar, Cucurull, Guillem, Casanova, Arantxa, Romero, Adriana, Liò, Pietro, & Bengio, Yoshua. (2018). Graph Attention Networks. *International Conference on Learning Representations.*

Vincent, Pascal, Larochelle, Hugo, Bengio, Yoshua, & Manzagol, Pierre-Antoine. (2008). Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th international conference on Machine learning*, 1096–1103.

Vlachos, Andreas, & Riedel, Sebastian. (2014). Fact checking: Task definition and dataset construction. *Proceedings of the ACL 2014 Workshop on Language Technologies and Computational Social Science*, 18–22. https://doi.org/10.3115/v1/W14-2508

Voita, Elena, Talbot, David, Moiseev, Fedor, Sennrich, Rico, & Titov, Ivan. (2019). Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 5797–5808.

Vrandečić, Denny, & Krötzsch, Markus. (2014). Wikidata: A free collaborative knowledgebase. *Communications of the ACM*, *57*(10), 78–85.

Wang, Bailin, Shin, Richard, Liu, Xiaodong, Polozov, Oleksandr, & Richardson, Matthew. (2020a). RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 7567–7578. https://doi.org/10.18653/v1/2020.acl-main.677

Wang, Guangtao, Ying, Rex, Huang, Jing, & Leskovec, Jure. (2020b). Direct multi-hop attention based graph neural network. *arXiv preprint arXiv:2009.14332*.

Welbl, Johannes, Stenetorp, Pontus, & Riedel, Sebastian. (2018). Constructing datasets for multi-hop reading comprehension across documents. *Transactions of the Association for Computational Linguistics*, *6*, 287–302.

Werbos, Paul J. (1982). Applications of advances in nonlinear sensitivity analysis. *System modeling and optimization* (pp. 762–770). Springer.

Wiegreffe, Sarah, & Pinter, Yuval. (2019). Attention is not not explanation. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 11–20. https://doi.org/10.18653/v1/D19-1002

Williams, Ronald J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, *8*(3-4), 229–256.

Wong, Yuk Wah, & Mooney, Raymond. (2007). Learning synchronous grammars for semantic parsing with lambda calculus. *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, 960–967.

Wu, Zonghan, Pan, Shirui, Chen, Fengwen, Long, Guodong, Zhang, Chengqi, & Philip, S Yu. (2020). A comprehensive survey on graph neural networks.

*IEEE Transactions on Neural Networks and Learning Systems*, *32*(1), 4–24.

Xie, Shangsheng, & Lu, Mingming. (2019). Interpreting and understanding graph convolutional neural network using gradient-based attribution method. *arXiv preprint arXiv:1903.03768*.

Xie, Tian, & Grossman, Jeffrey C. (2018). Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Physical review letters*, *120*(14), 145301.

Xiong, Chenyan, & Callan, Jamie. (2015a). Esdrank: Connecting query and documents through external semi-structured data. *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 951–960.

Xiong, Chenyan, & Callan, Jamie. (2015b). Query expansion with freebase. *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, 111–120.

Xu, Keyulu, Hu, Weihua, Leskovec, Jure, & Jegelka, Stefanie. (2019). How powerful are graph neural networks? *International Conference on Learning Representations (ICLR)*.

Xu, Kun, Reddy, Siva, Feng, Yansong, Huang, Songfang, & Zhao, Dongyan. (2016). Question answering on Freebase via relation extraction and textual evidence. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2326–2336. https://doi.org/10.18653/v1/P16-1220

Yang, Bishan, Yih, Scott Wen-tau, He, Xiaodong, Gao, Jianfeng, & Deng, Li. (2015). Embedding entities and relations for learning and inference in knowledge bases (Proceedings of the International Conference on Learning Representations (ICLR) 2015). *Proceedings of the International Conference on Learning Representations (ICLR) 2015*.

Yang, Xiaoyu, Nie, Feng, Feng, Yufei, Liu, Quan, Chen, Zhigang, & Zhu, Xiaodan. (2020). Program enhanced fact verification with verbalization and graph attention network. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 7810–7825. https://doi.org/10.18653/v1/2020.emnlp-main.628

Yao, Xuchen, & Van Durme, Benjamin. (2014). Information extraction over structured data: Question answering with Freebase. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 956–966. https://doi.org/10.3115/v1/P14-1090

Yao, Xuchen. (2015). Lean question answering over freebase from scratch. *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, 66–70.

Yih, Wen-tau, Chang, Ming-Wei, He, Xiaodong, & Gao, Jianfeng. (2015). Semantic parsing via staged query graph generation: Question answering with knowledge base. *Proceedings of the 53rd Annual Meeting of the Associa-*

*tion for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 1321–1331. https://doi.org/10.3115/v1/P15-1128

Yih, Wen-tau, Richardson, Matthew, Meek, Chris, Chang, Ming-Wei, & Suh, Jina. (2016). The value of semantic parse labeling for knowledge base question answering. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 201–206. https://doi.org/10.18653/v1/P16-2033

Yin, Pengcheng, Neubig, Graham, Yih, Wen-tau, & Riedel, Sebastian. (2020). TaBERT: Pretraining for joint understanding of textual and tabular data. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 8413–8426. https://doi.org/10.18653/v1/2020.acl-main.745

Ying, Zhitao, Bourgeois, Dylan, You, Jiaxuan, Zitnik, Marinka, & Leskovec, Jure. (2019). GNNExplainer: Generating explanations for graph neural networks. *Advances in Neural Information Processing Systems*, 9240–9251.

Yu, Mo, Chang, Shiyu, Zhang, Yang, & Jaakkola, Tommi. (2019). Rethinking cooperative rationalization: Introspective extraction and complement control. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 4094–4103. https://doi.org/10.18653/v1/D19-1420

Yu, Tao, Zhang, Rui, Yang, Kai, Yasunaga, Michihiro, Wang, Dongxu, Li, Zifan, Ma, James, Li, Irene, Yao, Qingning, Roman, Shanelle, Zhang, Zilin, & Radev, Dragomir. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 3911–3921. https://doi.org/10.18653/v1/D18-1425

Zaheer, Manzil, Kottur, Satwik, Ravanbakhsh, Siamak, Poczos, Barnabas, Salakhutdinov, Russ R, & Smola, Alexander J. (2017). Deep sets. *Advances in neural information processing systems*, 3391–3401.

Zaheer, Manzil, Guruganesh, Guru, Dubey, Kumar Avinava, Ainslie, Joshua, Alberti, Chris, Ontanon, Santiago, Pham, Philip, Ravula, Anirudh, Wang, Qifan, Yang, Li, et al. (2020). Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, *33*.

Zelle, John M., & Mooney, Raymond J. (1996). Learning to parse database queries using inductive logic programming. *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, 1050–1055.

Zettlemoyer, Luke S., & Collins, Michael. (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, 658–666.

Zhang, Li, Zhang, Shuo, & Balog, Krisztian. (2019). Table2vec: Neural word and entity embeddings for table population and retrieval. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1029–1032.

Zhang, Shuo, & Balog, Krisztian. (2018). Ad hoc table retrieval using semantic similarity. *Proceedings of the 2018 World Wide Web Conference*, 1553–1562.

Zhang, Shuo, & Balog, Krisztian. (2019). Auto-completion for data cells in relational tables. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 761–770.

Zhang, Yuhao, Qi, Peng, & Manning, Christopher D. (2018). Graph convolution over pruned dependency trees improves relation extraction. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2205–2215. https://doi.org/10.18653/v1/D18-1244

Zhong, Wanjun, Tang, Duyu, Feng, Zhangyin, Duan, Nan, Zhou, Ming, Gong, Ming, Shou, Linjun, Jiang, Daxin, Wang, Jiahai, & Yin, Jian. (2020). LogicalFactChecker: Leveraging logical operations for fact checking with graph module network. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 6053–6065. https://doi.org/10.18653/v1/2020.acl-main.539

Zhu, Hao, Lin, Yankai, Liu, Zhiyuan, Fu, Jie, Chua, Tat-Seng, & Sun, Maosong. (2019). Graph neural networks with generated parameters for relation extraction. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 1331–1339.

Zintgraf, Luisa M, Cohen, Taco S., Adel, Tameem, & Welling, Max. (2017). Visualizing deep neural network decisions: Prediction difference analysis. *International Conference on Learning Representations (ICLR)*.

# Samenvatting

Gestructureerde gegevens komen veelvuldig in de wereld voor, evenals de NLP-toepassingen waarmee gepoogd wordt conclusies te trekken over dergelijke gegevens. Ondanks hun succes hebben moderne neurale netwerkmodellen vaak moeite om gestructureerde informatie op te nemen. In dit proefschrift, *Het Opnemen van Structuur in Neurale Modellen voor Taalverwerking*, onderzoeken wij hoe effectieve neurale netwerkmodellen kunnen worden gebouwd om gestructureerde gegevens op te nemen voor de interpretatie van natuurlijke taal. Een natuurlijke vorm van weergave van gestructureerde informatie is door middel van grafen. De onlangs geïntroduceerde *Graph Neural Networks* (GNN's) bieden voor neurale netwerken de mogelijkheid om conclusies te trekken over grafen door middel van leerbare *message passing* functies. Als eerste introduceren we het effectief eerste GNN-model dat geschikt is voor de gerichte multirelationele gegevens die worden aangetroffen in de gangbare vormen van gestructureerde gegevens die relevant zijn voor NLP toepassingen, zoals kennisbanken (KB's). We bestuderen *encoders* van structuur voor *relational link prediction*, *question answering* en *fact verification.* Een grote uitdaging is de niet-interpreteerbare, *black-box*-aard van dergelijke *encoders.* Om dit probleem te verkleinen introduceren wij een nieuwe techniek voor het interpreteren van de voorspellingen van GNN's. Wij presenteren onze inspanningen in vier hoofdstukken:

1. Voor het voorspellen van relationele verbanden in kennisbanken voeren we *Relational Graph Convolutional Network (R-GCN) encoders* in. R-GCN's zijn een nieuwe variant van GNN's die geschikt zijn voor het modelleren van de gerichte, multirelationele gegevens die men in kennisbanken aantreft. Door onze R-GCN-*encoder* te combineren met een factorisatiedecoder uit de literatuur, bereikten we op het moment van publicatie state-of-the-art prestaties op de FB15k-237-dataset. Ons model presteert vooral goed voor gecompliceerde gevolgtrekkingen met knooppunten van hoge graad en zeldzame relaties.

2. We introduceren twee op GNN gebaseerde modellen voor *factoid question answering* over KB's. Deze modellen zijn gebaseerd op het kiezen van ofwel individuele antwoordknooppunten, of een beste pad naar het antwoord. Naast de in Hoofdstuk 3 geïntroduceerde R-GCN stellen we een variant voor die gates gebruikt om de te gebruiken kanten van de graaf te beperken. Bij deze keuze moedigen we spaarzaamheid aan door middel van een $L_1$-boete. De verbetering, die het gevolg is van ijlheid van de graaf, laat zien hoe op GNN gebaseerde modellen profiteren van het wegfilteren van overtollige kanten.

3. We introduceren een nieuw model voor *fact verification* over open verzamelingen van tabellen, waarbij we een RoBERTa-*encoder* voor gelineariseerde tabellen combineren met een *cross-attention*-mechanisme voor het samenvoegen van bewijsstukken. Linearisatie is een belangrijk alternatief voor het modelleren van structuur bij grafen. Wanneer we opereren in het open domein, bereikt onze aanpak prestaties die vergelijkbaar zijn met de huidige state-of-the-art in het gesloten domein; wanneer we opereren in het gesloten domein, resulteert onze aanpak in een nieuwe state-of-the-art. Verder voeren we twee nieuwe strategieën in voor het exploiteren van datasets met een gesloten domein om de prestaties in het open domein te verbeteren. Deze strategieën zijn gebaseerd op doelen die gezamenlijk *claim truth* en *evidence reranking* modelleren.

4. Onze ervaring laat zien dat interpreteerbaarheid een belangrijke kwestie ist voor GNN's (en voor *transformers*, zoals gebruikt in hoofdstuk 5). We stellen Graph Mask voor, een nieuwe post-hoc interpretatietechniek voor GNN-gebaseerde modellen. Door *end-to-end* differentieerbare nul-één-poorten voor elk bericht te leren, produceert Graph Mask getrouwe, schaalbare en eenvoudig te begrijpen verklaringen voor hoe GNN's tot specifieke voorspellingen komen. We testen onze aanpak op een synthetische taak met een bekende gouden standaard voor trouw. Hiermee tonen we aan dat Graph Mask gunstig afsteekt bij de huidige alternatieven. Daarnaast passen we onze techniek toe om de voorspellingen van twee NLP-modellen uit de literatuur te analyseren: een *semantic role labeling*-model en een *question answering*-model.

# Abstract

Structured data is abundant in the world, as is the multitude of NLP applications seeking to perform inferences over such data. Despite their success, modern neural network models often struggle to incorporate structured information. In this thesis, *Incorporating Structure into Neural Models for Language Processing*, we investigate how to build effective neural network models to incorporate structured data for natural language understanding. Graphs are a natural form of representation for structural information, and the recently proposed Graph Neural Networks (GNNs) allow neural networks to perform inference over graphs through learnable message passing functions. We begin by introducing effectively the first GNN model suitable for the directed, multirelational data found in common forms of structured data relevant to NLP applications, such as knowledge bases (KBs). We study structural encoders for relational link prediction, question answering, and fact verification. A significant challenge is the uninterpretable, black-box nature of such encoders. To alleviate this problem, we introduce a novel technique for interpreting the predictions of GNNs. Our efforts are presented in four chapters:

1. We propose Relational Graph Convolutional Network (R-GCN) encoders for relational link prediction in knowledge bases. R-GCNs are a novel variant of GNNs suitable for modeling the directed, multirelational data found in KBs. By combining our R-GCN encoder with a factorization decoder from the literature, we achieved state-of-the-art performance on the FB15k-237 dataset at the time of publication. Our model performs especially well for complicated inferences involving high-degree vertices and rare relations.

2. We introduce two GNN-based models for factoid question answering over KBs, relying either on choosing individual answer vertices or on choosing a best path to the answer. In addition to the R-GCN introduced in Chapter 3, we propose a variant that uses gates to limit which edges are used. We encourage sparsity in this choice through an $L_1$-penalty. The improvement

derived from sparsity demonstrates how GNN-based models benefit from filtering out superfluous edges.

3. We introduce a novel model for fact verification over open collections of tables, combining a RoBERTa-encoder for linearised tables with a cross-attention mechanism for fusing evidence documents. Linearisation represents an important alternative to graphs for modeling structure. When operating in the open domain, our approach achieves performance on par with the current closed-domain state of the art; when operating in the closed domain, our approach sets a new state of the art. We also introduce two novel strategies for exploiting closed-domain datasets to improve performance in the open domain, relying on objectives which jointly model claim truth and evidence reranking.

4. As our experience shows, interpretability is an important issue for GNNs (and for transformers, as used in Chapter 5). We propose GraphMask, a novel post-hoc interpretation technique for GNN-based models. By learning end-to-end differentiable zero-one gates for every message, GraphMask produces faithful, scalable, and easily understood explanations for how GNNs arrive at specific predictions. We test our approach on a synthetic task with a known gold standard for faithfulness, demonstrating that GraphMask compares favourably to current alternatives. We furthermore apply our technique to analyze the predictions of two NLP models from the literature -— a semantic role labeling model, and a question answering model.