Combining Representation Learning with Logic for Language Processing

Tim Rocktäschel

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
of
University College London.

Department of Computer Science
University College London

December 27, 2017

To Paula, Emily, Sabine, and Lutz.

Acknowledgements

I am deeply grateful to my supervisor and mentor Sebastian Riedel. He always has been a great source of inspiration and supportive and encouraging in all matters. I am particularly amazed by the trust he put into me from the first time we met, the freedom he gave me to pursue ambitious ideas, his contagious optimistic attitude, and the many opportunities he presented to me. There is absolutely no way I could have wished for a better supervisor.

I would like to thank my second supervisors Thore Graepel and Daniel Tarlow for their feedback, as well as Sameer Singh whose collaboration and guidance made my start into the Ph.D. very smooth, motivating, and fun. Furthermore, I thank Edward Grefenstette, Karl Moritz Hermann, Thomas Kociský and Phil Blunsom, who I was fortunate to work with during my internship at DeepMind in Summer 2015. I am thankful for the guidance by Thomas Demeester, Andreas Vlachos, Pasquale Minervini, Pontus Stenetorp, Isabelle Augenstein, and Jason Naradowsky during their time at the University College London Machine Reading lab. In addition, thanks to Dirk Weissenborn for many fruitful discussions. I would also like to thank Ulf Leser, Philippe Thomas, and Roman Klinger for preparing me well for the Ph.D. during my studies at Humboldt-Universität zu Berlin.

I had the pleasure to work with brilliant students at University College London. Thanks to Michal Daniluk, Luke Hewitt, Ben Eisner, Vladyslav Kolesnyk, Avishkar Bhoopchand, and Nayen Pankhania for their hard work and trust. Ph.D. life would not have been much fun without my lab mates Matko Bosnjak, George Spithourakis, Johannes Welbl, Marzieh Saeidi, and Ivan Sanchez. Thanks to Matko, Dirk, Pasquale, Thomas, Johannes, and Sebastian for feedback on this thesis. Furthermore, I thank my examiners Luke Dickens and Charles Sutton for their extremely helpful in-depth comments and corrections of this thesis. Thank you, Federation Coffee in Brixton, for making the best coffee in the world.

Many thanks to Microsoft Research for supporting my work through its Ph.D. Scholarship Programme, and to Google Research for a Ph.D. Fellowship in Natural

Language Processing. Thanks to their generous support, as well as the funding from the University College London Computer Science Department, I was able to travel to all conferences that I wanted to attend.

I am greatly indebted and thankful to my parents Sabine and Lutz. They sparked my interest in science, but more importantly, they ensured that I had a fantastic childhood. I always felt loved, supported, and well protected from the many troubles in life. Furthermore, I would like to thank Christa, Ulrike, Tillmann, Hella, Hans, Gretel, and Walter for their unconditional support over the last years.

Lastly, thanks to the two wonderful women in my life, Paula and Emily. Thank you, Paula, for keeping up with my ups and downs during the Ph.D., your love, motivation, and support. Thank you for giving me a family, and for making us feel at home wherever we are. Emily, you are the greatest wonder and joy in my life.

Declaration

I, Tim Rocktäschel confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

TIM ROCKTÄSCHEL

Abstract

The current state-of-the-art in many natural language processing and automated knowledge base completion tasks is held by representation learning methods which learn distributed vector representations of symbols via gradient based optimization. They require little or no hand-crafted features, thus avoiding the need for most preprocessing steps and task-specific assumptions. However, in many cases representation learning requires a large amount of annotated training data to generalize well to unseen data. Such labeled training data is provided by human annotators who often use formal logic as the language for specifying annotations.

This thesis investigates different combinations of representation learning methods with logic for reducing the need for annotated training data, and for improving generalization. We introduce a mapping of function-free first-order logic rules to loss functions that we combine with neural link prediction models. Using this method, logical prior knowledge is directly embedded in vector representations of predicates and constants. We find that this method learns accurate predicate representations for which no or little training data is available, while at the same time generalizing to other predicates not explicitly stated in rules. However, this method relies on grounding first-order logic rules, which does not scale to large rule sets. To overcome this limitation, we propose a scalable method for embedding implications in a vector space by only regularizing predicate representations. Subsequently, we explore a tighter integration of representation learning and logical deduction. We introduce an end-to-end differentiable prover – a neural network that is recursively constructed from Prolog's backward chaining algorithm. The constructed network allows us to calculate the gradient of proofs with respect to symbol representations and to learn these representations from proving facts in a knowledge base. In addition to incorporating complex first-order rules, it induces interpretable logic programs via gradient descent. Lastly, we propose recurrent neural networks with conditional encoding and a neural attention mechanism for determining the logical relationship between two natural language sentences.

Impact Statement

Machine learning, and representation learning in particular, is ubiquitous in many applications nowadays. Representation learning requires little or no hand-crafted features, thus avoiding the need for task-specific assumptions. At the same time, it requires a large amount of annotated training data. Many important domains lack such large training sets, for instance, because annotation is too costly or domain expert knowledge is generally hard to obtain.

The combination of neural and symbolic approaches investigated in this thesis has only recently regained significant attention due to advances of representation learning research in certain domains and, more importantly, their lack of success in other domains. The research conducted under this Ph.D. investigated ways of training representation learning models using explanations in form of function-free first-order logic rules in addition to individual training facts. This opens up the possibility of taking advantage of the strong generalization of representation learning models, while still being able to express domain expert knowledge. We hope that this work will be particularly useful for applying representation learning in domains where annotated training data is scarce, and that it will empower domain experts to train machine learning models by providing explanations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"*We are attempting to replace symbols by vectors so we can replace logic by algebra.*"

— Yann LeCun

The vast majority of knowledge produced by mankind is nowadays available in a digital but unstructured form such as images or text. It is hard for algorithms to extract meaningful information from such data resources, let alone to reason with it. This issue is becoming more severe as the amount of unstructured data is growing very rapidly.

In recent years, remarkable successes in processing unstructured data have been achieved by representation learning methods which automatically learn abstractions from large collections of training data. This is achieved by processing input data using artificial neural networks whose weights are adapted during training. Representation learning lead to breakthroughs in applications such as automated Knowledge Base (KB) completion [Nickel et al., 2012, Riedel et al., 2013, Socher et al., 2013, Chang et al., 2014, Yang et al., 2015, Neelakantan et al., 2015, Toutanova et al., 2015, Trouillon et al., 2016], as well as Natural Language Processing (NLP) applications like paraphrase detection [Socher et al., 2011, Hu et al., 2014, Yin and Schütze, 2015], machine translation [Bahdanau et al., 2014], image captioning [Xu et al., 2015], speech recognition [Chorowski et al., 2015] and sentence summarization [Rush et al., 2015], to name just a few.

Representation learning methods achieve remarkable results, but they usually rely on a large amount of annotated training data. Moreover, since representation learning operates on a subsymbolic level (for instance by replacing words with lists of real numbers – so-called *vector representations* or *embeddings*), it is hard to determine why we obtain a certain prediction, let alone how to correct systematic errors or how to incorporate domain and commonsense knowledge. In fact, a recent General Data Protection Regulation by the European Union introduces the "right to explanation" of decisions by algorithms and machine learning models that affect users [Council of the European Union, 2016], to be enacted in 2018. This has profound implications for the future development and research of machine learning algorithms [Goodman and Flaxman, 2016], especially for nowadays commonly used representation learning methods. Moreover, for many domains of interests there is not enough annotated training data, which renders applying recent representation learning methods difficult.

Many of these issues do not exist with purely symbolic approaches. For instance, given a KB of facts and first-order logic rules, we can use Prolog to obtain an answer as well as a proof for a query to this KB. Furthermore, we can easily incorporate domain knowledge by adding more rules. However, rule-based system do not generalize to new questions. For instance, given that an apple is a fruit and apples are similar to oranges, we would like to infer that oranges are likely also fruits.

To summarize, symbolic rule-based systems are interpretable and easy to modify. They do not need large amounts of training data and we can easily incorporate domain knowledge. On the other hand, learning subsymbolic representations requires a lot of training data. The trained models are generally opaque and it is hard to incorporate domain knowledge. Consequently, we would like to develop methods that take the best of both worlds.

1.1 Aims

In this thesis, we are investigating the combination of representation learning with first-order logic rules and reasoning. Representation learning methods achieve strong generalization by learning subsymbolic vector representations that can capture similarity and even logical relationships directly in a vector space [Mikolov et al., 2013]. Symbolic representations, on the other hand, allow us to formalize domain and commonsense knowledge using rules. For instance, we can state that every `human` is `mortal`, or that every `grandfather` is a `father` of a `parent`. Such rules are often worth many training facts. Furthermore, by using symbolic representations we can take advantage of algorithms for multi-hop reasoning like Prolog's backward chaining algorithm [Colmerauer, 1990]. Backward chaining is not only widely used for multi-hop question answering in KBs, but it also provides us with proofs in addition to the answer for a question. However, such symbolic reasoning is relying on a complete specification of background and commonsense knowledge in logical form. As an example, let us assume we are asking for the `grandpa` of a person, but only know the `grandfather` of that person. If there is no explicit rule connecting `grandpa` to `grandfather`, we will not find an answer. However, given a large

KB, we can use representation learning to learn that `grandpa` and `grandfather` mean the same thing, or that a `lecturer` is similar to a `professor` [Nickel et al., 2012, Riedel et al., 2013, Socher et al., 2013, Chang et al., 2014, Yang et al., 2015, Neelakantan et al., 2015, Toutanova et al., 2015, Trouillon et al., 2016]. This becomes more relevant once we do not only want to reason with structured relations but also use textual patterns as relations [Riedel et al., 2013].

The problem that we seek to address in this thesis is how symbolic logical knowledge can be combined with representation learning to make use of the best of both worlds. Specifically, we investigate the following research questions.

- Can we efficiently incorporate domain and commonsense knowledge in form of rules into representation learning methods?
- Can we use rules to alleviate the need for large amounts of training data while still generalizing beyond what is explicitly stated in these rules?
- Can we synthesize representation learning with symbolic multi-hop reasoning as used for automated theorem proving?
- Can we learn rules directly from data using representation learning?
- Can we determine the logical relationship between natural language sentences using representation learning?

1.2 Contributions

This thesis makes the following core contributions.

Regularizing Representations by First-order Logic Rules We introduce a method for incorporating function-free first-order logic rules directly into vector representations of symbols, which avoids the need for symbolic inference. Instead of symbolic inference, we regularize symbol representations by given rules such that logical relationships hold implicitly in the vector space (Chapter 3). This is achieved by mapping propositional logical rules to differentiable loss terms so that we can calculate the gradient of a given rule with respect to symbol representations. Given a first-order logic rule, we stochastically ground free variables using constants in the domain, and add the resulting loss term for the propositional rule to the training objective of a neural link prediction model for automated KB completion. This allows us to infer relations with little or no training facts in a KB. While mapping logical rules to soft rules using algebraic operations has a long tradition (*e.g.* in Fuzzy logic), our contribution is the connection to representation learning, *i.e.*, using such rules to directly learning better vector representations of symbols that can be used to improve performance on a downstream task such as automated KB completion. Content in this chapter first appeared in the following two publications:

> Tim Rocktäschel, Matko Bosnjak, Sameer Singh and Sebastian Riedel. 2014. Low-Dimensional Embeddings of Logic. In *Proceedings of Association for Computational Linguistics Workshop on Semantic Parsing (SP'14).*

> Tim Rocktäschel, Sameer Singh and Sebastian Riedel. 2015. Injecting Logical Background Knowledge into Embeddings for Relation Extraction. In *Proceedings of North American Chapter of the Association for Computational Linguistics – Human Language Technologies (NAACL HLT 2015).*

Lifted Regularization of Predicate Representations by Implications For the subclass of first-order logic implication rules, we present a scalable method that is independent of the size of the domain of constants, that generalizes to unseen constants, and that can be used with a broader class of training objectives (Chapter 4). Instead of relying on stochastic grounding, we use implication rules directly as regularizers for predicate representations. Compared to the method in Chapter 3, this method is independent of the number of constants and ensures that a given implication between two predicates holds for any possible pair of constants at test time. Our method is based on Order Embeddings [Vendrov et al., 2016] and our contribution is the extension to the task of automated KB completion which requires constraining entity representations to be non-negative. This chapter is based on the following two publications:

> Thomas Demeester, Tim Rocktäschel and Sebastian Riedel. 2016. Regularizing Relation Representations by First-order Implications. In *Proceedings of North American Chapter of the Association for Computational Linguistics (NAACL) Workshop on Automated Knowledge Base Construction (AKBC).*

> Thomas Demeester, Tim Rocktäschel and Sebastian Riedel. 2016. Lifted Rule Injection for Relation Embeddings. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP).*

My contribution to this work is the conceptualization of the model, the design of experiments, and the extraction of commonsense rules from WordNet.

End-to-end Differentiable Proving Current representation learning and neural link prediction models have deficits when it comes to complex multi-hop inferences such as transitive reasoning [Bouchard et al., 2015, Nickel et al., 2016]. Automated theorem provers, on the other hand, have a long tradition in computer science and provide elegant ways to reason with symbolic knowledge. In Chapter 5, we propose Neural Theorem Provers (NTPs): end-to-end differentiable theorem provers for automated KB completion based on neural networks that are recursively constructed and inspired by Prolog's backward chaining algorithm. By doing so, we can calculate the gradient of a proof success with respect to symbol representations in a KB. This allows us to learn symbol representations directly from facts in a KB, and to make use of the similarities of symbol representations and provided rules in proofs. In addition, we demonstrate that we can induce interpretable rules of predefined structure. On three out of four benchmark KBs, our method outperforms ComplEx [Trouillon et al., 2016], a state-of-the-art neural link prediction model. Work in this chapter appeared in:

> Tim Rocktäschel and Sebastian Riedel. 2016. Learning Knowledge Base Inference with Neural Theorem Provers. In *Proceedings of North American Chapter of the Association for Computational Linguistics (NAACL) Workshop on Automated Knowledge Base Construction (AKBC).*

> Tim Rocktäschel and Sebastian Riedel. 2017. End-to-End Differentiable Proving. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems (NIPS).*

Recognizing Textual Entailment with Recurrent Neural Networks Representation learning models such as Recurrent Neural Networks (RNNs) can be used to map natural language sentences to fixed-length vector representations, which has been successfully applied for various downstream NLP tasks including Recognizing Textual Entailment (RTE). In RTE, the task is to determine the logical relationship between two natural language sentences. This has so far been either approached by NLP pipelines with hand-crafted features, or neural network architectures that independently map the two sentences to fixed-length vector representations. Instead of encoding the two sentences independently, we propose a model that encodes the second sentence conditioned on an encoding of the first sentence. Furthermore, we apply a neural attention mechanism to bridge the hidden state bottleneck of the RNN (Chapter 6). Work in this chapter first appeared in:

> Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomas Kocisky and Phil Blunsom. 2016. Reasoning about Entailment with Neural Attention. In *Proceedings of International Conference on Learning Representations (ICLR).*

1.3 Thesis Structure

In Chapter 2, we provide background on representation learning, computation graphs, first-order logic, and the notation used throughout the thesis. Furthermore, we explain the task of automated KB completion and describe neural link prediction and path-based approaches that have been proposed for this task. Chapter 3 introduces a method for regularizing symbol representations by first-order logic rules. In Chapter 4, we subsequently focus on direct implications between predicates, a subset of first-order logic rules. For this class of rules, we provide an efficient method by directly regularizing predicate representations. In Chapter 5, we introduce a recursive construction of a neural network for automated KB completion based on Prolog's backward chaining algorithm. Chapter 6 presents a RNN for RTE based on conditional encoding and a neural attention mechanism. Finally, Chapter 7 concludes the thesis with a discussion of limitations, open issues, and future research avenues.

Chapter 2

Background

This chapter introduces core methods used in the thesis. Section 2.1 explains function approximation with neural networks and backpropagation. Subsequently, Section 2.2 introduces function-free first-order logic, the backward chaining algorithm, and inductive logic programming. Finally, Section 2.3 discusses prior work on automated knowledge base completion, linking the first two sections together.

2.1 Function Approximation with Neural Networks

In this thesis, we consider models that can be formulated as differentiable functions $f_{\boldsymbol{\theta}} : \mathcal{X} \rightarrow \mathcal{Y}$ parameterized by $\boldsymbol{\theta} \in \Theta$. Our task is to find such functions, *i.e.*, to learn parameters $\boldsymbol{\theta}$ from a set of training examples $\mathcal{T} = \{(x_i, y_i)\}$ where $x_i \in \mathcal{X}$ is the input and $y_i \in \mathcal{Y}$ some desired output of the ith training example. Both, x_i and y_i, can be structured objects. For instance, x_i could be a fact about the world, like `directedBy`(INTERSTELLAR, NOLAN), and y_i a corresponding target truth score (*e.g.* 1.0 for TRUE).

We define a loss function $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \times \Theta \rightarrow \mathbb{R}$ that measures the discrepancy between a provided output y and a predicted output $f_{\boldsymbol{\theta}}(x)$ on an input x, given a current setting of parameters $\boldsymbol{\theta}$. We seek to find those parameters $\boldsymbol{\theta}^*$ that minimize this discrepancy on a training set. We denote the global loss over the entire training data as $\mathfrak{L}$. Our learning problem can thus be written as

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathfrak{L} = \arg\min_{\boldsymbol{\theta}} \frac{1}{|\mathcal{T}|} \sum_{(x,y) \in \mathcal{T}} \mathcal{L}(f_{\boldsymbol{\theta}}(x), y, \boldsymbol{\theta}). \tag{2.1}$$

Note that $\mathcal{L}$ is also a function of $\boldsymbol{\theta}$, since we might not only want to measure the discrepancy between given and predicted outputs, but also use a regularizer on the parameters to improve generalization. Sometimes, we omit $\boldsymbol{\theta}$ in $\mathcal{L}$ for brevity. As $\mathcal{L}$ and $f_{\boldsymbol{\theta}}$ are differentiable functions, we can use gradient-based optimization methods, such as Stochastic Gradient Descent (SGD) [Nemirovski and Yudin, 1978], for iteratively updating $\boldsymbol{\theta}$ based on mini-batches $\mathcal{B} \subseteq \mathcal{T}$ of the training data[1]

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}_t} \frac{1}{|\mathcal{B}|} \sum_{(x,y) \in \mathcal{B}} \mathcal{L}(f_{\boldsymbol{\theta}}(x), y, \boldsymbol{\theta}_t) \tag{2.2}$$

where η denotes a learning rate, and $\nabla_{\boldsymbol{\theta}}$ denotes the differentiation operation of the loss with respect to parameters, given the current batch at time step t.

2.1.1 Computation Graphs

A useful abstraction for defining models as differentiable functions are *computation graphs* which illustrate the computations carried out by a model more precisely [Goodfellow et al., 2016]. In such a directed acyclic graph, nodes represent variables and directed edges from one or multiple nodes to another node correspond to a differentiable operation. As variables we consider scalars, vectors, matrices, and, more generally, higher-order tensors.[2] We denote scalars by lower case letters x, vectors by bold lower case letters $\boldsymbol{x}$, matrices by bold capital letters $\boldsymbol{X}$, and higher-order tensors by Euler script letters $\mathcal{X}$. Variables can either be inputs, outputs, or parameters of a model.

Figure 2.1a shows a simple computation graph that calculates $z = \sigma(\boldsymbol{x}^\top \boldsymbol{y})$. Here, σ and `sigm` refer to the element-wise (or scalar) sigmoid operation

$$\sigma(\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{x}}} \tag{2.3}$$

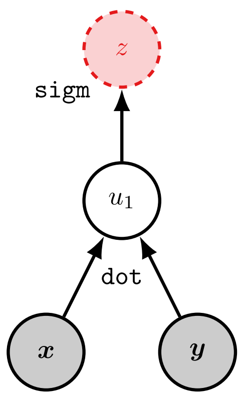
and `dot` and $\boldsymbol{x}^\top \boldsymbol{y}$ denote the dot product between two vectors. Furthermore, we name the ith intermediate expression as u_i. Figure 2.1b shows a slightly more complex computation graph with two parameters $\boldsymbol{w}$ and b. This computation graph in fact represents logistic regression $f(\boldsymbol{x}) = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)$.

2.1.2 From Symbols to Subsymbolic Representations

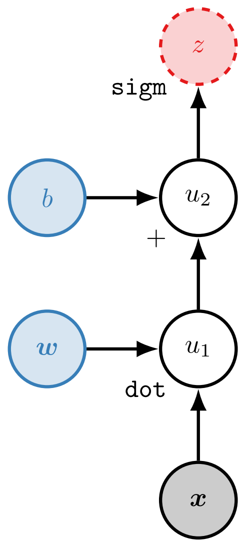
In this thesis, we will use neural networks to learn representations of symbols. For instance, such symbols can be words, constants, or predicate names. When we say we learn a subsymbolic representation for symbols, we mean that we map symbols to fixed-dimensional vectors or more generally tensors. This can be done by first

[1]Note that there are many alternative methods for minimizing the loss in Eq. 2.1, but all models in this thesis are optimized by variants of SGD.

[2]For implementation purpose we will also consider structured objects over such tensors, like tuples and lists. Current deep learning libraries such as Theano [Al-Rfou et al., 2016], Torch [Collobert et al., 2011] and TensorFlow [Abadi et al., 2016] come with support for tuples and lists. However, for brevity we leave them out of the description here.



(a) $z = \sigma(\boldsymbol{x}^\top \boldsymbol{y})$



(b) $z = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)$

Figure 2.1: Two simple computation graphs with inputs (gray), parameters (blue), intermediate variables u_i, and outputs (dashed). Operations are shown next to the nodes.

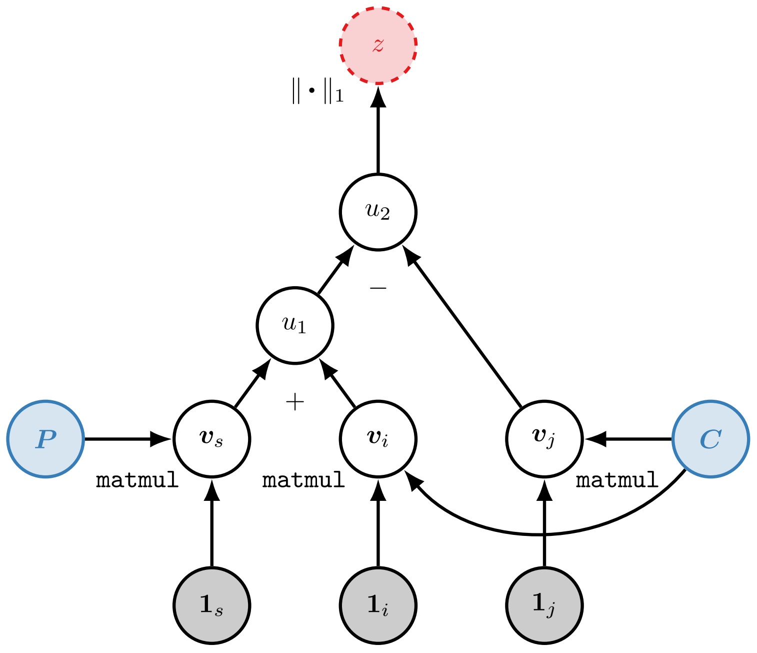


Figure 2.2: Computation graph for $z = \|\boldsymbol{C}\mathbf{1}_i + \boldsymbol{P}\mathbf{1}_s - \boldsymbol{C}\mathbf{1}_j\|_1$.

enumerating all symbols and then assigning the number i to the ith symbol. Let $\mathcal{S}$ be the set of all symbols. We denote the one-hot vector for the ith symbol as $\mathbf{1}_i \in \{0,1\}^{|\mathcal{S}|}$, which is 1 at index i and 0 everywhere else. Figure 2.2 shows a computation graph whose inputs are one-hot vectors of some symbols with indices s, i, and j. In the first layer, these one-hot vectors are mapped to dense vector representations via a matrix multiplication with so-called embedding lookup matrices ($\boldsymbol{P}$ and $\boldsymbol{C}$ in this case).

This computation graph corresponds to a neural link prediction model that we explain in more detail in Section 2.3.2. Here it serves only as an illustration for how symbols can be mapped to vector representations. In the remainder of this thesis, we will often omit the embedding layer for clarity. The goal is to learn symbol representations such as $\boldsymbol{v}_s$, $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ automatically from data. To this end, we need to be able to calculate the gradient of the output of the computation graph with respect to its parameters (the embedding lookup matrices in this case).

2.1.3 Backpropagation

For learning from data, we need to be able to calculate the gradient of a loss with respect to all model parameters. As we assume all operations in the computation graph are differentiable, we can recursively apply the chain rule of calculus.

Chain Rule of Calculus Assume we are given a composite function $\boldsymbol{z} = f(\boldsymbol{y}) = f(g(\boldsymbol{x}))$ with $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ and $g : \mathbb{R}^l \rightarrow \mathbb{R}^n$. The chain rule allows us to decompose the calculation of $\nabla_{\boldsymbol{x}}\boldsymbol{z}$, *i.e.*, the gradient of the entire computation $\boldsymbol{z}$ with respect to $\boldsymbol{x}$, as follows [Goodfellow et al., 2016]

$$\nabla_{\boldsymbol{x}}\boldsymbol{z} = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\top} \nabla_{\boldsymbol{y}}\boldsymbol{z}. \tag{2.4}$$

Here, $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ is the Jacobian matrix of g, *i.e.*, the matrix of partial derivatives, and $\nabla_{\boldsymbol{y}}\boldsymbol{z}$ is the gradient of $\boldsymbol{z}$ with respect to $\boldsymbol{y}$. Note that this approach generalizes to matrices and higher-order tensors by reshaping them to vectors before the gradient calculation (vectorization) and back to their original shape afterwards.

Backpropagation uses the chain rule to recursively define the efficient calculation of gradients of parameters (and inputs) in the computation graph by avoiding recalculation of previously calculated expressions. This is achieved via dynamic programming, *i.e.*, storing previously calculated gradient expressions and reusing them for later gradient calculations. We refer the reader to Goodfellow et al. [2016] for details. In order to run backpropagation with a differentiable operation f that we want to use in a computation graph, all we need to ensure is that this function is differentiable with respect to each one of its inputs.

Example Let us take the computation graph depicted in Fig. 2.1a as an example. Assume we are given an upstream gradient ∇_z and want to compute the gradient of z with respect to the inputs $\boldsymbol{x}$ and $\boldsymbol{y}$. The computations carried out by backpropagation

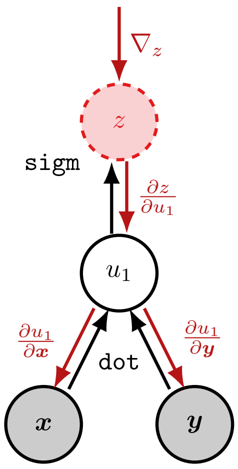

∇_z
z
sigm
$\frac{\partial z}{\partial u_1}$
u_1
$\frac{\partial u_1}{\partial \boldsymbol{x}}$
dot
$\frac{\partial u_1}{\partial \boldsymbol{y}}$
$\boldsymbol{x}$
$\boldsymbol{y}$

Figure 2.3: Backward pass for the computation graph shown in Fig. 2.1a.

are depicted in Fig. 2.3. For instance, by recursively applying the chain rule we can calculate $\nabla_{\boldsymbol{x}} z$ as follows:

$$\nabla_{\boldsymbol{x}} z = \frac{\partial z}{\partial \boldsymbol{x}} = \frac{\partial z}{\partial u_1} \frac{\partial u_1}{\partial \boldsymbol{x}} = \sigma(u_1)(1 - \sigma(u_1))\boldsymbol{y}. \tag{2.5}$$

Note that the computation of $\frac{\partial z}{\partial u_1}$ can be reused for calculating $\nabla_{\boldsymbol{y}} z$. We get the gradient of the entire computation graph (including upstream nodes) with respect to $\boldsymbol{x}$ via $\nabla_z \nabla_{\boldsymbol{x}} z$. Later we will use computation graphs where nodes are used in multiple downstream computations. Such nodes receive multiple gradients from downstream nodes during backpropagation, which are summed up to calculate the gradient of the computation graph with respect to the variable represented by the node.

In Chapter 3, we will use backpropagation for computing the gradient of differentiable propositional logic rules with respect to vector representations of symbols to develop models that combine representation learning with first-order logic. In Chapter 5, we take this further and construct a computation graph for all possible proofs in a Knowledge Base (KB) using the backward chaining algorithm. This will allow us to calculate the gradient of proofs with respect to symbol representations and to induce rules using gradient descent. Finally, in Chapter 6, we will use Recurrent Neural Networks (RNNs), *i.e.*, computation graphs that are dynamically constructed for input varying-length input sequences of word representations.

	Prolog/Datalog Syntax	List Representation
1	`fatherOf`(ABE, HOMER).	[[`fatherOf`, ABE, HOMER]]
2	`parentOf`(HOMER, LISA).	[[`parentOf`, HOMER, LISA]]
3	`parentOf`(HOMER, BART).	[[`parentOf`, HOMER, BART]]
4	`grandpaOf`(ABE, LISA).	[[`grandpaOf`, ABE, LISA]]
5	`grandfatherOf`(ABE, MAGGIE).	[[`grandfatherOf`, ABE, MAGGIE]]
6	`grandfatherOf`(X_1, Y_1) :– `fatherOf`(X_1, Z_1), `parentOf`(Z_1, Y_1).	[[`grandfatherOf`, X_1, Y_1], [`fatherOf`, X_1, Z_1], [`parentOf`, Z_1, Y_1]]
7	`grandparentOf`(X_2, Y_2) :– `grandfatherOf`(X_2, Y_2).	[[`grandparentOf`, X_2, Y_2], [`grandfatherOf`, X_2, Y_2]]
$\mathcal{P}$	{`fatherOf`, `parentOf`, `grandpaOf`, `grandfatherOf`, `grandparentOf`}	
$\mathcal{C}$	{ABE, HOMER, LISA, BART, MAGGIE}	
$\mathcal{V}$	$\{X_1, Y_1, Z_1, X_2, Y_2\}$	

Table 2.1: Example knowledge base using Prolog syntax (left) and as list representation as used in the backward chaining algorithm (right).

2.2 Function-free First-order Logic

We now turn to a brief introduction of function-free first-order logic to the extent it is used in subsequent chapters. This section follows the syntax of Prolog and Datalog [Gallaire and Minker, 1978], and is based on Lloyd [1987], Nilsson and Maluszynski [1990], and Dzeroski [2007].

2.2.1 Syntax

We start by defining an *atom* as a *predicate*[3] symbol and a list of terms. We will use lowercase names to refer to predicate and constant symbols (*e.g.* `fatherOf` and BART), and uppercase names for variables (*e.g.* X, Y, Z). In Prolog, one also considers function terms and defines constants as function terms with zero arity. However, in this thesis we will work only with function-free first-order logic rules, the subset of logic that Datalog supports. Hence, for us a *term* can be a constant or a variable. For instance, `grandfatherOf`(Q, BART) is an atom with the predicate `grandfatherOf`, and two terms, the variable Q and the constant BART, respectively. We define the *arity* of a predicate to be the number of terms it takes as arguments. Thus, `grandfatherOf` is a binary predicate. A *literal* is defined as a negated or non-negated atom. A *ground literal* is a literal with no variables (see rules 1 to 5 in Table 2.1). Furthermore, we consider first-order logic *rules*[4] of the form H :– $\mathbb{B}$, where the body $\mathbb{B}$ (also called condition or premise) is a possibly empty conjunction of atoms represented as a list, and the head H (also

[3]We will use predicate and relation synonymously throughout this thesis.
[4]We will use rule, clause and formula synonymously.

called conclusion, consequent or hypothesis) is an atom. Examples are rules 6 and 7 in Table 2.1. Such rules with only one atom as the head are called *definite rules*. In this thesis we only consider definite rules. Variables are universally quantified (*e.g.* $\forall \mathrm{X}_1, \mathrm{Y}_1, \mathrm{Z}_1$ in rule 6). A rule is a ground rule if all its literals are ground. We call a ground rule with an empty body a *fact*, hence the rules 1 to 5 in Table 2.1 are facts.[5] We define $\mathcal{S} = \mathcal{C} \cup \mathcal{P} \cup \mathcal{V}$ to be the set of symbols, containing constant symbols $\mathcal{C}$, predicate symbols $\mathcal{P}$, and variable symbols $\mathcal{V}$. We call a set of definite rules like the one in Table 2.1 a *knowledge base* or *logic program*. A *substitution* $\Psi = \{\mathrm{X}_1/t_1, \ldots, \mathrm{X}_N/t_N\}$ is an assignment of variable symbols X_i to terms t_i, and applying a substitution to an atom replaces all occurrences of variables X_i by their respective term t_i.

What we have defined so far is the syntax of logic used in this thesis. To assign meaning to this language (semantics), we need to be able to derive the truth value for facts. We focus on proof theory, *i.e.*, deriving the truth of a fact from other facts and rules in a KB.[6] In the next subsection we explain backward chaining, an algorithm for deductive reasoning. It is used to derive atoms from other atoms by applying rules.

2.2.2 Deduction with Backward Chaining

Representing knowledge (facts and rules) in symbolic form has the appeal that one can use automated deduction systems to infer new facts. For instance, given the logic program in Table 2.1, we can automatically deduce that $\texttt{grandfatherOf}(\textsc{abe}, \textsc{lisa})$ is a true fact by applying rule 6 using facts 1 and 2.

Backward chaining is a common method for automated theorem proving, and we refer the reader to Russell and Norvig [2010], Gelder [1987], Gallaire and Minker [1978] for details and to Fig. 2.4 for an excerpt of the pseudocode in style of a functional programming language. Particularly, we are making use of pattern matching to check for properties of arguments passed to a module. Note that "_" matches every argument and that the order matters, *i.e.*, if arguments match a line, subsequent lines are not evaluated. We denote sets by Euler script letters (*e.g.* $\mathcal{E}$), lists by small capital letters (*e.g.* E), lists of lists by blackboard bold letters (*e.g.* $\mathbb{E}$) and we use : to refer to prepending an element to a list (*e.g.* e : E or E : $\mathbb{E}$). While an atom is a list of a predicate symbol and terms, a rule can be seen as a list of atoms and thus a list of lists where the head of the list is the rule head.[7]

[5] We sometimes only call a rule a rule if it has a non-empty body. This will be clear from the context.

[6] See Dzeroski [2007] for other methods for semantics.

[7] For example, $[[\texttt{grandfatherOf}, \mathrm{X}, \mathrm{Y}], [\texttt{fatherOf}, \mathrm{X}, \mathrm{Z}], [\texttt{parentOf}, \mathrm{Z}, \mathrm{Y}]]$.

```
1. $\text{or}(\text{G}, S) = [S' \mid S' \in \text{and}(\mathbb{B}, \text{unify}(\text{H}, \text{G}, S)) \text{ for H :– } \mathbb{B} \in \mathfrak{K}]$

2. $\text{and}(\_, \texttt{FAIL}) = \texttt{FAIL}$
3. $\text{and}([\,], S) = S$
4. $\text{and}(\text{G} : \mathbb{G}, S) = [S'' \mid S'' \in \text{and}(\mathbb{G}, S') \text{ for } S' \in \text{or}(\text{substitute}(\text{G}, S), S)]$

5. $\text{unify}(\_, \_, \texttt{FAIL}) = \texttt{FAIL}$
6. $\text{unify}([\,], [\,], S) = S$
7. $\text{unify}([\,], \_, \_) = \texttt{FAIL}$
8. $\text{unify}(\_, [\,], \_) = \texttt{FAIL}$
9. $\text{unify}(h : \text{H}, g : \text{G}, S) = \text{unify}\left(\text{H}, \text{G}, \left\{\begin{array}{ll} S \cup \{h/g\} & \text{if } h \in \mathcal{V} \\ S \cup \{g/h\} & \text{if } g \in \mathcal{V}, h \notin \mathcal{V} \\ S & \text{if } g = h \\ \texttt{FAIL} & \text{otherwise} \end{array}\right\}\right)$

10. $\text{substitute}([\,], \_) = [\,]$
11. $\text{substitute}(g : \text{G}, S) = \left\{\begin{array}{ll} x & \text{if } g/x \in S \\ g & \text{otherwise} \end{array}\right\} : \text{substitute}(\text{G}, S)$
```

Figure 2.4: Simplified pseudocode for symbolic backward chaining (cycle detection omitted for brevity, see Russell and Norvig [2010], Gelder [1987], Gallaire and Minker [1978] for details).

Given a goal such as $\texttt{grandparentOf}(\text{Q}_1, \text{Q}_2)$, backward chaining finds substitutions of free variables with constants in facts in a KB (*e.g.* $\{\text{Q}_1/\text{ABRAHAM}, \text{Q}_2/\text{BART}\}$). This is achieved by recursively iterating through rules that translate a goal into subgoals which we attempt to prove, thereby exploring possible proofs. For example, the KB could contain the following rule that can be applied to find answers for the above goal:

$$\texttt{grandfatherOf}(\text{X}, \text{Y}) \text{ :– } \texttt{fatherOf}(\text{X}, \text{Z}), \texttt{parentOf}(\text{Z}, \text{Y}).$$

The proof exploration in backward-chaining is divided into two functions called <u>or</u> and <u>and</u> that perform a depth-first search through the space of possible proofs. The function <u>or</u> (line 1) attempts to prove a goal by unifying it with the head of every rule in a KB, yielding intermediate substitutions. Unification (lines 5-9) iterates through pairs of symbols in the two lists corresponding to the atoms that we want to unify and updates the substitution set if one of the two symbols is a variable. It

Rule	Remaining Goals	S
	[[grandparentOf, Q_1, Q_2]]	{ }
7	[[grandfatherOf, Q_1, Q_2]]	$\{X_2/Q_1, Y_2/Q_2\}$
6	[[fatherOf, Q_1, Z_1], [parentOf, Z_1, Q_2]]	$\{X_2/Q_1, Y_2/Q_2, X_1/Q_1, Y_1/Q_2\}$
1	[[parentOf, HOMER, Q_2]]	$\{X_2/Q_1, Y_2/Q_2, X_1/Q_1, Y_1/Q_2, Q_1/\text{ABE}, Z_1/\text{HOMER}\}$
2	[ ]	$\{X_2/Q_1, Y_2/Q_2, X_1/Q_1, Y_1/Q_2, Q_1/\text{ABE}, Z_1/\text{HOMER}, Q_2/\text{LISA}\}$

Table 2.2: Example proof using backward chaining.

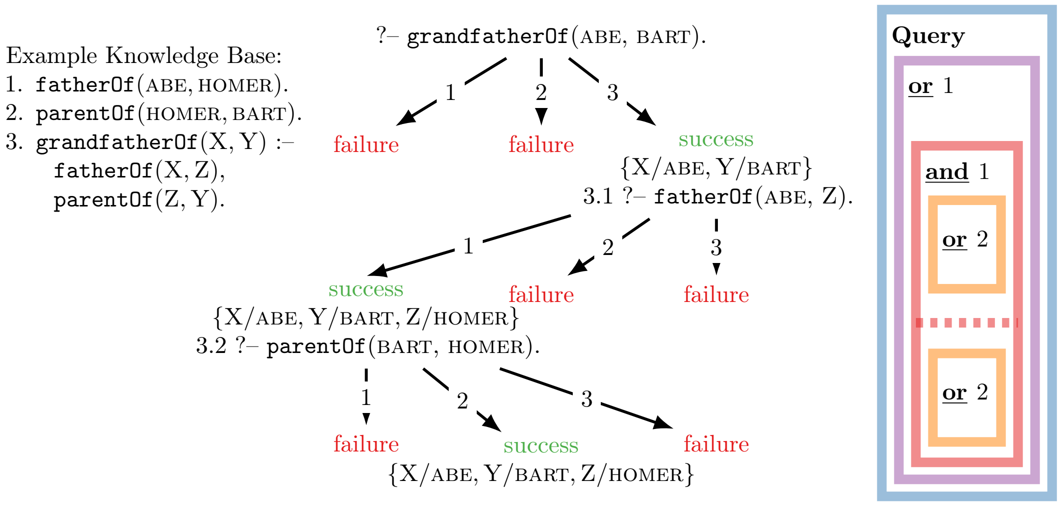


Figure 2.5: Full proof tree for a small knowledge base.

returns a failure if two non-variable symbols are not identical or the two atoms have different arity. For rules where unification with the head of the rule succeeds, the body and substitution are passed to and (lines 2-4), which then attempts to prove every atom in the body sequentially by first applying substitutions and subsequently calling or. This is repeated recursively until unification fails, atoms are proven by unification via grounding with facts in the KB, or a certain proof-depth is exceeded. Table 2.2 shows a proof for the query grandparentOf(Q_1, Q_2) given the KB in Table 2.1 using Fig. 2.4. The method substitute (lines 10-11) replaces variables in an atom by the corresponding symbol if there exists a substitution for that variable in the substitution list. Figure 2.5 shows the full proof tree for a small knowledge base and the query ?– grandfatherOf(ABE, BART).[8] The numbers on the arrows correspond to the application of the respective rules. We visualize the recursive calls to or and and together with the proof depth on the right side. Note how most proofs can be aborted early due to unification failure.

Though first-order logic can be used for complex multi-hop reasoning, a drawback is that for such symbolic inference there is no generalization beyond what we explicitly specify in the facts and rules. For instance, given a large KB we would

[8] We denote queries by the prefix "?–".

like to learn automatically that in many cases where we observe `fatherOf` we also observe `parentOf`. This can be approached by statistical relational learning, inductive logic programming, and particularly neural networks for KB completion which we discuss in the remainder of this chapter.

2.2.3 Inductive Logic Programming

While backward chaining is used for *deduction*, *i.e.*, for inferring facts given rules and other facts in a KB, Inductive Logic Programming (ILP) [Muggleton, 1991] combines logic programming with inductive learning to learn logic programs from training data. This training data can include facts, but also a provided logic program that the ILP system is supposed to extend. Specifically, given facts and rules, the task of an ILP system is to find further regularities and form hypotheses for unseen facts [Dzeroski, 2007]. Crucially, these hypothesis are again formulated using first-order logic.

There are different variants of ILP learning tasks [Raedt, 1997] and we focus on *learning from entailment* [Muggleton, 1991]. That is, given examples of positive facts and negative facts, the ILP system is supposed to find rules such that positive facts can be deduced, but negative facts cannot. There are many variants of ILP systems, and we refer the reader to Muggleton and Raedt [1994] and Dzeroski [2007] for an overview. One of the most prominent systems is the First Order Inductive Learner (FOIL) [Quinlan, 1990], which is a greedy algorithm that induces one rule at a time by constructing a body that satisfies the maximum number of positive facts and the minimum number of negative facts.

In Chapter 5, we will construct neural networks for proving facts in a KB and introduce a method for inducing logic programs using gradient descent while learning vector representations of symbols.

2.3 Automated Knowledge Base Completion

Automated KB completion is the task of inferring facts from information contained in a KB and other resources such as text. This is an important task as real-world KBs are usually incomplete. For instance, the `placeOfBirth` predicate is missing for 71% of people in Freebase [Dong et al., 2014]. Prominent recent approaches to automated KB completion learn vector representations of symbols via neural link prediction models. The appeal of learning such *subsymbolic* representations lies in their ability to capture similarity and even implicature directly in a vector space. Compared to ILP systems, neural link prediction models do not rely on a combinatorial search over the space of logic programs, but instead learn a local scoring function based on subsymbolic representations using continuous optimization. However, this comes at the cost of uninterpretable models and no straightforward ways of incorporating logical background knowledge – drawbacks that we seek to address in this thesis. Another benefit of neural link prediction models over ILP is that inferring whether a fact is true or not often amounts to efficient algebraic operations (feed-forward passes in shallow neural networks), which makes test-time inference very scalable. In addition, representations of symbols can be compositional. For instance, we can compose a representation of a natural language predicate from a sequence of word representations [Verga et al., 2016a].

In recent years, many models for automated KB completion have been proposed. In the next sections, we discuss prominent approaches. On a high level, these methods can be categorized into (i) neural link prediction models which define a local scoring function for the truth of a fact based on estimated symbol representations, and (ii) models that use paths between two entities in a KB for predicting new relations between them.

2.3.1 Matrix Factorization

In this section, we describe the matrix factorization relation extraction model by Riedel et al. [2013], which is an instance of a simple neural link prediction model. We discuss this model in detail, as it the basis on which we develop rule injection methods in Chapter 3 and 4.

Assume a set of observed entity pair symbols $\mathcal{C}$ and a set of predicate symbols $\mathcal{P}$, which can either represent structured binary relations from Freebase, a large collaborative knowledge base [Bollacker et al., 2008], or unstructured Open Information Extraction (OpenIE) [Etzioni et al., 2008] textual surface patterns collected from news articles. Examples for structured and unstructured relations are `company/founders` and `#2-co-founder-of-#1`, respectively. Here, `#2-co-founder-of-#1` is a textual pattern where `#1` and `#2` are placeholders for entities. For instance, the relationship between Elon Musk and Tesla in the sentence "Elon Musk, the co-founder of Tesla and the CEO of SpaceX, cites The Foundation Trilogy by Isaac Asimov as a major influence on his thinking"[9] could be expressed by the ground atom `#2-co-founder-of-#1`(TESLA, ELON MUSK). In this example, ELON MUSK appeared first in the textual pattern, but `#2` indicates that this constant will be used as the second argument in the predicate corresponding to the pattern. That way we can later introduce a rule

[9]`http://www.dailymail.co.uk/news/article-4045816`

$\forall \mathrm{X}, \mathrm{Y}$: `company/founders`(X, Y) :– `#2-co-founder-of-#1`(X, Y) without changing the order of the variables in the body of the rule.

Let $\mathcal{O} = \{r_s(e_i, e_j))\}$ be the set of observed ground atoms. Model F by Riedel et al. [2013] maps all symbols in a knowledge base to subsymbolic representations, *i.e.*, it learns a dense k-dimensional vector representation for every relation and entity pair. Thus, a training fact $r_s(e_i, e_j)$ is represented by two vectors, $\boldsymbol{v}_s \in \mathbb{R}^k$ and $\boldsymbol{v}_{ij} \in \mathbb{R}^k$, respectively. We will refer to these as *embeddings*, *subsymbolic representations*, *vector representations*, *neural representations*, or simply (symbol) *representations* when it is clear from the context. The truth estimate of a fact is modeled via the sigmoid of the dot product of the two symbol representations:

$$p_{sij} = \sigma(\boldsymbol{v}_s^\top \boldsymbol{v}_{ij}). \tag{2.6}$$

In fact, this expression corresponds to the computation graph shown in Fig. 2.1a with $\boldsymbol{x} = \boldsymbol{v}_s$ and $\boldsymbol{y} = \boldsymbol{v}_{ij}$. The score p_{sij} is measuring the compatibility between the relation and entity pair representation and can be interpreted as the probability of the fact being true conditioned on parameters of the model. We would like to train symbol representations such that true ground atoms get a score close to one and false ground atoms a score close to zero. This results in a low-rank matrix factorization corresponding to Generalized Principal Component Analysis (GPCA) [Collins et al., 2001]

$$\boldsymbol{K} \approx \sigma(\boldsymbol{P}\boldsymbol{C}^\top) \qquad \in \mathbb{R}^{|\mathcal{P}|\times|\mathcal{C}|} \tag{2.7}$$

where $\boldsymbol{P} \in \mathbb{R}^{|\mathcal{P}|\times k}$ and $\boldsymbol{C} \in \mathbb{R}^{|\mathcal{C}|\times k}$ are matrices of all relation and entity pair representations, respectively. This factorization is depicted in Fig. 2.6 where known facts are shown in green and the task is to complete this matrix for cells with a question mark. Equation 2.7 leads to generalization with respect to unseen facts as every relation and entity pair is represented in a low-dimensional space, and this information bottleneck will lead to similar entity pairs being represented close in distance in the vector space (likewise for similar predicates).

The distributional hypothesis states that "one shall know a word by the company it keeps" [Firth, 1957]. It has been used for learning word meaning from large collections of text [Lowe and McDonald, 2000, Padó and Lapata, 2007]. Applied to automated KB completion, one could say that the meaning of a relation or entity pair can be estimated from the entity pairs and relations that respectively appear together

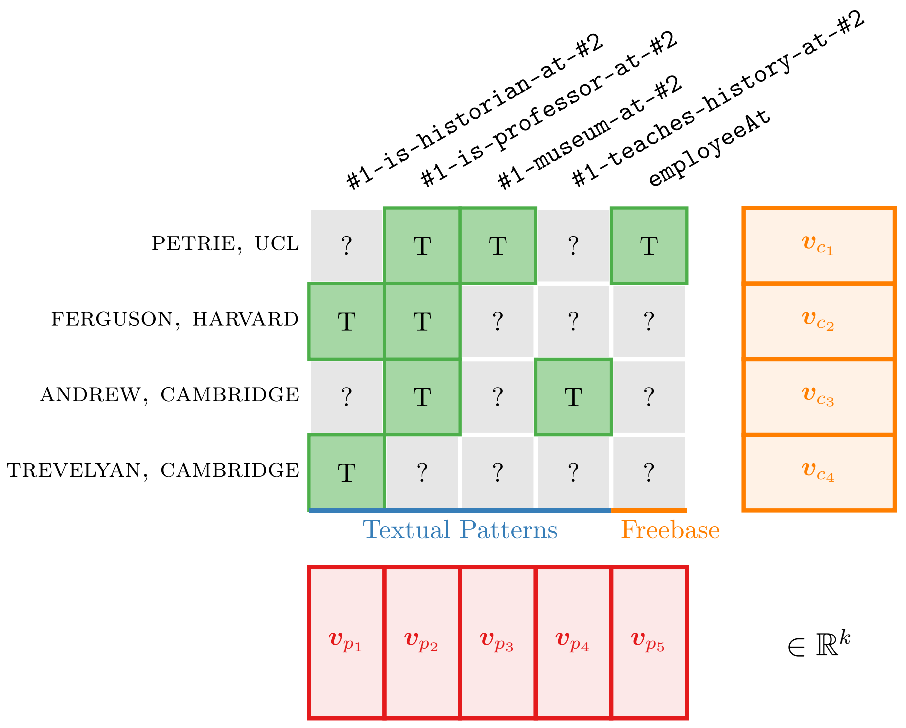


Figure 2.6: Knowledge base inference as matrix completion with true training facts (green), unobserved facts (question mark), relation representations (red), and entity pair representations (orange).

in facts in a KB. That is, if for many entity pairs we observe that both `fatherOf` and `dadOf` is true, we can assume that both relations are similar.

2.3.1.1 Bayesian Personalized Ranking

A common problem in automated KB completion is that we do not observe any negative facts. In the recommendation literature, this problem is called implicit feedback [Rendle et al., 2009]. Applied to KB completion, we would like to infer (or recommend) for a target relation some (unobserved) facts that we do not know from facts that we do know. Facts that we do not know can be unknown either because they are not true or because they are true but missing in the KB.

One method to address this issue is to formulate the problem in terms of a ranking loss, and sample unobserved facts as negative facts during training. Given a known fact $r_s(e_i, e_j) \in \mathcal{O}$, Bayesian Personalized Ranking (BPR) [Rendle et al., 2009] samples another entity pair $(e_m, e_n) \in \mathcal{C}$ such that $r_s(e_m, e_n) \notin \mathcal{O}$ and adds the soft-constraint

$$\boldsymbol{v}_s^\top \boldsymbol{v}_{ij} \geq \boldsymbol{v}_s^\top \boldsymbol{v}_{mn}. \tag{2.8}$$

This follows a relaxation of the local Closed World Assumption (CWA) [Galárraga et al., 2013, Dong et al., 2014]. In the CWA, the knowledge about true facts

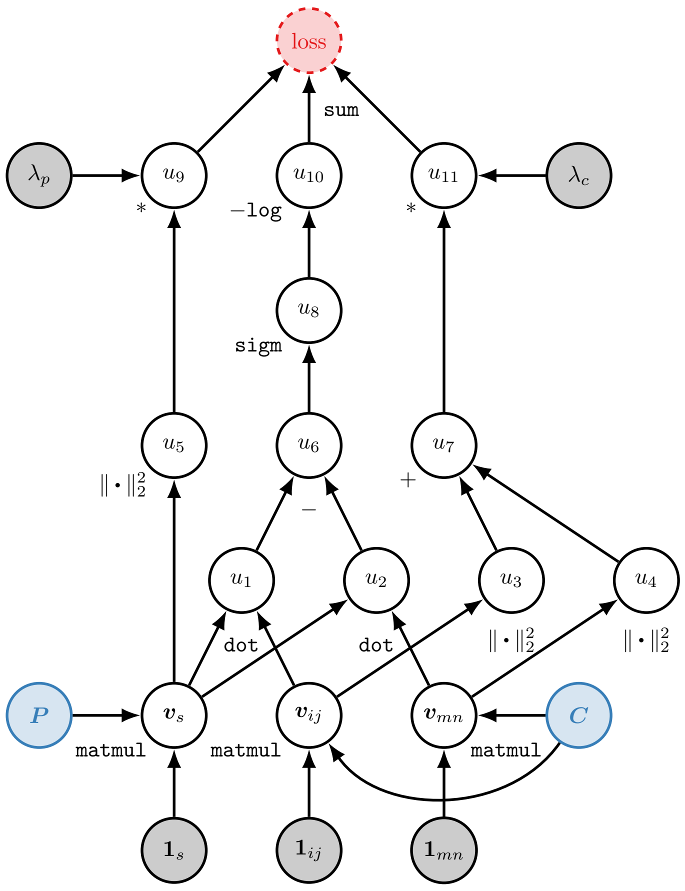


Figure 2.7: A complete computation graph of a single training example for Bayesian personalized ranking with ℓ_2-regularization.

for relation r_s is assumed to be complete and any sampled unobserved fact can consequently be assumed to be negative. In BPR, the assumption is that unseen facts are not necessarily false but their probability of being true should be less than for known facts. Thus, sampled unobserved facts should have a lower score than known facts. Instead of working with a fixed set of samples, we resample negative facts for every known fact in every epoch, where an epoch is a full iteration through all known facts in a KB. We denote a sample from the set of constant pairs as $(e_m, e_n) \sim \mathcal{C}$.[10] This leads to the overall approximate loss

$$\mathfrak{L} = \sum_{\substack{r_s(e_i,e_j) \in \mathcal{O}, \\ (e_m,e_n) \sim \mathcal{C}, \\ r_s(e_m,e_n) \notin \mathcal{O}}} -w_s \log \sigma(\boldsymbol{v}_s^\top \boldsymbol{v}_{ij} - \boldsymbol{v}_s^\top \boldsymbol{v}_{mn}) + \lambda_p \|\boldsymbol{v}_s\|_2^2 + \lambda_c(\|\boldsymbol{v}_{ij}\|_2^2 + \|\boldsymbol{v}_{mn}\|_2^2) \tag{2.9}$$

[10]Note that $\mathcal{C}$ is the set of constant-pairs.

where λ_p and λ_c are the strength of the ℓ_2 regularization of the relation and entity pair representations, respectively. Furthermore, w_s is a relation-dependent implicit weight accounting for how often (e_m, e_n) is sampled during training time. As we resample an unobserved fact every time we visit an observed fact during training, unobserved facts for relations with many observed facts are more likely to be sampled as negative facts than for relations with few observed facts. A complete computation graph of a single training example for matrix factorization using BPR with ℓ_2 regularization is shown in Fig. 2.7.

2.3.2 Other Neural Link Prediction Models

An example for a simple neural link prediction model is the matrix factorization approach from the previous section. Alternative methods define the score p_{sij} in Eq. 2.6 in different ways, use different loss functions, and parameterize relation and entity (or entity pair) representations differently. For instance, Bordes et al. [2011] train two projection matrices per relation, one for the left and one for the right-hand argument position, respectively. Subsequently, the score of a fact is defined as the ℓ_1 norm of the difference of the two projected entity argument embeddings

$$p_{sij} = \|\boldsymbol{M}_s^{\text{left}}\boldsymbol{v}_i - \boldsymbol{M}_s^{\text{right}}\boldsymbol{v}_j\|_1. \tag{2.10}$$

Note that compared to the matrix factorization model by Riedel et al. [2013] which embeds entity-pairs, here we learn individual entity embeddings. Similarly, TransE [Bordes et al., 2013] models the score as the ℓ_1 or ℓ_2 norm of the difference between the right entity embedding and a translation of the left entity embedding via the relation embedding

$$p_{sij} = \|\boldsymbol{v}_i + \boldsymbol{v}_s - \boldsymbol{v}_j\|_1 \tag{2.11}$$

where $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ are constrained to be unit-length. The computation graph for this model is shown in Fig. 2.2 in Section 2.1.2.

RESCAL [Nickel et al., 2012] represents relations as matrices and defines the score of a fact as

$$p_{sij} = \boldsymbol{v}_i^\top \boldsymbol{M}_s \boldsymbol{v}_j. \tag{2.12}$$

In contrast to the other models mentioned in this section, RESCAL is not optimized with SGD but using alternating least squares [Nickel et al., 2011]. TRESCAL [Chang et al., 2014] extends RESCAL with entity type constraints for Freebase relations.

Neural Tensor Networks [Socher et al., 2013] add a relation-dependent compatibility score to RESCAL

$$p_{sij} = \boldsymbol{v}_i^\top \boldsymbol{M}_s \boldsymbol{v}_j + \boldsymbol{v}_i^\top \boldsymbol{v}_s^{\text{left}} + \boldsymbol{v}_j^\top \boldsymbol{v}_s^{\text{right}} \tag{2.13}$$

and are optimized with L-BFGS [Byrd et al., 1995]. DistMult by Yang et al. [2015] is modeling the score as the trilinear dot product

$$p_{sij} = \boldsymbol{v}_s^\top (\boldsymbol{v}_i \odot \boldsymbol{v}_j) = \sum_k \boldsymbol{v}_{sk} \boldsymbol{v}_{ik} \boldsymbol{v}_{jk} \tag{2.14}$$

where $\odot$ is the element-wise multiplication. This model is a special case of RESCAL where $\boldsymbol{M}_s$ is constrained to be diagonal. ComplEx by Trouillon et al. [2016] uses complex vectors $\boldsymbol{v}_s, \boldsymbol{v}_i, \boldsymbol{v}_j \in \mathbb{C}^k$ for representing relations and entities. Let $\text{real}(\boldsymbol{v})$ denote the real part and $\text{imag}(\boldsymbol{v})$ the imaginary part of a complex vector $\boldsymbol{v}$. The scoring function defined by ComplEx is

$$\begin{aligned} p_{sij} = {} & \text{real}(\boldsymbol{v}_s)^\top (\text{real}(\boldsymbol{v}_i) \odot \text{real}(\boldsymbol{v}_j)) \\ & + \text{real}(\boldsymbol{v}_s)^\top (\text{imag}(\boldsymbol{v}_i) \odot \text{imag}(\boldsymbol{v}_j)) \\ & + \text{imag}(\boldsymbol{v}_s)^\top (\text{real}(\boldsymbol{v}_i) \odot \text{imag}(\boldsymbol{v}_j)) \\ & - \text{imag}(\boldsymbol{v}_s)^\top (\text{imag}(\boldsymbol{v}_i) \odot \text{real}(\boldsymbol{v}_j)). \end{aligned} \tag{2.15}$$

The benefit of ComplEx over RESCAL and DistMult is that by using complex vectors it can capture symmetric as well as asymmetric relations.

Building upon Riedel et al. [2013], Verga et al. [2016a] developed a column-less factorization approach by encoding surface form patterns using Long Short-Term Memories (LSTMs) [Hochreiter and Schmidhuber, 1997] instead of learning a non-compositional representation. Similarly, Toutanova et al. [2015] uses Convolutional Neural Networks (CNNs) to encode surface form patterns. In a follow-up study, Verga et al. [2016b] propose a row-less method where entity pair representations are not learned but instead computed from observed relations, thereby generalizing to new entity pairs at test time.

2.3.3 Path-based Models

While all methods presented in the previous section model the truth of a fact as a local scoring function of the representations of the relation and entities (or entity pairs), path-based models score facts based either on random walks over the KBs (path ranking) or by encoding entire paths in a vector space (path encoding).

2.3.3.1 Path Ranking

The Path Ranking Algorithm (PRA) [Lao and Cohen, 2010, Lao et al., 2011] learns to predict a relation between two entities based on logistic regression over features collected from random walks between these entities in the KB up to some predefined length. Lao et al. [2012] extend PRA inference to OpenIE surface patterns in addition to structured relations contained in KBs.

A related approach to PRA is Programming with Personalized PageRank (ProPPR) [Wang et al., 2013], which is a first-order probabilistic logic programming language. It uses Prolog's Selective Linear Definite clause resolution (SLD) [Kowalski and Kuehner, 1971], a depth-first search strategy for theorem proving, to construct a graph of proofs. Instead of returning deterministic proofs for a given query, ProPPR defines a stochastic process on the graph of proofs using PageRank [Page et al., 1999]. Furthermore, in ProPPR one can use features on the head of rules whose weights are learned from data to guide stochastic proofs. Experiments with ProPPR were conducted on comparably small KBs and contrary to neural link prediction models and the extensions to PRA below, it has not yet been scaled to large real-world KBs [Gardner et al., 2014].

A shortcoming of PRA and ProPPR is that they are operating on symbols instead of vector representations of symbols. This limits generalization as it results in an explosion in the number of paths to consider when increasing the path length. To overcome this limitation, Gardner et al. [2013] extend PRA to include vector representations of verbs. These verb representations are obtained from pre-training via PCA on a matrix of co-occurrences of verbs and subject-object tuples collected from a large dependency-parsing corpus. Subsequently, these representations are used for clustering relations, thus avoiding an explosion of path features in prior PRA work while improving generalization. Gardner et al. [2014] take this approach further by introducing vector space similarity into random walk inference, thus dealing with paths containing unseen surface forms by measuring the similarity to surface forms seen during training, and following relations proportionally to this similarity.

2.3.3.2 Path Encoding

While Gardner et al. introduced vector representations into PRA, these representations are not trained end-to-end from task data but instead pretrained on an external corpus. This means that relation representations cannot be adapted during training on a KB.

Neelakantan et al. [2015] propose RNNs for learning embeddings of entire paths. The input to these RNNs are trainable relation representations. Given a known relation between two entities and a path connecting the two entities in the KB, an RNN for the target relation is trained to output an encoding of the path such that the dot product of that encoding and the relation representation is maximal.

Das et al. [2017] note three limitations of the work by Neelakantan et al. [2015]. First, there is no parameter sharing of RNNs that encode different paths for different target relations. Second, there is no aggregation of information from multiple path encodings. Lastly, there is no use of entity information along the path as only relation representations are fed to the RNN. Das et al. address the first issue by using a single RNN whose parameters are shared across all paths. To address the second issue, Das et al. train an aggregation function over the encodings of multiple paths connecting two entities. Finally, to obtain entity representations that are fed into the RNN alongside relation representations they sum learned vector representations of the entity's annotated Freebase types.

Chapter 3

Regularizing Representations by First-order Logic Rules

In this chapter, we introduce a paradigm for combining neural link prediction models for automated Knowledge Base (KB) completion (Section 2.3.2) with background knowledge in the form of first-order logic rules. We investigate simple baselines that enforce rules through symbolic inference before and after matrix factorization. Our main contribution is a novel joint model that learns vector representations of relations and entity pairs using both distant supervision and first-order logic rules, such that these rules are captured directly in the vector space of symbol representations. To this end, we map symbolic rules to differentiable computation graphs representing real-valued losses that can be added to the training objective of existing neural link prediction models. At test time, inference is still efficient as only a local scoring function over symbol representations is used and no logical inference is needed. We present an empirical evaluation where we incorporate automatically mined rules into a matrix factorization neural link prediction model. Our experiments demonstrate the benefits of incorporating logical knowledge for Freebase relation extraction. Specifically, we find that joint factorization of distant and logic supervision is efficient, accurate, and robust to noise (Section 3.5). By incorporating logical rules, we were able to train relation extractors for which no or only few training facts are observed.

3.1 Matrix Factorization Embeds Ground Literals

In Section 2.3.1, we introduced matrix factorization as a method for learning representations of predicates and constant pairs for automated KB completion. In this section, we elaborate on how matrix factorization indeed embeds ground atoms in a vector space and lay out the foundation for developing a method that embeds first-order logic rules.

Let $\text{F} \in \mathcal{F}$ denote a rule in a KB. For instance, F could be a ground rule without a body (*i.e.* a fact) like $\texttt{parentOf}(\text{HOMER}, \text{BART})$. Furthermore, let $[\![\text{F}]\!]$ denote the probability of this rule being true conditioned on the parameters of the model. For now, we restrict F to ground atoms and discuss ground literals, propositional rules, and first-order rules later. With slight abuse of notation, let $[\![\bullet]\!]$ also denote the mapping of predicate or constant symbols (or a pair of constant symbols) to their subsymbolic representation as assigned by the model. Note that this mapping depends on the neural link prediction model. For matrix factorization, $[\![\bullet]\!]$ is a function $\mathcal{S} \to \mathbb{R}^k$ from symbols (constant pairs and predicates) to k-dimensional dense vector representations. For RESCAL (see Section 2.3.2), $[\![\bullet]\!]$ maps constants to $\mathbb{R}^k$ and predicates to $\mathbb{R}^{k \times k}$. Using this notation, matrix factorization decomposes the probability of a fact $r_s(e_i, e_j)$ as

$$[\![r_s(e_i, e_j)]\!] = \sigma([\![r_s]\!]^\top [\![e_i, e_j]\!]) = \sigma(\boldsymbol{v}_s^\top \boldsymbol{v}_{ij}). \tag{3.1}$$

Training Objective Riedel et al. [2013] used Bayesian Personalized Ranking (BPR) [Rendle et al., 2009] as training objective, *i.e.*, they encouraged the score of known true facts to be higher than unknown facts (Section 2.3.1.1). However, as we will later model the probability of a rule from the probability of ground atoms scored by a neural link prediction model, we need to ensure that all scores are in the interval $[0, 1]$. Instead of BPR, we thus use the negative log-likelihood loss to directly maximize the probability of all rules, *including ground atoms*, in a KB (we omit ℓ_2 regularization for brevity):

$$\mathfrak{L} = \sum_{\text{F} \in \mathcal{F}} -\log([\![\text{F}]\!]). \tag{3.2}$$

Therefore, instead of learning to rank facts, we optimize representations to assign a score close to 1.0 to rules (including facts). Our model can thus be seen as generalization of a neural link prediction model to rules beyond ground atoms.

For matrix factorization as neural link prediction model, we are embedding ground atoms in a vector space of predicate and constant pair representations. Next, we will extend this to ground literals and afterwards to propositional and then first-order logic rules.

Negation Let $\textsc{f} = \neg\,\textsc{g}$ be the negation of a ground atom $\textsc{g}$. We can model the probability of $\textsc{f}$ as follows:

$$[\![\textsc{f}]\!] = [\![\neg\,\textsc{g}]\!] = 1 - [\![\textsc{g}]\!]\,. \tag{3.3}$$

By using ground literals in the training objective in Eq. 3.2, we can say that matrix factorization is embedding ground literals via learning predicate and constant pair representations. In other words, given known ground literals (negated and non-negated facts), matrix factorization embeds symbols in a low-dimensional vector space such that a scoring function assigns a high probability to these ground literals. As symbols are embedded in a low-dimensional vector space, this method can generalize to unknown facts and predict a probability for these at test time by placing similar symbols close in distance in the embedding space.

Note that so far we have not gained anything over matrix factorization as explained in Section 2.3.1. Equations 3.2 and 3.3 are only introducing notation that will make it easier to embed more complex rules later. Now that we can embed negated and non-negated facts, we can ask the question whether we can also embed propositional and first-order logic rules.

3.2 Embedding Propositional Logic

We know from propositional logic that with the negation and conjunction operators we can model any other Boolean operator and propositional rule. In Eq. 3.3, we effectively turned a symbolic logical operation (negation) into a differentiable operation that can be used to learn subsymbolic representations for automated KB completion. If we can find such a differentiable operation for conjunction, then we could backpropagate through any propositional logical expression, and learn vector representations of symbols that encode given background knowledge in propositional logic.

Conjunction In Product Fuzzy Logic, conjunction is modeled using a Product t-Norm [Lukasiewicz, 1920]. Let $\textsc{f} = \textsc{a} \wedge \textsc{b}$ be the conjunction of two propositional expressions $\textsc{a}$ and $\textsc{b}$. The probability of $\textsc{h}$ is then defined as follows:

$$[\![\textsc{f}]\!] = [\![\textsc{a} \wedge \textsc{b}]\!] = [\![\textsc{a}]\!]\,[\![\textsc{b}]\!]\,. \tag{3.4}$$

In other words, we replaced conjunction, a symbolic logical operation, with multiplication, a differentiable operation. Note that alternatives for modeling conjunction exist. For instance, one could take the min of $[\![\textsc{a}]\!]$ and $[\![\textsc{b}]\!]$ (Gödel t-Norm [Gödel, 1932]).

Given the probability of ground atoms, we can use Product Fuzzy Logic to calculate the probability of the conjunction of these atoms. However, we will go a step further and assume that we know the ground truth probability of the conjunction of two atoms. We can then use the negative log-likelihood loss to measure the discrepancy between the predicted probability of the conjunction and the ground truth. Our contribution is backpropagating this discrepancy through the propositional rule and a neural link prediction model that scores ground atoms to calculate a gradient with respect to vector representations of symbols. Subsequently, we update these representations using gradient descent, thereby encoding the ground truth of a propositional rule directly in the vector representations of symbols. At test time, predicting a score for any unobserved ground atom $r_s(e_i, e_j)$ is done efficiently by calculating $[\![r_s(e_i, e_j)]\!]$.

Disjunction Let $\textsc{f} = \textsc{a} \vee \textsc{b}$ be the disjunction of two propositional expressions $\textsc{a}$ and $\textsc{b}$. Using De Morgan's law and Eqs. 3.3 and 3.4, we can model the probability of $\textsc{f}$ as follows:

$$\begin{aligned} [\![\textsc{f}]\!] &= [\![\textsc{a} \vee \textsc{b}]\!] \\ &= [\![\neg\,(\neg\,(\textsc{a} \vee \textsc{b}))]\!] \\ &= [\![\neg\,(\neg\,\textsc{a} \wedge \neg\,\textsc{b})]\!] \\ &= 1 - (1 - [\![\textsc{a}]\!])(1 - [\![\textsc{b}]\!]) \\ &= [\![\textsc{a}]\!] + [\![\textsc{b}]\!] - [\![\textsc{a}]\!]\,[\![\textsc{b}]\!]\,. \end{aligned} \tag{3.5}$$

Note that Eq. 3.3 not only holds for ground atoms, but any propositional logical expression. Furthermore, any propositional logical expression can be normalized to Conjunctive Normal Form. Thus, with Eqs. 3.3 to 3.5 we now have a way to construct a differentiable computation graph, and thus a real-valued loss term, for any symbolic expression in propositional logic.

Implication A particular class of logical expressions that we care about in practice are propositional implication rules of the form H :– $\mathbb{B}$, where the body $\mathbb{B}$ is a possibly

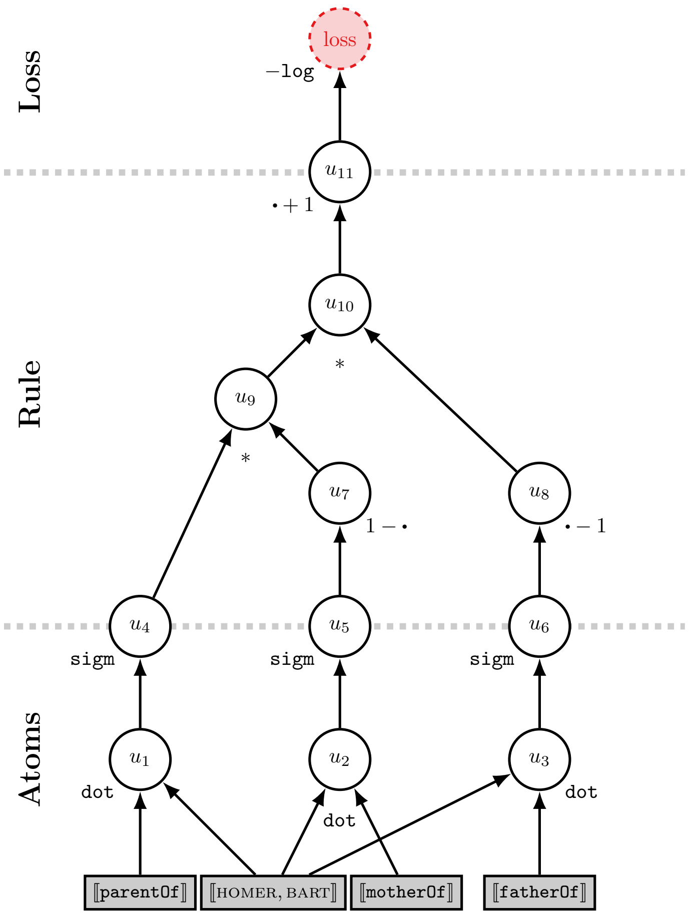


Figure 3.1: Computation graph for rule in Eq. 3.7 where · denotes a placeholder for the output of the connected node.

empty conjunction of atoms represented as a list, and the head H is an atom. Let $\text{F} = \text{H} :\!- \text{B}$. The probability of F is then modeled as:

$$\begin{aligned}
[\![\text{F}]\!] &= [\![\text{H} :\!- \text{B}]\!] \\
&= [\![\neg\, \text{B} \vee \text{H}]\!] \\
&= [\![\neg\, \text{B}]\!] + [\![\text{H}]\!] - [\![\neg\, \text{B}]\!]\,[\![\text{H}]\!] \\
&= 1 - [\![\text{B}]\!] + [\![\text{H}]\!] - (1 - [\![\text{B}]\!])\,[\![\text{H}]\!] \\
&= 1 - [\![\text{B}]\!] + [\![\text{H}]\!] - [\![\text{H}]\!] + [\![\text{B}]\!]\,[\![\text{H}]\!] \\
&= 1 - [\![\text{B}]\!] + [\![\text{B}]\!]\,[\![\text{H}]\!] \\
&= [\![\text{B}]\!]\,([\![\text{H}]\!] - 1) + 1. \qquad (3.6)
\end{aligned}$$

Say we want to ensure that

$$\begin{aligned}&\texttt{fatherOf}(\text{HOMER}, \text{BART}) :\!- \\ &\quad \texttt{parentOf}(\text{HOMER}, \text{BART}) \\ &\quad \neg\,\texttt{motherOf}(\text{HOMER}, \text{BART}).\end{aligned} \tag{3.7}$$

We now have a way to map this rule to a differentiable expression that we can use alongside facts in Eq. 3.2 and optimize the symbol representations using gradient descent as we did previously for matrix factorization. The computation graph that allows us to calculate the gradient of this rule with respect to symbol representations is shown in Fig. 3.1. While the structure of the bottom part (Atoms) of this computation graph is determined by the neural link prediction model, the middle part (Rule) is determined by the propositional rule. Note that we can use any neural link predictor (see Section 2.3.2) instead of matrix factorization for obtaining a probability of ground atoms. The only requirement is that ground atom scores need to lie in the interval $[0, 1]$. However, for models where this is not the case, we can always apply a transformation such as the sigmoid.

Independence Assumption Equation 3.4 underlies a strong assumption, namely that the probability of the arguments of the conjunction are conditionally independent given symbol embeddings. We already get a violation of this assumption for the simple case $[\![\text{F} \wedge \text{F}]\!]$ with $0 < [\![\text{F}]\!] < 1$, which results in $[\![\text{F} \wedge \text{F}]\!] = [\![\text{F}]\!]\,[\![\text{F}]\!] < [\![\text{F}]\!]$. However, for dependent arguments we get an approximation to the probability of the conjunction that can still be used for gradient updates of the symbol representations, and we demonstrate empirically that conjunction as modeled in Eq. 3.4 is useful for improving automated KB completion. In Chapter 4, we will present a way to avoid this independence assumption for implications.

3.3 Embedding First-order Logic via Grounding

Now that we can backpropagate through propositional Boolean expressions, we turn to embedding first-order logic rules in the vector space of symbol representations. Note that in this process, we do not chain rules, *i.e.*, we do not perform logical inference at training or test time. Instead, a provided rule is used to construct a loss term for optimizing vector representations of symbols. When training the model using gradient descent, we attempt to find a minimum of the global loss Eq. 3.2 such that the probability of all rules (including facts) is high. This minimum might be attained where the prediction of the neural link prediction model for scoring ground atoms not only agrees with a single but all rules, thereby predicting ground atoms as

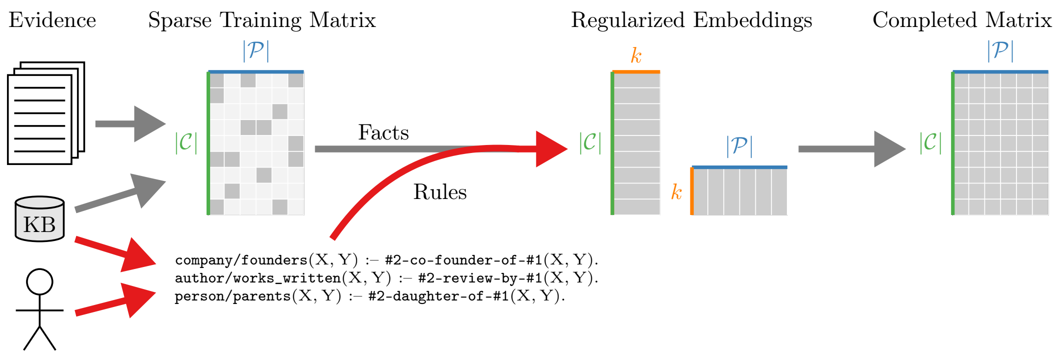

Evidence
Sparse Training Matrix
Regularized Embeddings
Completed Matrix
|𝒫|
|𝒞|
KB
Facts
Rules
k
company/founders(X, Y) :– #2-co-founder-of-#1(X, Y).
author/works_written(X, Y) :– #2-review-by-#1(X, Y).
person/parents(X, Y) :– #2-daughter-of-#1(X, Y).

Figure 3.2: Given a set of training ground atoms, matrix factorization learns k-dimensional predicate and constant pair representations. Here, we also consider additional first-order logic rules (red) and seek to learn symbol representations such that the predictions (completed matrix) comply with these given rules.

if we chained rules. A high-level overview of this process for matrix factorization as neural link prediction model of ground atoms is shown in Fig. 3.2.

Assuming a finite set of constants, we can ground any first-order rule by replacing free variables with constants in the domain. Let $\text{F}(\text{X}, \text{Y})$ denote a universally quantified rule with two free variables X and Y, then

$$[\![\forall \text{X}, \text{Y} : \text{F}(\text{X}, \text{Y})]\!] = \left[\!\!\left[\bigwedge_{(e_i,e_j)\in\mathcal{C}} \text{F}(e_i, e_j) \right]\!\!\right] \tag{3.8}$$

where $\mathcal{C}$ is the set of all entity pairs. With Eqs. 3.2 and 3.4 we obtain the following ground loss for F:

$$-\log\left(\left[\!\!\left[\bigwedge_{(e_i,e_j)\in\mathcal{C}} \text{F}(e_i, e_j) \right]\!\!\right]\right) = -\log\left(\prod_{(e_i,e_j)\in\mathcal{C}} [\![\text{F}(e_i, e_j)]\!] \right) = -\sum_{(e_i,e_j)\in\mathcal{C}} \log [\![\text{F}(e_i, e_j)]\!]. \tag{3.9}$$

3.3.1 Stochastic Grounding

For large domains of constants, Eq. 3.9 becomes very costly to optimize as it would result in many expressions that are added to the training objective of the underlying neural link prediction model. For rules over pairs of constants, we can reduce the number of terms drastically by only considering pairs of constants $\mathcal{C}^{\text{train}}$ that appeared together in training facts. Still, $\mathcal{C}^{\text{train}}$ might be a prohibitively large set of constant pairs. Thus, we resort to a heuristic similar to BPR for sampling constant pairs. Given a rule, we obtain a ground propositional rule for every constant pair for which at least one atom in the rule is a known training fact when substituting free variables with these constants. In addition, we sample as many constant pairs that appeared together in training facts but for which all atoms in the rule are unknown when substituting free variables with the sampled constant pairs.

3.3.2 Baseline

Background knowledge in form of first-order rules can be seen as *hints* that can be used to generate additional training data [Abu-Mostafa, 1990]. For *pre-factorization inference* we first perform symbolic logical inference on the training data using provided rules and add inferred facts as additional training data. For example, for a rule $r_t(\mathrm{X}, \mathrm{Y}) \coloneq r_s(\mathrm{X}, \mathrm{Y})$, we add an additional observed training facts $r_t(e_i, e_j)$ for any pair of constants (e_i, e_j) for which $r_s(e_i, e_j)$ is a true fact in the training data. This is repeated until no further facts can be inferred. Subsequently, we run matrix factorization on the extended set of observed facts. The intuition is that the additional training data generated by rules provide evidence of the logical dependencies between relations to the matrix factorization model, while at the same time allowing the factorization to generalize to unobserved facts and to deal with ambiguity and noise in the data. No further logical inference is performed during or after training of the factorization model as we expect that the learned embeddings encode the provided rules.

One drawback of pre-factorization inference is that the rules are enforced only on observed atoms, *i.e.*, first-order dependencies on *predicted* facts are ignored. In contrast, with the loss in Eq. 3.9 we add terms for the rule directly to the matrix factorization objective, thus jointly optimizing embeddings to reconstruct known facts, as well as to obey to provided first-order logical background knowledge. However, as we stochastically ground first-order rules, we have no guarantee that a given rule will indeed hold for all possible entity pairs at test time. While we next demonstrate that this approach is still useful for KB completion despite this limitation, in Chapter 4 we will introduce a method that overcomes this limitation for simple first-order implication rules.

3.4 Experiments

There are two orthogonal questions when evaluating the method above. First, does regularizing symbol embeddings by first-order logic rules indeed capture such logical knowledge in a vector space and can it be used to improve KB completion? Second, where can we obtain background knowledge in form of rules that is useful for a particular KB completion task? The latter is a well-studied problem [Hipp et al., 2000, Schoenmackers et al., 2010a, Völker and Niepert, 2011]. Thus, we focus the evaluation on the ability of various approaches to benefit from rules that we directly extract from the training data using a simple method.

Distant Supervision Evaluation We follow the procedure of Riedel et al. [2013] for evaluating knowledge base completion of Freebase [Bollacker et al., 2008] with textual data from the NYT corpus [Sandhaus, 2008]. The training matrix consists of $4\,111$ columns, representing 151 Freebase relations and $3\,960$ textual patterns, $41\,913$ rows (constant pairs) and $118\,781$ training facts of which $7\,293$ belong to Freebase relations. The constant pairs are divided into train and test, and we remove all Freebase facts for these test pairs from the training data. Our primary evaluation measure is (weighted) Mean Average Precision (MAP) [Manning et al., 2008, Riedel et al., 2013]. Let $\mathcal{R}$ be the set of test relations and let $\{f_{1j}, \ldots, f_{mj}\}$ be the set of test facts for relation $r_j \in \mathcal{R}$. Furthermore, let R_{kj} be the ranked list of facts for relation r_j scored by a model up until fact f_{kj} is reached. MAP is then defined as

$$\text{MAP}(\mathcal{R}) = \frac{1}{|\mathcal{R}|} \sum_{j=1}^{|\mathcal{R}|} \frac{1}{m_j} \sum_{k=1}^{m_j} \text{precision}(R_{kj}) \tag{3.10}$$

where precision calculates the fraction of correctly predicted test facts of all predicted facts. For weighted MAP, the average precision for every relation is weighted by the number of true facts for the respective relation [Riedel et al., 2013]. Note that the MAP metric operates only on the ranking of facts as predicted by the model and does not take the absolute predicted score into account.

Rule Extraction and Annotation We use a simple technique for extracting rules from a matrix factorization model based on Sanchez et al. [2015]. We first run matrix factorization over the complete training data to learn symbol representations. After training, we iterate over all pairs of relations (r_s, r_t) where r_t is a Freebase relation. For every relation pair we iterate over all training atoms $r_s(e_i, e_j)$, evaluate the score $[\![r_t(e_i, e_j) :\!- r_s(e_i, e_j)]\!]$ using Eq. 3.6, and calculate the average to arrive at a score as the proxy for the coverage of the rule. Finally, we rank all rules by their score and manually inspect and filter the top 100, which resulted in 36 annotated high-quality rules (see Table 3.1 for the top rules for five different Freebase target relations and Appendix A for the list of all annotated rules). Note that our rule extraction approach does not observe the relations for test constant pairs and that all extracted rules are simple first-order logic expressions. All models in our experiments have access to these rules, except for the matrix factorization baseline.

Methods Our proposed methods for injecting logic into symbol embeddings are *pre-factorization inference* (**Pre**; Section 3.3.2) which is a baseline method that performs

Rule	Score
`org/parent/child`(X, Y) :– `#2-unit-of-#1`(X, Y).	0.97
`location/containedby`(X, Y) :– `#2-city-of-#1`(X, Y).	0.97
`person/nationality`(X, Y) :– `#2-minister-#1`(X, Y).	0.97
`person/company`(X, Y) :– `#2-executive-#1`(X, Y).	0.96
`company/founders`(X, Y) :– `#2-co-founder-of-#1`(X, Y).	0.96

Table 3.1: Top rules for five different Freebase target relations. These implications were extracted from the matrix factorization model and manually annotated. The premises of these implications are shortest paths between entity arguments in dependency tree, but we present a simplified version to make these patterns more readable. See Appendix A for the list of all annotated rules.

regular matrix factorization after propagating the logic rules in a deterministic manner, and *joint optimization* (**Joint**; Section 3.3) which maximizes an objective that combines terms from facts and first-order logic rules. Additionally, we evaluate three baselines. The *matrix factorization* (**MF**; Section 2.3.1) model uses only ground atoms to learn relation and constant representations (*i.e.* it has no access to any rules). Furthermore, we consider pure *symbolic logical inference* (**Inf**). Since we restrict ourselves to a set of consistent, simple rules, this inference can be performed efficiently. Our final approach, *post-factorization inference* (**Post**), first runs matrix factorization and then performs logical inference on the known and predicted facts. Post-inference is computationally expensive since for all premises of rules we have to iterate over *all* rows (constant pairs) in the matrix to assess whether the premise is predicted to be true or not.

3.4.1 Training Details

Since we have no negative training facts, we follow Riedel et al. [2013] by sampling unobserved ground atoms that we assume to be false. For rules, we use stochastic grounding as described in Section 3.3.1. Thus, in addition to a loss over the score of training facts, we have a loss over sampled unobserved ground atoms that we assume to be negative, as well as loss terms for ground rules. In other words, we learn symbol embeddings by minimizing Eq. 3.2 where $\mathcal{F}$ includes known and unobserved atoms, as well as ground propositional rules sampled using stochastic grounding. In addition, we use ℓ_2-regularization on all symbol representations. For minimizing the training loss we use AdaGrad [Duchi et al., 2011].

Remember that at test time, predicting a score for any unobserved ground atom $r_s(e_i, e_j)$ is done efficiently by calculating $[\![r_s(e_i, e_j)]\!]$. Note that this does not involve any explicit logical inference. Instead, we expect the vector space of symbol embeddings to incorporate all given rules.

Hyperparameters Based on Riedel et al. [2013], we use $k = 100$ as the dimension for symbol representations in all models, $\lambda = 0.01$ as parameter for ℓ_2-regularization, and $\alpha = 0.1$ as the initial learning rate for AdaGrad, which we run for 200 epochs.

Runtime Each AdaGrad update is defined over a single cell of the matrix, and thus training data can be provided one ground atom at a time. For matrix factorization, each AdaGrad epoch touches all the observed ground atoms once per epoch, and as many sampled negative ground atoms. With provided rules, it additionally revisits all the observed ground atoms that appear as an atom in the rules (and as many sampled negative ground atoms), and thus more general rules will be more expensive. However, the updates on ground atoms are performed independently and thus not all the data needs to be stored in memory. All presented models take less than 15 minutes to train on a 2.8 GHz Intel Core i7 machine.

3.5 Results and Discussion

To asses the utility of injecting logic rules into symbol representations, we present a comparison on a variety of benchmarks. First, we study the scenario of learning extractors for relations for which we do not have any Freebase alignments, *i.e.*, the number of entity pairs that appear both in textual patterns and structured Freebase relations is zero. This measures how well the different models are able to generalize only from logic rules and textual patterns (Section 3.5.1). In Section 3.5.2, we then describe an experiment where the number of Freebase alignments is varied in order to assess the effect of combining distant supervision and background knowledge on the accuracy of predictions. Although the methods presented here target relations with insufficient alignments, we also provide a comparison on the complete distant supervision dataset in Section 3.5.3.

3.5.1 Zero-shot Relation Learning

We start with the scenario of learning extractors for relations that do not appear in the KB, *i.e.*, those that do not have any textual alignments. Such a scenario occurs in practice when a new relation is added to a KB for which there are no facts that connect the new relation to existing relations or textual surface forms. For accurate extraction of such relations, we can only rely on background domain knowledge, *e.g.*, in form of rules, to identify relevant textual alignments. However, at the same time, there are correlations between textual patterns that can be utilized for improved generalization. To simulate this setup, we remove all alignments between all entity

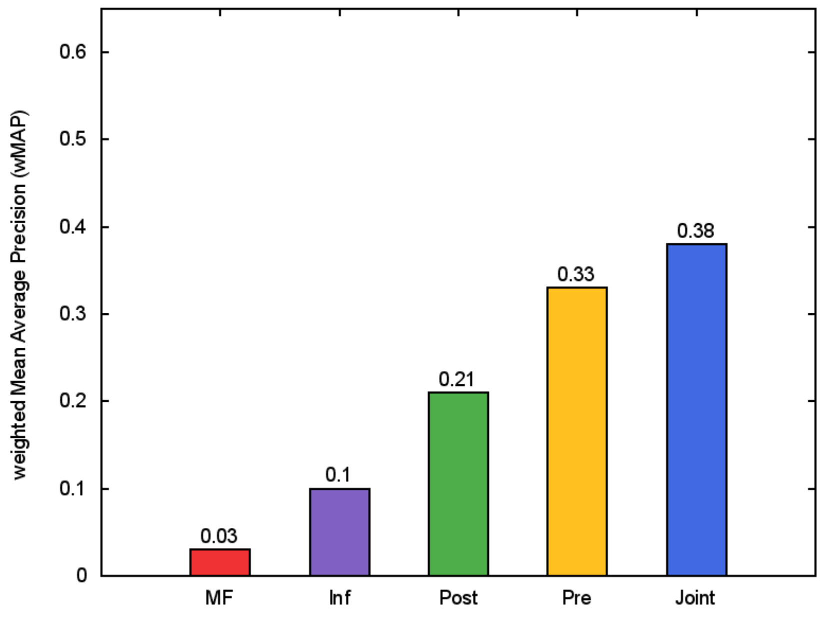


Figure 3.3: Weighted MAP scores for zero-shot relation learning.

Relation	#	MF	Inf	Post	Pre	Joint
`person/company`	102	0.07	0.03	0.15	0.31	**0.35**
`location/containedby`	72	0.03	0.06	0.14	0.22	**0.31**
`author/works_written`	27	0.02	0.05	0.18	**0.31**	0.27
`person/nationality`	25	0.01	*0.19*	0.09	0.15	*0.19*
`parent/child`	19	0.01	0.01	0.48	0.66	**0.75**
`person/place_of_birth`	18	0.01	0.43	0.40	0.56	**0.59**
`person/place_of_death`	18	0.01	0.24	0.23	**0.27**	0.23
`neighborhood/neighborhood_of`	11	0.00	0.00	0.60	0.63	**0.65**
`person/parents`	6	0.00	0.17	0.19	0.37	**0.65**
`company/founders`	4	0.00	0.25	0.13	0.37	**0.77**
`film/directed_by`	2	0.00	0.50	0.50	0.36	**0.51**
`film/produced_by`	1	0.00	*1.00*	*1.00*	*1.00*	*1.00*
MAP		0.01	0.23	0.34	0.43	**0.52**
Weighted MAP		0.03	0.10	0.21	0.33	**0.38**

Table 3.2: (Weighted) MAP with relations that do not appear in any of the annotated rules omitted from the evaluation. The difference between **Pre** and **Joint** is significant according to the sign-test ($p < 0.05$).

pairs and Freebase relations from the distant supervision data, use the extracted logic rules (Section 3.4) as background knowledge, and assess the ability of the different methods to recover the lost alignments.

Figure 3.3 provides detailed results. Unsurprisingly, matrix factorization (**MF**) performs poorly since predicate representations cannot be learned for the Freebase

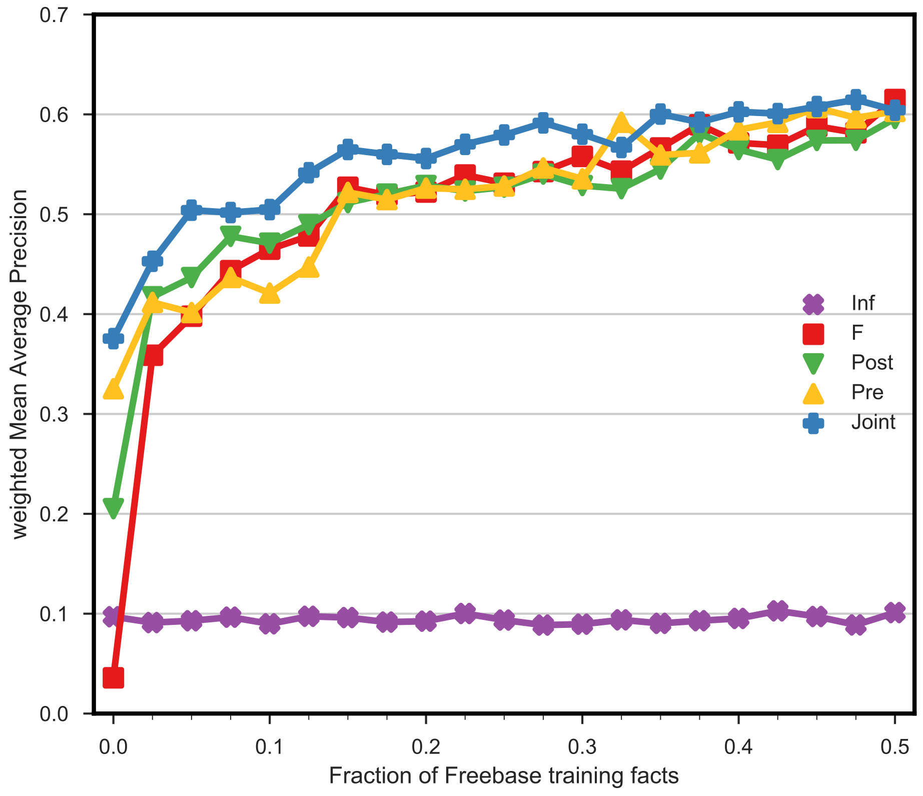


Figure 3.4: Weighted MAP of the various models as the fraction of Freebase training facts is varied. For 0% Freebase training facts we get the zero-shot relation learning results presented in Table 3.2.

relations without any observed facts. The fact that we see a non-zero MAP score for matrix factorization is due to random predictions. Symbolic logical inference (**Inf**) is limited by the number of known facts that appear as a premise in one of the implications, and thus performs poorly too. Although post-factorization inference (**Post**) is able to achieve a large improvement over logical inference, explicitly injecting logic rules into symbol representations using pre-factorization inference (**Pre**) or joint optimization (**Joint**) gives superior results. Finally, we observe that jointly optimizing the probability of facts and rules is able to best combine logic rules and textual patterns for accurate, zero-shot learning of relation extractors.

Table 3.2 shows detailed results for each of the Freebase test relations. Except for `author/works_written` and `person/place_of_death`, jointly optimizing the probability of facts and rules yields superior results.

3.5.2 Relations with Few Distant Labels

In this section, we study the scenario of learning relations that have only a few distant supervision alignments, *i.e.*, structured Freebase relations where only few textual patterns are observed for entity-pairs. In particular, we observe the behavior of the various methods as the amount of distant supervision is varied. We run all methods on training data that contains different fractions of Freebase training facts (and therefore different degrees of relation/text pattern alignment), but keep all textual patterns in addition to the set of annotated rules.

Figure 3.4 summarizes the results. The performance of symbolic logical inference does not depend on the amount of distant supervision data since it does not take advantage of the correlations in this data. Matrix factorization does not make use of logical rules, and thus is the baseline performance when only using distant supervision. For the factorization based methods, only a small fraction (15%) of the training data is needed to achieve around 0.50 weighted MAP performance, thus demonstrating that they are efficiently exploiting correlations between predicates, and generalizing to unobserved facts.

Pre-factorization inference, however, does not outperform post-factorization inference and is on par with matrix factorization for most of the curve. This suggests that it is not an effective way of injecting logic into symbol representations when ground atoms are also available. In contrast, the joint model learns symbol representations that outperform all other methods in the 0 to 30% Freebase training data interval. Beyond 30%, there seem to be sufficient Freebase facts for matrix factorization to encode these rules, thus yielding diminishing returns.

3.5.3 Comparison on Complete Data

Although the focus of our work is on injecting logical rules for relations without sufficient alignments to the knowledge base, we also present an evaluation on the complete distant supervision data by Riedel et al. [2013]. Compared to Riedel et al.'s matrix factorization model **Riedel13-F**, our reimplementation (**MF**) achieves a lower wMAP (64% vs 68%) and a higher MAP (66% vs 64%). We attribute this difference to the different loss function (BPR vs. negative log-likelihood). We show the precision-recall curve in Fig. 3.5, demonstrating that joint optimization provides benefits over the existing factorization and distant supervision techniques even on the complete dataset, and obtains 66% weighted MAP and 69% MAP, respectively. This improvement over the matrix factorization model can be explained by noting that the joint model reinforces high-quality annotated rules.

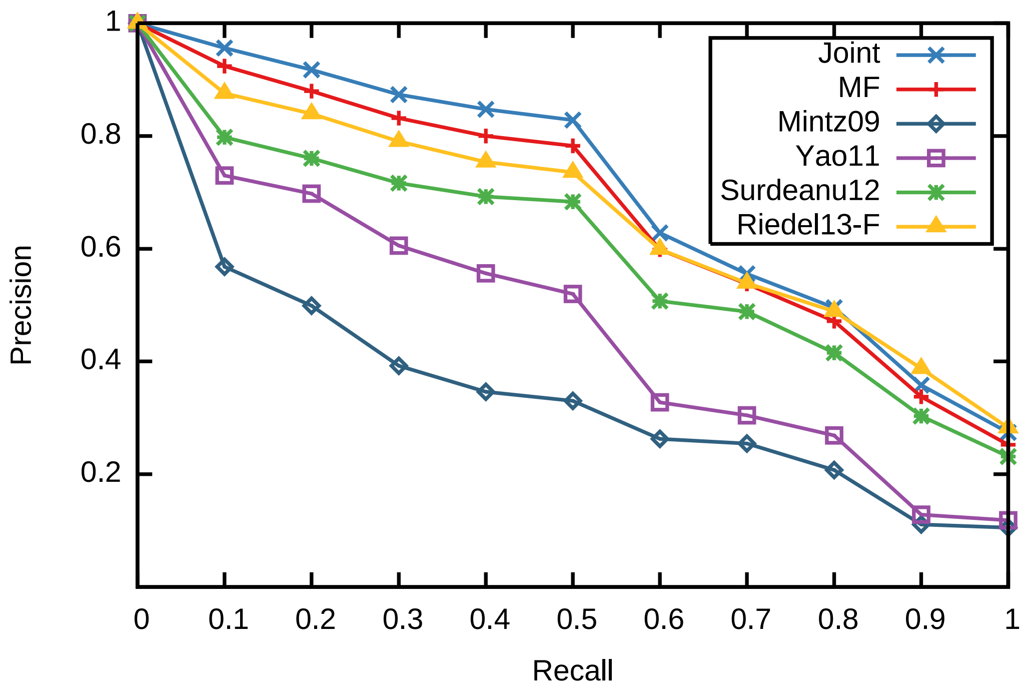


Figure 3.5: Precision-recall curve demonstrating that the **Joint** method, which incorporates annotated rules derived from the data, outperforms existing factorization approaches (**MF** and **Riedel13-F**).

3.6 Related Work

Embeddings for Knowledge Base Completion Many methods for embedding predicates and constants (or pairs of constants) based on training facts for knowledge base completion have been proposed in the past (see Section 2.3.2). Our work goes further in that we learn embeddings that follow not only factual but also first-order logic knowledge. Note that the method of regularizing symbol embeddings by rules described in this chapter are generally compatible with any existing neural link prediction model that provides per-atom scores between 0.0 and 1.0. In our experiments we only worked with matrix factorization as neural link prediction model but based on our work Guo et al. [2016] were able to incorporate transitivity rules into TransE [Bordes et al., 2013] which models entities separately instead of learning a representation for every entity pair.

Logical Inference A common alternative, where adding first-order logic knowledge is trivial, is to perform symbolic logical inference [Bos and Markert, 2005, Baader et al., 2007, Bos, 2008]. However, such purely symbolic approaches cannot deal with the uncertainty inherent to natural language and generalize poorly.

Probabilistic Inference To ameliorate some of the drawbacks of symbolic logical inference, probabilistic logic based approaches have been proposed [Schoenmackers et al., 2008, Garrette et al., 2011, Beltagy et al., 2013, 2014]. Since logical connections between relations are modeled explicitly, such approaches are generally hard to scale to large KBs. Specifically, approaches based on Markov Logic Networks (MLNs) [Richardson and Domingos, 2006] encode logical knowledge in dense, loopy graphical models, making structure learning, parameter estimation, and inference hard for the scale of our data. In contrast, in our model the logical knowledge is captured directly in symbol representations, leading to efficient inference at test time as we only have to calculate the forward pass of a neural link prediction model. Furthermore, as symbols are embedded in a low-dimensional vector space, we have a natural way of dealing with linguistic ambiguities and label errors that appear once OpenIE textual patterns are included as predicates for automated KB completion [Riedel et al., 2013].

Stochastic grounding is related to locally grounding a query in Programming with Personalized PageRank (ProPPR) [Wang et al., 2013]. One difference is that we use stochastically grounded rules as differentiable terms in a representation learning training objective, whereas in ProPPR such grounded rules are used for stochastic inference without learning symbol representations.

Weakly Supervised Learning Our work is also inspired by weakly supervised approaches [Ganchev et al., 2010] that use structural constraints as a source of indirect supervision. These methods have been used for several NLP tasks [Chang et al., 2007, Mann and McCallum, 2008, Druck et al., 2009, Singh et al., 2010]. The semi-supervised information extraction work by Carlson et al. [2010] is in spirit similar to our goal as they are using commonsense constraints to jointly train multiple information extractors. A main difference is that we are learning symbol representations and allow for arbitrarily complex logical rules to be used as regularizers for these representations.

Combining Symbolic and Distributed Representations There have been a number of recent approaches that combine trainable subsymbolic representations with symbolic knowledge. Grefenstette [2013] describes an isomorphism between first-order logic and tensor calculus, using full-rank matrices to exactly *memorize* facts. Based on this isomorphism, Rocktäschel et al. [2014] combine logic with matrix factorization for learning low-dimensional symbol embeddings that approximately satisfy given rules and generalize to unobserved facts on toy data. Our work extends this workshop paper by proposing a simpler formalism without tensor-based logical connectives, presenting results on a large real-world task, and demonstrating the utility of this approach for learning relations with no or few textual alignments.

Chang et al. [2014] use Freebase entity types as hard constraints in a tensor factorization objective for universal schema relation extraction. In contrast, our approach is imposing soft constraints that are formulated as universally quantified first-order rules.

de Lacalle and Lapata [2013] combine first-order logic knowledge with a topic model to improve surface pattern clustering for relation extraction. Since these rules only specify which relations can be clustered and which cannot, they do not capture the variety of dependencies embeddings can model, such as asymmetry. Lewis and Steedman [2013] use distributed representations to cluster predicates before logical inference. Again, this approach is not as expressive as learning subsymbolic representations for predicates, as clustering does not deal with asymmetric logical relationships between predicates.

Several studies have investigated the use of symbolic representations (such as dependency trees) to guide the composition of symbol representations [Clark and Pulman, 2007, Mitchell and Lapata, 2008, Coecke et al., 2010, Hermann and Blunsom, 2013]. Instead of guiding composition, we are using first-order logic rules as prior domain knowledge in form of regularizers to directly learn better symbol representations.

Combining symbolic information with neural networks has a long tradition. Towell and Shavlik [1994] introduce Knowledge-Based Artificial Neural Networks whose topology is isomorphic to a KB of facts and inference rules. There, facts are input units, intermediate conclusions hidden units, and final conclusions (inferred facts) output units. Unlike in our work, there are no learned symbol representations. Hölldobler et al. [1999] and Hitzler et al. [2004] prove that for every logic program there exists a recurrent neural network that approximates the semantics of that program. This is a theoretical insight that unfortunately does not provide a way of constructing such a neural network. Recently, Bowman [2013] demonstrated that Neural Tensor Networks (NTNs) [Socher et al., 2013] can accurately learn natural logic reasoning.

The method presented in this chapter is also related to the recently introduced Neural Equivalence Networks (EqNets) [Allamanis et al., 2017]. EqNets recursively construct neural representations of symbolic expressions to learn about equivalence classes. In our approach, we recursively construct neural networks for evaluating Boolean expressions and use them as regularizers to learn better symbol representations for automated KB completion.

3.7 Summary

In this chapter, we introduced a method for mapping symbolic first-order logic rules to differentiable terms that can be used to regularize symbol representations learned by neural link prediction models for automated KB completion. Specifically, we proposed a joint training objective that maximizes the probability of known training facts as well as propositional rules that we made continuous by replacing logical operators with differentiable functions. While this is inspired by Fuzzy Logic, our contribution is backpropagating a gradient from a negative log-likelihood loss through the propositional rule and a neural link prediction model that scores ground atoms to calculate a gradient with respect to vector representations of symbols. Subsequently, we update these representations using gradient descent, thereby encoding the ground truth of a propositional rule directly in the vector representations of symbols. This leads to efficient predictions at test time as we only have to calculate the forward pass of the neural link prediction model. We described a stochastic grounding process for incorporating first-order logic rules. Our experiments for automated KB completion show that the proposed method can be used to learn extractors for relations with little to no observed textual alignments, while at the same time benefiting from correlations between textual surface form patterns.

Chapter 4

Lifted Regularization of Predicate Representations by Implications

The method for incorporating first-order logic rules into symbol representations introduced in the previous chapter relies on stochastic grounding. Moreover, not only vector representations of predicates but also representations of pairs of constants are optimized to maximize the probability of provided rules. This is problematic for the following reasons.

Scalability Even with stochastic grounding, incorporating first-order logic rules with the method described so far is dependent on the size of the domain of constants. As an example take the simple rule `isMortal`(X) :– `isHuman`(X) and assume we observe `isHuman` for seven billion constants. Only for this single rule, we would already add seven billion loss terms to the training objective in Eq. 3.2.

Generalizability Since we backpropagate upstream gradients of a rule not only into predicate representations but also into representations of pairs of constants, there is no theoretical guarantee that the rule will indeed hold for constant pairs not observed during training.

Flexibility of Training Loss The previous method is not compatible with rank-based losses such as Bayesian Personalized Ranking (BPR). Instead, we had to use the negative log-likelihood loss, which results in lower performance compared to BPR for automated Knowledge Base (KB) completion.

Independence Assumption As explained in Section 3.2, we had to assume that the probability of ground atoms is conditionally independent given symbol representations. This assumption is already violated for the simple Boolean expression $[\![\mathrm{F} \wedge \mathrm{F}]\!]$ with $0 < [\![\mathrm{F}]\!] < 1$, which results in $[\![\mathrm{F} \wedge \mathrm{F}]\!] = [\![\mathrm{F}]\!]\,[\![\mathrm{F}]\!] < [\![\mathrm{F}]\!]$.

Ideally, we would like to have a way to incorporate first-order logic rules into symbol representations in a way that (i) is independent of the size of the domain of constants, (ii) generalizes to unseen constant pairs, (iii) can be used with a broader class of training objectives, and (iv) does not assume that the probability of ground atoms is conditionally independent given symbol representations.

In this chapter, we present a method that satisfies these desiderata, but only for simple implication rules instead of general function-free first-order logic rules, and for matrix factorization as neural link prediction model. However, we note that such simple implications are commonly used and can improve automated KB completion. The method we propose incorporates implications into vector representations of predicates while maintaining the computational efficiency of only modeling training facts. This is achieved by enforcing a partial order of predicate representations in the vector space, which is entirely independent of the number of constants in the domain. It only involves representations of the predicates that are mentioned in rules, as well as a general rule-independent constraint on the embedding space of constant pairs. In the example given above, if we require that every component of the vector representation $[\![\texttt{isHuman}]\!]$ is smaller or equal to the corresponding component of the predicate representation $[\![\texttt{isMortal}]\!]$, then we can show that the implication holds for any *non-negative* representation of a constant. Hence, our method avoids the need for separate loss terms for every ground atom resulting from grounding rules. In statistical relational learning, this type of approach is often referred to as *lifted* inference or learning [Poole, 2003, Braz, 2007] because it deals with groups of random variables at a first-order level. In this sense, our approach is a lifted form of rule injection. This allows us to impose a large number of rules while learning distributed representations of predicates and constant pairs. Furthermore, once these constraints are satisfied, the injected rules always hold, even for unseen but inferred ground atoms. In addition, it does not rely on the assumption of conditional independence of ground atoms.

4.1 Method

We want to incorporate implications of the form $\forall \mathrm{X}, \mathrm{Y} : \texttt{h}(\mathrm{X}, \mathrm{Y}) :\!-\ \texttt{b}(\mathrm{X}, \mathrm{Y})$ and we consider matrix factorization as the neural link prediction model for scoring atoms, *i.e.*, $[\![r(e_i, e_j)]\!] = \sigma([\![r]\!]^\top [\![e_i, e_j]\!])$ (Section 2.3.1). A necessary condition for the implication to hold is that for every possible assignment of constants e_i to X and e_j to Y, the score of $[\![\texttt{h}(e_i, e_j)]\!]$ needs to be as least as large as $[\![\texttt{b}(e_i, e_j)]\!]$. In the discrete case this means that if $[\![\texttt{b}(e_i, e_j)]\!]$ is 1.0 (True) within some small epsilon, $[\![\texttt{h}(e_i, e_j)]\!]$ needs to be 1.0 too, but not vice versa. As sigmoid is a monotonic

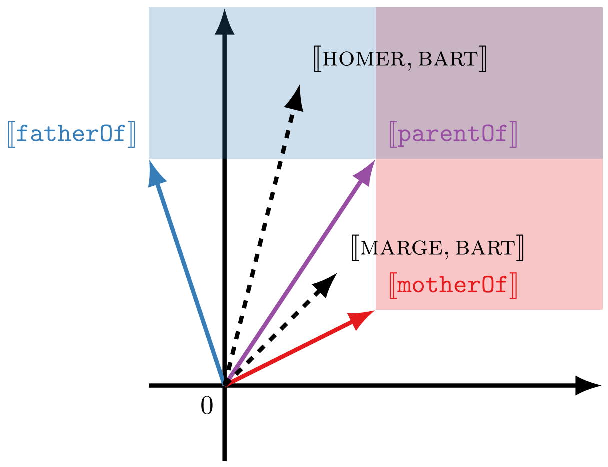


Figure 4.1: Example for implications that are directly captured in a vector space.

function, we can rewrite $\forall \mathrm{X}, \mathrm{Y} : \mathtt{h}(\mathrm{X}, \mathrm{Y}) \coloneq \mathtt{b}(\mathrm{X}, \mathrm{Y})$ in terms of grounded rules as the following condition:

$$\forall (e_i, e_j) \in \mathcal{C} : [\![\mathtt{h}]\!]^\top [\![e_i, e_j]\!] \geq [\![\mathtt{b}]\!]^\top [\![e_i, e_j]\!] . \tag{4.1}$$

Ordering symbol representations in a vector space such that implications are directly captured is inspired by Order Embeddings [Vendrov et al., 2016]. An example where Eq. 4.1 holds is illustrated in Fig. 4.1. Here, fatherOf and motherOf both imply parentOf, since every component of $[\![\mathtt{parentOf}]\!]$ is larger than the corresponding component in $[\![\mathtt{fatherOf}]\!]$ or $[\![\mathtt{motherOf}]\!]$. Predicate representations in the blue, red and purple area are implied by fatherOf, motherOf and parentOf, respectively. Both constant pair representations, $[\![\mathrm{HOMER}, \mathrm{BART}]\!]$ and $[\![\mathrm{MARGE}, \mathrm{BART}]\!]$, are non-negative. Thus, for any score of $[\![\mathtt{fatherOf}(\mathrm{HOMER}, \mathrm{BART})]\!]$, the score of $[\![\mathtt{parentOf}(\mathrm{HOMER}, \mathrm{BART})]\!]$ is larger, but not vice versa. Note that this holds for any representation of constant pairs $[\![e_i, e_j]\!]$ as long as it is non-negative (*i.e.* placed in the upper-right quadrant).

The main insight is that we can make this condition independent of e_i, e_j if we make sure that $[\![e_i, e_j]\!] \in \mathbb{R}^k_+$, *i.e.*, all constant representations are non-negative. Thus, Eq. 4.1 becomes

$$[\![\mathtt{h}]\!] \geq [\![\mathtt{b}]\!] , \quad \forall (e_i, e_j) \in \mathcal{C} : [\![e_i, e_j]\!] \in \mathbb{R}^k_+ \tag{4.2}$$

where $\geq$ is the component-wise comparison. In other words, for ensuring that b implies h for any pair of constants, we only need one relation-specific loss term that makes sure all components in $[\![\mathtt{h}]\!]$ are at least as large as in $[\![\mathtt{b}]\!]$, and one general restriction on the representations of constant pairs.

Non-negative Representation of Constant Pairs There are many choices for ensuring all constant pair representations are positive. One option is to initialize constant pair representations to non-negative vectors and projecting gradient updates to make sure they stay non-negative. Another option is to apply a transformation $f : \mathbb{R}^k \to \mathbb{R}^k_+$ before constant pair representations are used in a neural link prediction model for scoring atoms. For instance, we could use $\exp(x) = e^x$ or $\mathtt{ReLU}(x) = \max(0, x)$ for f. However, we choose to restrict constant representations even more than required, and decided to use a transformation to approximately Boolean embeddings [Kruszewski et al., 2015]. For every constant pair representation $[\![e_i, e_j]\!] \in \mathbb{R}^k$, we obtain a non-negative representation $[\![e_i, e_j]\!]_+ \in [0, 1]^k$ by applying the element-wise sigmoid function. Thus, the matrix factorization objective with BPR for facts in Eq. 2.9 becomes the following approximate loss

$$\mathfrak{L} = \sum_{\substack{r_s(e_i,e_j) \in \mathcal{O},\\ (e_m,e_n) \sim \mathcal{C},\\ r_s(e_m,e_n) \notin \mathcal{O}}} -w_r \log \sigma(\boldsymbol{v}_r^\top \sigma(\boldsymbol{v}_{ij}) - \boldsymbol{v}_r^\top \sigma(\boldsymbol{v}_{mn})) + \lambda_p \|\boldsymbol{v}_s\|_2^2 + \lambda_c(\|\boldsymbol{v}_{ij}\|_2^2 + \|\boldsymbol{v}_{mn}\|_2^2) \tag{4.3}$$

where we sample constant pairs as in Section 2.3.1.1. We denote the extension with sigmoidal constant pair representations of the matrix factorization model (**F**) by [Riedel et al., 2013] as **FS** (S = sigmoidal).

Implication Loss There are various ways for modeling $[\![h]\!] \geq [\![b]\!]$ to incorporate the implication $\mathtt{h} :\!- \mathtt{b}$. Here, we propose to use a hinge-like loss

$$\mathcal{L}([\![\mathtt{h} :\!- \mathtt{b}]\!]) = \sum_i^k \max(0, [\![\mathtt{b}]\!]_i - [\![\mathtt{h}]\!]_i + \epsilon) \tag{4.4}$$

where ϵ is a small positive margin to ensure that the gradient does not disappear before the inequality is actually satisfied. A nice property of this loss compared to the method presented in the previous chapter is that once the implication holds (*i.e.* $[\![\mathtt{h}]\!]$ is larger than $[\![\mathtt{b}]\!]$), the gradient is zero and both predicate representations are not further updated for this rule. For every given implication rule, we add the corresponding loss term to the fact loss in Eq. 4.3. The global approximate loss over facts and rules in a set $\mathcal{R}$ is thus

$$\mathfrak{L}' = \mathfrak{L} + \sum_{\mathtt{h} :- \mathtt{b} \in \mathcal{R}} \mathcal{L}([\![\mathtt{h} :\!- \mathtt{b}]\!]) \tag{4.5}$$

and we denote the resulting model as **FSL** (L = logic). Furthermore, we use a margin of $\epsilon = 0.01$ in all experiments. As in Chapter 3, we predict the probability of an atom $r_s(e_i, e_j)$ at test time via $[\![r_s(e_i, e_j)]\!]$, which is efficient as there is no logical inference.

4.2 Experiments

We follow the experimental setup from the previous chapter (Section 3.4) and evaluate on the NYT corpus [Riedel et al., 2013]. Again, we test how well the presented models can incorporate rules when there is no alignment between textual surface forms and Freebase relations (Zero-shot Relation Learning), and when the number of Freebase training facts is increased (Relations with Few Distant Labels). In addition, we experiment with rules automatically extracted from WordNet [Miller, 1995] to improve automated KB completion on the full dataset.

Incorporating Background Knowledge from WordNet We use WordNet hypernyms to generate rules for the NYT dataset. To this end, we iterate over all surface form patterns in the dataset and attempt to replace words in the pattern by their hypernyms. If the resulting surface form is contained in the dataset, we generate the corresponding rule. For instance, we generate a rule `#1-official-#2`(X, Y) :– `#1-diplomat-#2`(X, Y) since both patterns are contained in the dataset and we know from WordNet that `official` is a hypernym of `diplomat`. This resulted in 427 generated rules that we subsequently annotated manually, yielding 36 high-quality rules listed in Appendix B. Note that all of these rules are between surface form patterns. Thus, none of these rules has a Freebase relation as the head predicate. Although the test relations all originate from Freebase, we still hope to see improvements by transitive effects, such as better surface form representations that in turn help to predict Freebase facts.

4.2.1 Training Details

All models were implemented in TensorFlow [Abadi et al., 2016]. We use the hyper-parameters of Riedel et al. [2013], with $k = 100$ as the size of symbol representations and a weight of 0.01 for the ℓ_2 regularization (Eq. 4.3). We use ADAM [Kingma and Ba, 2015] for optimization with an initial learning rate of 0.005 and a batch size of 8192. The embeddings are initialized by sampling k values uniformly from $[-0.1, 0.1]$.

Test relation	#	**Riedel13-F**	**F**	**FS**	**FSL**
`person/company`	106	0.75	0.73	0.74	**0.77**
`location/containedby`	73	0.69	0.62	0.70	**0.71**
`person/nationality`	28	0.19	0.20	0.20	**0.21**
`author/works_written`	27	0.65	**0.71**	0.69	0.65
`person/place_of_birth`	21	0.72	0.69	**0.72**	0.70
`parent/child`	19	0.76	0.77	0.81	**0.85**
`person/place_of_death`	19	0.83	*0.85*	0.83	*0.85*
`neighborhood/neighborhood_of`	11	**0.70**	0.67	0.63	0.62
`person/parents`	6	0.61	0.53	*0.66*	*0.66*
`company/founders`	4	**0.77**	0.73	0.64	0.67
`sports_team/league`	4	**0.59**	0.44	0.43	0.56
`team_owner/teams_owned`	2	0.38	*0.64*	*0.64*	0.61
`team/arena_stadium`	2	*0.13*	*0.13*	*0.13*	0.12
`film/directed_by`	2	**0.50**	0.18	0.17	0.13
`broadcast/area_served`	2	0.58	0.83	0.83	**1.00**
`structure/architect`	2	*1.00*	*1.00*	*1.00*	*1.00*
`composer/compositions`	2	**0.67**	0.64	0.51	0.50
`person/religion`	1	*1.00*	*1.00*	*1.00*	*1.00*
`film/produced_by`	1	0.50	*1.00*	*1.00*	0.33
Weighted MAP		0.67	0.65	0.67	**0.69**

Table 4.1: Weighted MAP for our reimplementation of the matrix factorization model (**F**), compared to restricting the constant embedding space (**FS**) and to injecting WordNet rules (**FSL**). The orginial matrix factorization model by Riedel et al. [2013] is denoted as **Riedel13-F**.

4.3 Results and Discussion

Before turning to the injection of rules, we compare model **F** with model **FS**, and show that restricting the constant embedding space has a regularization effect rather than limiting the expressiveness of the model (Section 4.3.1). We then demonstrate that model **FSL** is capable of zero-shot learning (Section 4.3.2), that it can take advantage of alignments between textual surface forms and Freebase relations alongside rules (Section 4.3.3), and we show that injecting high-quality WordNet rules leads to improved predictions on the full dataset (Section 4.3.4). Lastly, we provide details on the computational efficiency of the lifted rule injection method (Section 4.3.5) and demonstrate that it correctly captures the asymmetry of implication rules (Section 4.3.6).

4.3.1 Restricted Embedding Space for Constants

Before incorporating external commonsense knowledge into relation representations, we were curious about how much we lose by restricting the embedding space of constant symbols to approximately Boolean embeddings. Surprisingly, we find that the expressiveness of the model does not suffer from this strong restriction. From Table 4.1 we see that restricting the constant embedding space (**FS**) yields a 2 percentage points higher weighted Mean Average Precision (MAP) compared to a real-valued constant embedding space (**F**). This result suggests that the restriction has a regularization effect that improves generalization. We also provide the original results for the matrix factorization model by Riedel et al. [2013] denoted as **Riedel13-F** for comparison. Due to a different implementation and optimization procedure, the results for our model **F** compared to **Riedel13-F** are slightly worse (65% vs 67% wMAP).

4.3.2 Zero-shot Relation Learning

In Section 3.5.1, we observed that by injecting implications where the head is a Freebase relation for which no training facts are available, we can infer Freebase facts based on rules and correlations between textual surface patterns. Here, we repeat this experiment. The lifted rule injection model (**FSL**) reaches a weighted MAP of 35%, comparable to the 38% of the method presented in the last chapter. For this experiment, we initialized the predicate representations of Freebase relations implied by the rules with negative random vectors sampled uniformly from $[-7.9, -8.1]$. The reason is that without any negative training facts for these relations, their components can only go up due to the lifted implication loss. Consequently, starting with high values before optimization would impede the freedom with which these representations can be ordered in the embedding space. This demonstrates that while **FSL** performs a bit worse than the **Joint** model in Chapter 3, it can still be used for zero-shot relation learning.

4.3.3 Relations with Few Distant Labels

Figure 4.2 shows how the relation extraction performance improves when more Freebase facts are added. As in the last chapter, it measures how well the proposed models, matrix factorization (**F**), propositionalized rule injection (**Joint**), and our lifted rule injection model (**FSL**), can make use of provided implication rules, as well as correlations between textual surface form patterns and increasing numbers of Freebase facts. Although **FSL** starts at a lower performance than **Joint** when no Freebase training facts are present, it outperforms **Joint** and a plain matrix factorization model by a substantial margin when provided with more than 7.5% of Freebase facts. This indicates that, in addition to being much faster than **Joint**, it can make better use of provided rules and few training facts. We attribute this

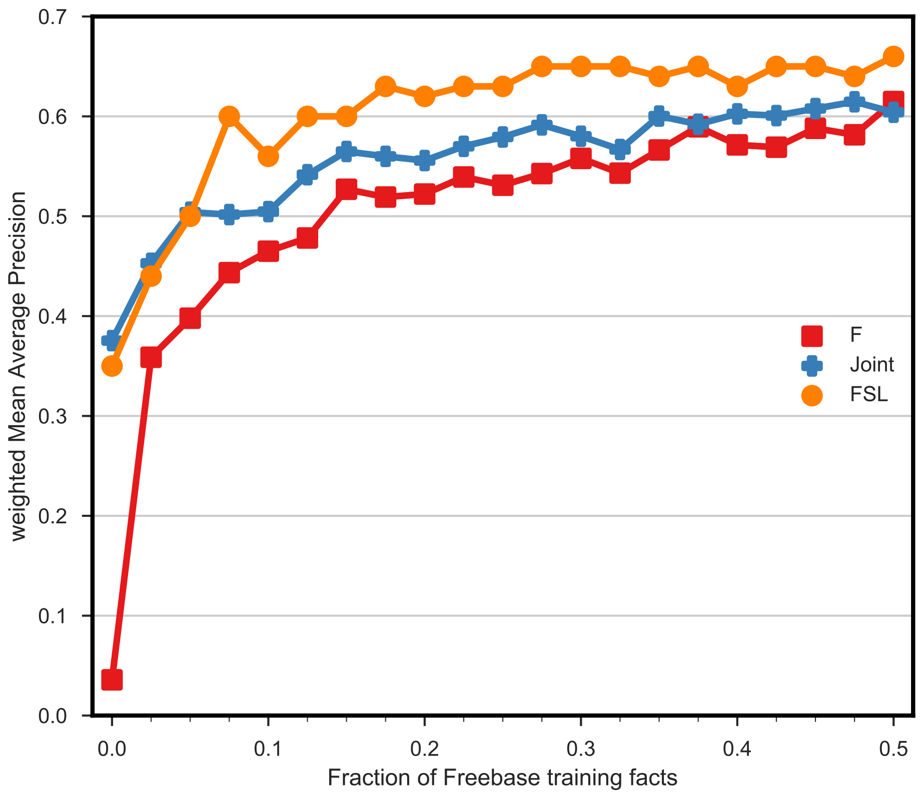


Figure 4.2: Weighted MAP for injecting rules as a function of the fraction of Freebase facts.

to being able to use BPR as loss for ground atoms and the regularization effect of restricting the embedding space of constants pairs. The former is compatible with our rule-injection method, but not with the approach of maximizing the expectation of propositional rules presented in the previous chapter.

4.3.4 Incorporating Background Knowledge from WordNet

In column **FSL** in Table 4.1, we show results obtained by injecting WordNet rules. Compared to **FS**, we obtain an increase of weighted MAP by 2%, as well as 4% compared to our reimplementation of the matrix factorization model **F**. This demonstrates that imposing a partial order based on implication rules can be used to incorporate logical commonsense knowledge and increase the quality of information extraction and automated KB completion systems. Note that our evaluation setting guarantees that only indirect effects of the rules are measured, *i.e.*, we do not use any rules directly implying Freebase test relations. Consequently, our increase of prediction performance is due to an improved predicate embedding space beyond those predicates that are explicitly stated in provided rules. For example, injecting

Rule head	:–	Rule body	FS $[\![$head$(e_i, e_j)]\!]$	FS $[\![$body$(\hat{e}_i, \hat{e}_j)]\!]$	FSL $[\![$head$(e_i, e_j)]\!]$	FSL $[\![$body$(\hat{e}_i, \hat{e}_j)]\!]$
`#1-organization-#2`	:–	`#1-party-#2`	0.70	0.86	0.99	0.22
`#1-parent-#2`	:–	`#1-father-#2`	0.72	0.89	0.96	0.00
`#1-lawyer-#2`	:–	`#1-prosecutor-#2`	0.87	0.80	0.99	0.01
`#1-newspaper-#2`	:–	`#1-daily-#2`	0.90	0.86	0.98	0.79
`#1-diplomat-#2`	:–	`#1-ambassador-#2`	0.93	0.84	0.31	0.05
Average over all rules			0.74	0.70	0.95	0.28

Table 4.2: Average score of facts with constants that appear in the body of facts (e_i, e_j) or in the head $(\hat{e}_i, \hat{e}_j)$ of a rule.

the rule `#1-parent-#2`(X, Y) :– `#1-father-#2`(X, Y) can contribute to improved predictions for the Freebase test relation `parent/child` via co-occurring entity pairs between `#1-parent-#2`(X, Y) and `parent/child`.

4.3.5 Computational Efficiency of Lifted Rule Injection

In order to assess the computational efficiency of the proposed method, we measure the time needed per training epoch when using a single 2.4GHz CPU core. We measure on average $6.33s$ per epoch when not using rules (model **FS**), against $6.76s$ and $6.97s$ when using 36 filtered and 427 unfiltered rules (model **FSL**), respectively. Increasing the number of rules from 36 to 427 leads to an increase of only 3% in computation time. Furthermore, using 427 rules only adds an overhead of 10% to the computation time needed for learning ground atoms. This demonstrates that lifted rule injection scales well with the number of rules.

4.3.6 Asymmetry

One concern with incorporating implications into a vector space is that the vector representation of the predicates in the head and body are simply moving closer together. This would violate the asymmetry of implications. In the experiments above we might not observe that this is a problem as we are only testing how well the model predicts facts for Freebase relations and not how well we can predict textual surface form patterns. Thus, we perform the following experiment. After incorporating WordNet rules of the form `head` :– `body`, we select all constant pairs (e_i, e_j) for which we observe `body`(e_i, e_j) in the training set. If the implication holds, $[\![$`head`$(e_i, e_j)]\!]$ should yield a high score. If we conversely select $(\hat{e}_i, \hat{e}_j)$ based on known facts `head`$(\hat{e}_i, \hat{e}_j)$ and assume that `head` and `body` are not equivalent, then we expect a lower score for $[\![$`body`$(\hat{e}_i, \hat{e}_j)]\!]$ than for $[\![$`head`$(\hat{e}_i, \hat{e}_j)]\!]$ as `body`$(\hat{e}_i, \hat{e}_j)$ might not be true.

Table 4.2 lists these scores for five sampled WordNet rules, and the average over all WordNet rules when injecting these rules (model **FSL**) or not (model **FS**). From this table, we can see that the score of the head atom is on average much higher than the score for the body atom once we incorporate the rule in the vector space (**FSL**). In contrast, if we only run matrix factorization with the restricted constant embedding space, we often see high predictions for both, the body and head atom. This suggests that matrix factorization merely captures similarity between predicates. In contrast, by injecting implications we trained predicate representations that indeed yield asymmetric predictions. Given a high score for a `body` ground atom, the model predicts a high score for the `head`, but not vice versa. The reason that we also get a high score for `body` ground atoms in the fourth rule is that `newspaper` and `daily` are synonymously used in training texts.

4.4 Related Work

Recent research on combining rules with learned vector representations has been important for new developments in the field of automated KB completion. Wang et al. [2015] demonstrated how different types of rules can be incorporated using an Integer Linear Programming approach. Wang and Cohen [2016] learned embeddings for facts and first-order logic rules using matrix factorization. Yet, all of these approaches, and the method presented in the previous chapter, ground first-order rules with constants in the domain. This limits their scalability towards large rule sets and KBs with large domains of constants. It formed an important motivation for our lifted rule injection model, which by construction does not suffer from that limitation. Wei et al. [2015] proposed an alternative strategy to tackle the scalability problem by reasoning on a filtered subset of ground atoms.

Wu et al. [2015] proposed to use the Path Ranking Algorithm (PRA) (Section 2.3.3) for capturing long-range interactions between entities in conjunction with modeling pairwise relations. Our model differs substantially from their approach, in that we consider pairs of constants instead of separate constants, and that we inject a provided set of rules. Yet, by creating a partial order in the relation embeddings as a result of injecting implication rules, model **FSL** can also capture interactions beyond the predicates directly mentioned in these rules, which we demonstrated in Section 4.3.4 by injecting rules between surface patterns and measuring an improvement on predictions for structured Freebase relations.

Combining logic and distributed representations is also an active field of research outside of automated KB completion. Recent advances include the work by Faruqui et al. [2015], who injected ontological knowledge from WordNet into word embeddings to improve performance on downstream NLP tasks. Furthermore, Vendrov et al. [2016] proposed to enforce a partial order in an embeddings space of images and phrases. Our method is related to such Order Embeddings since we define a partial order on relation embeddings. We extend this work to automated KB completion where we ensure that implications hold for all pairs of constants by introducing a restriction on the embedding space of constant pairs. Another important contribution is the recent work by Hu et al. [2016], who proposed a framework for injecting rules into general neural network architectures by jointly training on target outputs and on rule-regularized predictions provided by a so-called teacher network. Although quite different at first sight, their work could offer a way to use our model in various neural network architectures by integrating the proposed lifted loss into the teacher network.

4.5 Summary

We presented a fast approach for incorporating first-order implication rules into distributed representations of predicates for automated KB completion. We termed our approach *lifted rule injection*, as the main contribution over the previous chapter is the fact that it avoids the costly grounding of first-order implication rules and is thus independent of the size of the domain of constants. By construction, these rules are satisfied for any observed or unobserved ground atom. The presented approach requires a restriction on the embedding space of constant pairs. However, experiments on a large-scale real-world KB show that this does not impair the expressiveness of the learned representations. On the contrary, it appears to have a beneficial regularization effect.

By incorporating rules generated from WordNet hypernyms, our model improved over a matrix factorization baseline for KB completion. Especially for domains where annotation is costly and only small amounts of training facts are available, our approach provides a way to leverage external knowledge sources efficiently for inferring facts.

On the downside, the lifted rule injection method presented here is only applicable for implication rules and when using matrix factorization as the neural link prediction model. Furthermore, it is unclear how far regularizing predicate representations can be pushed without constraining the embedding space too much. Specifically, it is unclear how more complex rules such as transitivity can be incorporated in a lifted way. Hence, we are exploring a more direct synthesis of representation learning and first-order logic inference in the next chapter.

Chapter 5

End-to-End Differentiable Proving

Current state-of-the-art methods for automated Knowledge Base (KB) completion use neural link prediction models to learn distributed vector representations of symbols (*i.e.* subsymbolic representations) for scoring atoms [Nickel et al., 2012, Riedel et al., 2013, Socher et al., 2013, Chang et al., 2014, Yang et al., 2015, Toutanova et al., 2015, Trouillon et al., 2016]. Such subsymbolic representations enable these models to generalize to unseen facts by encoding similarities: If the vector of the predicate symbol `grandfatherOf` is similar to the vector of the symbol `grandpaOf`, both predicates likely express a similar relation. Likewise, if the vector of the constant symbol LISA is similar to MAGGIE, similar relations likely hold for both constants (*e.g.* they live in the same city, have the same parents etc.).

This simple form of reasoning based on similarities is remarkably effective for automatically completing large KBs. However, in practice it is often important to capture more complex reasoning patterns that involve several inference steps. For example, if ABE is the father of HOMER and HOMER is a parent of BART, we would like to infer that ABE is a grandfather of BART. Such transitive reasoning is inherently hard for neural link prediction models as they only learn to score facts locally. In contrast, symbolic theorem provers like Prolog [Gallaire and Minker, 1978] enable exactly this type of multi-hop reasoning. Furthermore, Inductive Logic Programming (ILP) [Muggleton, 1991] builds upon such provers to learn interpretable rules from data and to exploit them for reasoning in KBs. However, symbolic provers lack the ability to learn subsymbolic representations and similarities between them from large KBs, which limits their ability to generalize to queries with similar but not identical symbols.

While the connection between logic and machine learning has been addressed by statistical relational learning approaches, these models traditionally do not support reasoning with subsymbolic representations (*e.g.* Kok and Domingos [2007]), and when using subsymbolic representations they are not trained end-to-end from training data (*e.g.* Gardner et al. [2013, 2014], Beltagy et al. [2017]). Neural multi-hop reasoning models [Neelakantan et al., 2015, Peng et al., 2015, Das et al., 2017, Weissenborn, 2016, Shen et al., 2016] address the aforementioned limitations to some extent by encoding reasoning chains in a vector space or by iteratively refining subsymbolic representations of a question before comparison with answers. In many ways, these models operate like basic theorem provers, but they lack two of their most crucial ingredients: interpretability and straightforward ways of incorporating domain-specific knowledge in form of rules.

Our approach to this problem is inspired by recent neural network architectures like Neural Turing Machines [Graves et al., 2014], Memory Networks [Weston et al., 2014], Neural Stacks/Queues [Grefenstette et al., 2015, Joulin and Mikolov, 2015], Neural Programmer [Neelakantan et al., 2016], Neural Programmer-Interpreters [Reed and de Freitas, 2016], Hierarchical Attentive Memory [Andrychowicz et al., 2016] and the Differentiable Forth Interpreter [Bosnjak et al., 2017]. These architectures replace discrete algorithms and data structures by end-to-end differentiable counterparts that operate on real-valued vectors. At the heart of our approach is the idea to translate this concept to basic symbolic theorem provers, and hence combine their advantages (multi-hop reasoning, interpretability, easy integration of domain knowledge) with the ability to reason with vector representations of predicates and constants. Specifically, we keep variable binding symbolic but compare symbols using their subsymbolic vector representations.

In this chapter we introduce Neural Theorem Provers (NTPs): End-to-end differentiable provers for basic theorems formulated as queries to a KB. We use Prolog's backward chaining algorithm as a recipe for recursively constructing neural networks that are capable of proving queries to a KB using subsymbolic representations. The success score of such proofs is differentiable with respect to vector representations of symbols, which enables us to learn such representations for predicates and constants in ground atoms, as well as parameters of function-free first-order logic rules of predefined structure. By doing so, NTPs learn to place representations of similar symbols in close proximity in a vector space and to induce rules given prior assumptions about the structure of logical relationships in a KB such as transitivity. Furthermore, NTPs can seamlessly reason with provided domain-specific rules. As NTPs operate on distributed representations of symbols, a single hand-crafted rule can be leveraged for many proofs of queries with symbols that have a similar representation. Finally, NTPs demonstrate a high degree of interpretability as they induce latent rules that we can decode to human-readable symbolic rules.

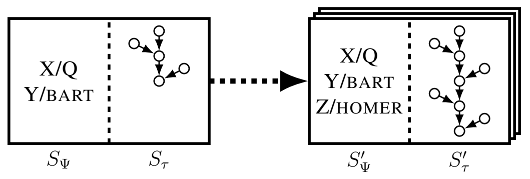


Figure 5.1: A module is mapping an upstream proof state (left) to a list of new proof states (right), thereby extending the substitution set S_Ψ and adding nodes to the computation graph of the neural network S_τ representing the proof success.

Our contributions are threefold: (i) We present the construction of NTPs inspired by Prolog's backward chaining algorithm and a differentiable unification operation using subsymbolic representations, (ii) we propose optimizations to this architecture by joint training with a neural link prediction model, batch proving, and approximate gradient calculation, and (iii) we experimentally show that NTPs can learn representations of symbols and function-free first-order rules of predefined structure, enabling them to learn to perform multi-hop reasoning on benchmark KBs and to outperform ComplEx [Trouillon et al., 2016], a state-of-the-art neural link prediction model, on three out of four KBs.

5.1 Differentiable Prover

In the following, we describe the recursive construction of NTPs – neural networks for end-to-end differentiable proving that allow us to calculate the gradient of proof successes with respect to vector representations of symbols. We define the construction of NTPs in terms of *modules* similar to dynamic neural module networks [Andreas et al., 2016]. Each module takes as inputs *discrete objects* (atoms and rules) and a *proof state*, and returns a list of new proof states (see Figure 5.1 for a graphical representation).

A proof state $S = (\Psi, \tau)$ is a tuple consisting of the substitution set Ψ constructed in the proof so far and a neural network τ that outputs a real-valued success score of a (partial) proof. While discrete objects and the substitution set are only used during construction of the neural network, once the network is constructed a continuous proof success score can be calculated for many different goals at training and test time. To summarize, modules are instantiated by discrete objects and the substitution set. They construct a neural network representing the (partial) proof success score and recursively instantiate submodules to continue the proof.

The shared signature of modules is $\mathcal{D} \times \mathcal{S} \to \mathcal{S}^N$ where $\mathcal{D}$ is a domain that controls the construction of the network, $\mathcal{S}$ is the domain of proof states, and N is the number of output proof states. Furthermore, let S_Ψ denote the substitution set of the proof state S and let S_τ denote the neural network for calculating the proof success. Akin to the pseudocode of backward chaining in Chapter 2, we use pseudocode in style of a functional programming language to define the behavior of modules and auxiliary functions.

5.1.1 Unification Module

Unification of two atoms, *e.g.*, a goal that we want to prove and a rule head, is a central operation in backward chaining. Two non-variable symbols (predicates or constants) are checked for equality and the proof can be aborted if this check fails. However, we want to be able to apply rules even if symbols in the goal and head are not equal but similar in meaning (*e.g.* `grandfatherOf` and `grandpaOf`) and thus replace symbolic comparison with a computation that measures the similarity of both symbols in a vector space.

The module `unify` updates a substitution set and creates a neural network for comparing the vector representations of non-variable symbols in two sequences of terms. The signature of this module is $\mathcal{L} \times \mathcal{L} \times \mathcal{S} \to \mathcal{S}$ where $\mathcal{L}$ is the domain of lists of terms. `unify` takes two atoms represented as lists of terms and an upstream proof state, and maps these to a new proof state (substitution set and proof success). To this end, `unify` iterates through the list of terms of two atoms and compares their symbols. If one of the symbols is a variable, a substitution is added to the substitution set. Otherwise, the vector representations of the two non-variable symbols are compared using a Radial Basis Function (RBF) kernel [Broomhead and Lowe, 1988] where μ is a hyperparameter that we set to $\frac{1}{\sqrt{2}}$ in our experiments. The following pseudocode implements `unify`. Note that "_" matches every argument and that the order matters, *i.e.*, if arguments match a line, subsequent lines are not evaluated.

1. $\texttt{unify}_{\boldsymbol{\theta}}([\,],[\,],S) = S$
2. $\texttt{unify}_{\boldsymbol{\theta}}([\,],\_,\_) = \texttt{FAIL}$
3. $\texttt{unify}_{\boldsymbol{\theta}}(\_,[\,],\_) = \texttt{FAIL}$
4. $\texttt{unify}_{\boldsymbol{\theta}}(h:\mathrm{H}, g:\mathrm{G}, S) = \texttt{unify}_{\boldsymbol{\theta}}(\mathrm{H},\mathrm{G},S') = (S'_\Psi, S'_\tau)$ where

$$S'_\Psi = \left\{\begin{array}{ll} S_\Psi \cup \{h/g\} & \text{if } h \in \mathcal{V} \\ S_\Psi \cup \{g/h\} & \text{if } g \in \mathcal{V}, h \notin \mathcal{V} \\ S_\Psi & \text{otherwise} \end{array}\right\}$$

$$S'_\tau = \min\left(S_\tau, \left\{\begin{array}{ll} \exp\left(\frac{-\|\boldsymbol{\theta}_{h:} - \boldsymbol{\theta}_{g:}\|_2}{2\mu^2}\right) & \text{if } h, g \notin \mathcal{V} \\ 1 & \text{otherwise} \end{array}\right\}\right)$$

Here, S' refers to the new proof state, $\mathcal{V}$ refers to the set of variable symbols, h/g is a substitution from the variable symbol h to the symbol g, and $\boldsymbol{\theta}_{g:}$ denotes the embedding lookup of the non-variable symbol with index g. `unify` is parameterized by an embedding matrix $\boldsymbol{\theta} \in \mathbb{R}^{|\mathcal{Z}|\times k}$ where $\mathcal{Z}$ is the set of non-variables symbols and k is the dimension of vector representations of symbols. Furthermore, `FAIL` represents a unification failure due to mismatching arity of two atoms. Once a failure is reached, we abort the creation of the neural network for this branch of proving. In addition, we constrain proofs to be cycle-free by checking whether a variable is already bound. Note that this is a simple heuristic that prohibits applying the same non-ground rule twice. There are more sophisticated ways for finding and avoiding cycles in a proof graph such that the same rule can still be applied multiple times (*e.g.* Gelder [1987]), but we leave this for future work.

Example Assume that we are unifying two atoms $[\texttt{grandpaOf}, \text{ABE}, \text{BART}]$ and $[s, \text{Q}, i]$ given an upstream proof state $S = (\varnothing, \tau)$ where the latter input atom has placeholders for a predicate s and a constant i, and the neural network τ would output 0.7 when evaluated. Furthermore, assume `grandpaOf`, ABE and BART represent the indices of the respective symbols in a global symbol vocabulary. Then, the new proof state constructed by `unify` is:

$$\begin{aligned}&\texttt{unify}_{\boldsymbol{\theta}}([\texttt{grandpaOf}, \text{ABE}, \text{BART}], [s, \text{Q}, i], (\varnothing, \tau)) = (S'_\Psi, S'_\tau) = \\ &\quad(\{\text{Q}/\text{ABE}\}, \min(\tau, \exp(-\|\boldsymbol{\theta}_{\texttt{grandpaOf}:} - \boldsymbol{\theta}_{s:}\|_2), \exp(-\|\boldsymbol{\theta}_{\text{BART}:} - \boldsymbol{\theta}_{i:}\|_2)))\end{aligned}$$

Thus, the output score of the neural network S'_τ will be high if the subsymbolic representation of the input s is close to `grandpaOf` and the input i is close to BART. However, the score cannot be higher than 0.7 due to the upstream proof success score in the forward pass of the neural network τ. Note that in addition to extending the neural network τ to S'_τ, this module also outputs a substitution set $\{\text{Q}/\text{ABE}\}$ at graph creation time that will be used to instantiate submodules.

Furthermore, note that unification is applied multiple times for a proof that involves more than one step, resulting in chained application of the RBF kernel and $\min$ operations. The choice of $\min$ stems from the property that for a successful proof, all unifications should be successful (conjunction). This could also be realized with a multiplication of unification scores along the proof, but it would likely result in unstable optimization for longer proofs due to exploding gradients.

5.1.2 OR Module

Based on `unify`, we now define the `or` module which attempts to apply rules in a KB. The signature of `or` is $\mathcal{L} \times \mathbb{N} \times \mathcal{S} \to \mathcal{S}^N$ where $\mathcal{L}$ is the domain of goal atoms and $\mathbb{N}$ is the domain of integers used for specifying the maximum proof depth of the neural network. Furthermore, N is the number of possible output proof states for a goal of a given structure and a provided KB.[1] We implement `or` as

1. $\texttt{or}^{\mathfrak{K}}_{\boldsymbol{\theta}}(\mathrm{G}, d, S) = [S' \mid S' \in \texttt{and}^{\mathfrak{K}}_{\boldsymbol{\theta}}(\mathbb{B}, d, \texttt{unify}_{\boldsymbol{\theta}}(\mathrm{H}, \mathrm{G}, S))$ for $\mathrm{H} \coloneq \mathbb{B} \in \mathfrak{K}]$

where $\mathrm{H} \coloneq \mathbb{B}$ denotes a rule in a given KB $\mathfrak{K}$ with a head atom H and a list of body atoms $\mathbb{B}$. In contrast to the symbolic OR method, the `or` module is able to use the `grandfatherOf` rule above for a query involving `grandpaOf` provided that the subsymbolic representations of both predicates are similar as measured by the RBF kernel in the `unify` module.

Example For a goal $[s, \mathrm{Q}, i]$, `or` would instantiate an `and` submodule based on the rule $[\texttt{grandfatherOf}, \mathrm{X}, \mathrm{Y}] \coloneq [[\texttt{fatherOf}, \mathrm{X}, \mathrm{Z}], [\texttt{parentOf}, \mathrm{Z}, \mathrm{Y}]]$ as follows

$$\begin{aligned}&\texttt{or}^{\mathfrak{K}}_{\boldsymbol{\theta}}([s, \mathrm{Q}, i], d, S) = \\ &\qquad [S' \mid S' \in \texttt{and}^{\mathfrak{K}}_{\boldsymbol{\theta}}([[\texttt{fatherOf}, \mathrm{X}, \mathrm{Z}], [\texttt{parentOf}, \mathrm{Z}, \mathrm{Y}]], d, \underbrace{(\{\mathrm{X}/\mathrm{Q}, \mathrm{Y}/i\}, \hat{S}_\tau)}_{\text{result of } \texttt{unify}}), \ldots]\end{aligned}$$

5.1.3 AND Module

For implementing `and` we first define an auxiliary function called substitute which applies substitutions to variables in an atom if possible. This is realized via

1. $\text{substitute}([\,], \_) = [\,]$
2. $\text{substitute}(g : \mathrm{G}, \Psi) = \left\{\begin{array}{ll} x & \text{if } g/x \in \Psi \\ g & \text{otherwise} \end{array}\right\} : \text{substitute}(\mathrm{G}, \Psi)$

For example, $\text{substitute}([\texttt{fatherOf}, \mathrm{X}, \mathrm{Z}], \{\mathrm{X}/\mathrm{Q}, \mathrm{Y}/i\}) = [\texttt{fatherOf}, \mathrm{Q}, \mathrm{Z}]$.

[1]The creation of the neural network is dependent on the KB but also the structure of the goal. For instance, the goal $s(\mathrm{Q}, i)$ would result in a different neural network, and hence a different number of output proof states, than $s(i, j)$.

The signature of and is $\mathcal{L} \times \mathbb{N} \times \mathcal{S} \rightarrow \mathcal{S}^N$ where $\mathcal{L}$ is the domain of lists of atoms and N is the number of possible output proof states for a list of atoms with a known structure and a provided KB. This module is implemented as

1. $\texttt{and}^{\mathfrak{K}}_{\boldsymbol{\theta}}(\_, \_, \texttt{FAIL}) = \texttt{FAIL}$
2. $\texttt{and}^{\mathfrak{K}}_{\boldsymbol{\theta}}(\_, 0, \_) = \texttt{FAIL}$
3. $\texttt{and}^{\mathfrak{K}}_{\boldsymbol{\theta}}([\,], \_, S) = S$
4. $\texttt{and}^{\mathfrak{K}}_{\boldsymbol{\theta}}(\mathrm{G} : \mathbb{G}, d, S) =$
   $[S'' \mid S'' \in \texttt{and}^{\mathfrak{K}}_{\boldsymbol{\theta}}(\mathbb{G}, d, S') \text{ for } S' \in \texttt{or}^{\mathfrak{K}}_{\boldsymbol{\theta}}(\text{substitute}(\mathrm{G}, S_\Psi), d-1, S)]$

where the first two lines define the failure of a proof, either because of an upstream unification failure that has been passed from the or module (line 1), or because the maximum proof depth has been reached (line 2). Line 3 specifies a proof success, *i.e.*, the list of subgoals is empty before the maximum proof depth has been reached. Lastly, line 4 defines the recursion: The first subgoal G is proven by instantiating an or module after substitutions are applied, and every resulting proof state S' is used for proving the remaining subgoals $\mathbb{G}$ by again instantiating and modules.

Example Continuing the example from Section 5.1.2, the and module would instantiate submodules as follows:

$$\texttt{and}^{\mathfrak{K}}_{\boldsymbol{\theta}}([[\texttt{fatherOf}, \mathrm{X}, \mathrm{Z}], [\texttt{parentOf}, \mathrm{Z}, \mathrm{Y}]], d, \underbrace{(\{\mathrm{X}/\mathrm{Q}, \mathrm{Y}/i\}, \hat{S}_\tau)}_{\text{result of } \texttt{unify} \text{ in } \texttt{or}}) =$$

$$[S'' \mid S'' \in \texttt{and}^{\mathfrak{K}}_{\boldsymbol{\theta}}([[\texttt{parentOf}, \mathrm{Z}, \mathrm{Y}]], d, S')$$
$$\text{for } S' \in \texttt{or}^{\mathfrak{K}}_{\boldsymbol{\theta}}(\underbrace{[\texttt{fatherOf}, \mathrm{Q}, \mathrm{Z}]}_{\text{result of substitute}}, d-1, \underbrace{(\{\mathrm{X}/\mathrm{Q}, \mathrm{Y}/i\}, \hat{S}_\tau)}_{\text{result of } \texttt{unify} \text{ in } \texttt{or}})]$$

5.1.4 Proof Aggregation

Finally, we define the overall success score of proving a goal G using a KB $\mathfrak{K}$ with parameters $\boldsymbol{\theta}$ as

$$\texttt{ntp}^{\mathfrak{K}}_{\boldsymbol{\theta}}(\mathrm{G}, d) = \underset{\substack{S \in \texttt{or}^{\mathfrak{K}}_{\boldsymbol{\theta}}(\mathrm{G}, d, (\varnothing, 1)) \\ S \neq \texttt{FAIL}}}{\arg\max} S_\tau$$

where d is a predefined maximum proof depth and the initial proof state is set to an empty substitution set and a proof success score of 1. Hence, the success of proving

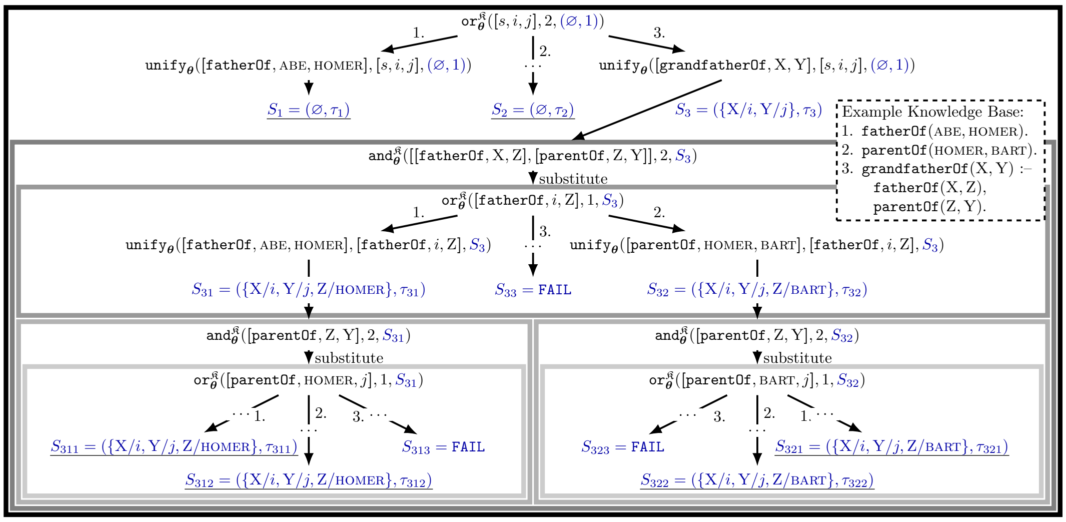

or$_{\theta}^{\mathfrak{K}}([s, i, j], 2, (\varnothing, 1))$
1.
2.
3.
unify$_{\theta}$([fatherOf, ABE, HOMER], $[s, i, j], (\varnothing, 1)$)
unify$_{\theta}$([grandfatherOf, X, Y], $[s, i, j], (\varnothing, 1)$)
$S_1 = (\varnothing, \tau_1)$
$S_2 = (\varnothing, \tau_2)$
$S_3 = (\{X/i, Y/j\}, \tau_3)$
Example Knowledge Base:
1. fatherOf(ABE, HOMER).
2. parentOf(HOMER, BART).
3. grandfatherOf(X, Y) :– fatherOf(X, Z), parentOf(Z, Y).
and$_{\theta}^{\mathfrak{K}}$([[fatherOf, X, Z], [parentOf, Z, Y]], 2, S_3)
substitute
or$_{\theta}^{\mathfrak{K}}$([fatherOf, i, Z], 1, S_3)
unify$_{\theta}$([fatherOf, ABE, HOMER], [fatherOf, i, Z], S_3)
unify$_{\theta}$([parentOf, HOMER, BART], [fatherOf, i, Z], S_3)
$S_{31} = (\{X/i, Y/j, Z/\text{HOMER}\}, \tau_{31})$
$S_{33} = \texttt{FAIL}$
$S_{32} = (\{X/i, Y/j, Z/\text{BART}\}, \tau_{32})$
and$_{\theta}^{\mathfrak{K}}$([parentOf, Z, Y], 2, S_{31})
and$_{\theta}^{\mathfrak{K}}$([parentOf, Z, Y], 2, S_{32})
or$_{\theta}^{\mathfrak{K}}$([parentOf, HOMER, j], 1, S_{31})
or$_{\theta}^{\mathfrak{K}}$([parentOf, BART, j], 1, S_{32})
$S_{311} = (\{X/i, Y/j, Z/\text{HOMER}\}, \tau_{311})$
$S_{313} = \texttt{FAIL}$
$S_{312} = (\{X/i, Y/j, Z/\text{HOMER}\}, \tau_{312})$
$S_{323} = \texttt{FAIL}$
$S_{321} = (\{X/i, Y/j, Z/\text{BART}\}, \tau_{321})$
$S_{322} = (\{X/i, Y/j, Z/\text{BART}\}, \tau_{322})$

Figure 5.2: Exemplary construction of an NTP computation graph for a toy knowledge base. Indices on arrows correspond to application of the respective KB rule. Proof states (blue) are subscripted with the sequence of indices of the rules that were applied. Underlined proof states are aggregated to obtain the final proof success. Boxes visualize instantiations of modules (omitted for unify). The proofs S_{33}, S_{313} and S_{323} fail due to cycle-detection (the same rule cannot be applied twice).

a goal is a max-pooling operation over the output of neural networks representing all possible proofs up to some depth.

Example Figure 5.2 illustrates an examplary NTP computation graph constructed for a toy KB. Note that such an NTP is constructed once before training, and can then be used for proving goals of the structure $[s, i, j]$ at training and test time where s is the index of an input predicate, and i and j are indices of input constants. Final proof states which are used in proof aggregation are underlined.

5.1.5 Neural Inductive Logic Programming

We can use NTPs for ILP by gradient descent instead of a combinatorial search over the space of rules as, for example, done by the First Order Inductive Learner (FOIL) [Quinlan, 1990]. Specifically, we are using the concept of learning from entailment [Muggleton, 1991] to induce rules that let us prove known ground atoms, but that do not give high proof success scores to sampled unknown ground atoms.

Let $\boldsymbol{\theta}_{r:}, \boldsymbol{\theta}_{s:}, \boldsymbol{\theta}_{t:} \in \mathbb{R}^k$ be representations of some unknown predicates with indices r, s and t respectively. The prior knowledge of a transitivity between three unknown predicates can be specified via $r(\mathrm{X}, \mathrm{Y}) \coloneq s(\mathrm{X}, \mathrm{Z}), t(\mathrm{Z}, \mathrm{Y})$. We call this a *parameterized rule* as the corresponding predicates are unknown and their representations are learned from data. Such a rule can be used for proofs at training and test time in the same way as any other given rule. During training, the predicate representations of parameterized rules are optimized jointly with all other subsymbolic representations. Thus, the model can adapt parameterized rules such that proofs for known facts succeed while proofs for sampled unknown ground atoms fail, thereby inducing rules of predefined structures like the one above. Inspired by Wang and Cohen [2015], we use rule templates for conveniently defining the structure of multiple parameterized rules by specifying the number of parameterized rules that should be instantiated for a given rule structure (see Section 5.3.1 for examples).

5.1.5.1 Rule Decoding and Implicit Rule Confidence

For inspection after training, we decode a parameterized rule by searching for the closest representations of known predicates. Given an induced rule such as $\boldsymbol{\theta}_{1:}(\mathrm{X},\mathrm{Y})$:– $\boldsymbol{\theta}_{2:}(\mathrm{X},\mathrm{Z}), \boldsymbol{\theta}_{2:}(\mathrm{Z},\mathrm{Y})$ where $\boldsymbol{\theta}_{1:}$ and $\boldsymbol{\theta}_{2:}$ have been trained, we find the closest representation of a known predicate for every parameterized predicate representation in the rule (*e.g.*, $\boldsymbol{\theta}_{1:} \rightarrow \boldsymbol{\theta}_{\texttt{grandparentOf}:}$, $\boldsymbol{\theta}_{2:} \rightarrow \boldsymbol{\theta}_{\texttt{parentOf}:}$). Formally, we decode $\boldsymbol{\theta}_{i:}$ to a predicate symbol from the set of all predicates $\mathcal{P}$ using

$$\texttt{decode}(\boldsymbol{\theta}_{i:}) = \arg\max_{r_s \in \mathcal{P}} \exp(-\|\boldsymbol{\theta}_{i:} - \boldsymbol{\theta}_{r_s:}\|_2). \tag{5.1}$$

In addition, we provide users with a rule confidence by taking the minimum similarity between unknown and decoded predicate representations using the RBF kernel in `unify`. Let $\Theta = [\boldsymbol{\theta}_i]$ be the list of predicate representations of a parameterized rule. The confidence of that rule is then calculated as

$$\gamma = \min_{\boldsymbol{\theta}_{i:} \in \Theta} \max_{r_s \in \mathcal{P}} \exp(-\|\boldsymbol{\theta}_{i:} - \boldsymbol{\theta}_{r_s:}\|_2). \tag{5.2}$$

This confidence score is an upper bound on the proof success score that can be achieved when the induced rule is used in proofs.

5.2 Optimization

In this section, we present the basic training loss that we use for NTPs, a training loss where a neural link prediction models is used as auxiliary task, as well as various computational optimizations.

5.2.1 Training Objective

Let $\mathcal{K}$ be the set of known facts in a given KB. Usually, we do not observe negative facts and thus resort to sampling corrupted ground atoms as done in previous work [Bordes et al., 2013]. Specifically, for every $[s,i,j] \in \mathcal{K}$ we obtain corrupted ground atoms $[s,\hat{\imath},j], [s,i,\hat{\jmath}], [s,\tilde{\imath},\tilde{\jmath}] \notin \mathcal{K}$ by sampling $\hat{\imath}, \hat{\jmath}, \tilde{\imath}$ and $\tilde{\jmath}$ from the set of constants. These corrupted ground atoms are resampled in every iteration of training, and we denote the set of known and corrupted ground atoms together with their target score (1.0 for known ground atoms and 0.0 for corrupted ones) as $\mathcal{T}$. We use the negative log-likelihood of the proof success score as loss function for an NTP with parameters $\boldsymbol{\theta}$ and a given KB $\mathfrak{K}$

$$\mathcal{L}_{\mathtt{ntp}^{\mathfrak{K}}_{\boldsymbol{\theta}}} = \sum_{([s,i,j],y)\,\in\,\mathcal{T}} -y\log(\mathtt{ntp}^{\mathfrak{K}}_{\boldsymbol{\theta}}([s,i,j],d)_{\tau}) - (1-y)\log(1-\mathtt{ntp}^{\mathfrak{K}}_{\boldsymbol{\theta}}([s,i,j],d)_{\tau})$$

where $[s,i,j]$ is a training ground atom and y its target proof success score. Note that since in our application all training facts are ground atoms, we only make use of the proof success score τ and not the substitution list of the resulting proof state. We can prove known facts trivially by a unification with themselves, resulting in no parameter updates during training and hence no generalization. Therefore, during training we are masking the calculation of the unification success of a known ground atom that we want to prove. Specifically, we set the unification score to 0 to temporarily hide that training fact and assume it can be proven from other facts and rules in the KB.

5.2.2 Neural Link Prediction as Auxiliary Loss

At the beginning of training all subsymbolic representations are initialized randomly. When unifying a goal with all facts in a KB we consequently get very noisy success scores in early stages of training. Moreover, as only the maximum success score will result in gradient updates for the respective subsymbolic representations along the maximum proof path, it can take a long time until NTPs learn to place similar symbols close to each other in the vector space and to make effective use of rules.

To speed up learning subsymbolic representations, we train NTPs jointly with ComplEx [Trouillon et al., 2016] (Eq. 2.15 in Chapter 2). ComplEx and the NTP share the same subsymbolic representations, which is feasible as the RBF kernel in `unify` is also defined for complex vectors. While the NTP is responsible for multi-hop reasoning, the neural link prediction model learns to score ground atoms locally. At test time, only the NTP is used for predictions. Thus, the training loss for ComplEx can be seen as an auxiliary loss for the subsymbolic representations learned by the NTP. We term the resulting model NTPλ. Based on the loss in Section 5.2.1, the joint training loss is defined as

$$\mathcal{L}_{\mathtt{ntp}\lambda_{\boldsymbol{\theta}}^{\mathfrak{K}}} = \mathcal{L}_{\mathtt{ntp}_{\boldsymbol{\theta}}^{\mathfrak{K}}} + \sum_{([s,i,j],y)\,\in\,\mathcal{T}} \big(-y\log(\mathtt{complex}_{\boldsymbol{\theta}}(s,i,j)) \\ -(1-y)\log(1-\mathtt{complex}_{\boldsymbol{\theta}}(s,i,j))\big)$$

where $[s,i,j]$ is a training atom, y is its ground truth target, and $\mathtt{complex}(s,i,j) = p_{sij}$ as defined in Eq. 2.15.

5.2.3 Computational Optimizations

NTPs as described above suffer from severe computational limitations since the neural network is representing all possible proofs up to some predefined depth. In contrast to symbolic backward chaining where a proof can be aborted as soon as unification fails, in differentiable proving we only get a unification failure for atoms whose arity does not match or when we detect cyclic rule application. We propose two optimizations to speed up NTPs. First, we make use of modern GPUs by batch processing many proofs in parallel (Section 5.2.3.1). Second, we exploit the sparseness of gradients caused by the $\min$ and $\max$ operations used in the unification and proof aggregation respectively to derive a heuristic for a truncated forward and backward pass that drastically reduces the number of proofs that have to be considered for calculating gradients (Section 5.2.3.2).

5.2.3.1 Batch Proving

Let $\boldsymbol{A} \in \mathbb{R}^{N\times k}$ be a matrix of N subsymbolic representations that are to be unified with M other representations $\boldsymbol{B} \in \mathbb{R}^{M\times k}$. We can adapt the unification module to calculate the unification success in a batched way using

$$\exp\left(-\sqrt{\left(\begin{bmatrix}\sum_{i=1}^{k}\boldsymbol{A}_{1i}^2\\ \vdots \\ \sum_{i=1}^{k}\boldsymbol{A}_{Ni}^2\end{bmatrix}\mathbf{1}_M^\top\right) + \left(\mathbf{1}_N\begin{bmatrix}\sum_{i=1}^{k}\boldsymbol{B}_{1i}^2\\ \vdots \\ \sum_{i=1}^{k}\boldsymbol{B}_{Mi}^2\end{bmatrix}^\top\right) - 2\boldsymbol{A}\boldsymbol{B}^\top}\right) \in \mathbb{R}^{N\times M}$$

where $\mathbf{1}_N$ and $\mathbf{1}_M$ are vectors of N and M ones respectively, and the square root is taken element-wise. In practice, we partition the KB into rules that have the same structure and batch-unify goals with all rule heads per partition at the same time on a Graphics Processing Unit (GPU). Furthermore, substitution sets bind variables to vectors of symbol indices instead of single symbol indices, and min and max operations are taken per goal.

5.2.3.2 $K \max$ Gradient Approximation

NTPs allow us to calculate the gradient of proof success scores with respect to subsymbolic representations and rule parameters. While backpropagating through this large computation graph will give us the exact gradient, it is computationally infeasible for any reasonably-sized KB. Consider the parameterized rule $\boldsymbol{\theta}_{1:}(\mathrm{X}, \mathrm{Y}) \coloneq \boldsymbol{\theta}_{2:}(\mathrm{X}, \mathrm{Z}), \boldsymbol{\theta}_{3:}(\mathrm{Z}, \mathrm{Y})$ and let us assume the given KB contains $1\ 000$ facts with binary predicates. While X and Y will be bound to the respective representations in the goal, Z we will be substituted with every possible second argument of the $1\ 000$ facts in the KB when proving the first atom in the body. Moreover, for each of these $1\ 000$ substitutions, we will again need to compare with all facts in the KB when proving the second atom in the body of the rule, resulting in $1\ 000\ 000$ proof success scores. However, note that since we use the max operator for aggregating the success of different proofs, only subsymbolic representations in one out of $1\ 000\ 000$ proofs will receive gradients.

To overcome this computational limitation, we propose the following heuristic. We assume that when unifying the first atom with facts in the KB, it is unlikely for any unification successes below the top K successes to attain the maximum proof success when unifying the remaining atoms in the body of a rule with facts in the KB. That is, after the unification of the first atom, we only keep the top K substitutions and their success scores, and continue proving only with these. This means that all other partial proofs will not contribute to the forward pass at this stage, and consequently not receive any gradients on the backward pass of backpropagation. We term this the $K \max$ heuristic. Note that we cannot guarantee anymore that the gradient of the proof success is the exact gradient, but for a large enough K we get a close approximation to the true gradient.

5.3 Experiments

Consistent with previous work, we carry out experiments on four benchmark KBs and compare ComplEx with the NTP and NTPλ in terms of area under the Precision-Recall-curve (AUC-PR) on the Countries KB, and Mean Reciprocal Rank (MRR) and HITS@m [Bordes et al., 2013] on the other KBs described below. Training details, including hyperparameters and rule templates, can be found in Section 5.3.1.

Countries The Countries KB is a dataset introduced by Bouchard et al. [2015] for testing reasoning capabilities of neural link prediction models. It consists of 244 countries, 5 regions (*e.g.* EUROPE), 23 subregions (*e.g.* WESTERN EUROPE, NORTHERN AMERICA), and 1158 facts about the neighborhood of countries, and

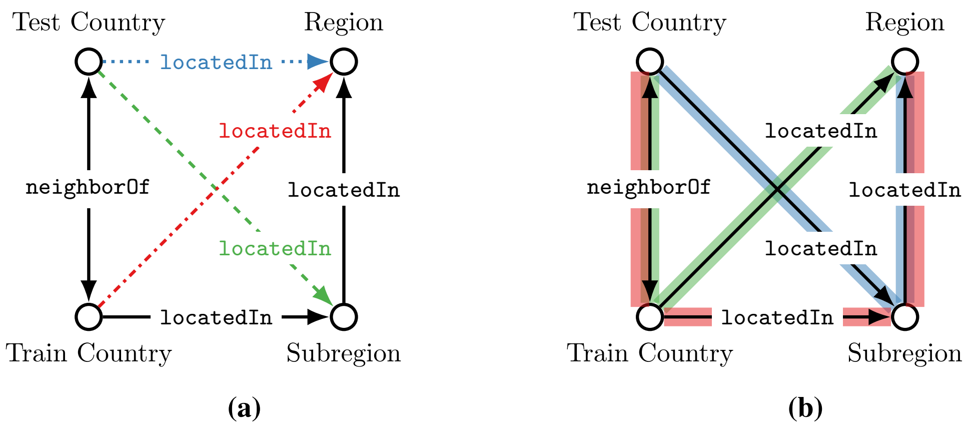


Figure 5.3: Overview of different tasks in the Contries dataset as visualized by Nickel et al. [2016]. The left part (a) shows which atoms are removed for each task (dotted lines), and the right part (b) illustrates the rules that can be used to infer the location of test countries. For task **S1**, only facts corresponding to the blue dotted line are removed from the training set. For task **S2**, additionally facts corresponding to the green dashed line are removed. Finally, for task **S3** also facts for the red dash-dotted line are removed.

the location of countries and subregions. We follow Nickel et al. [2016] and split countries randomly into a training set of 204 countries (train), a development set of 20 countries (dev), and a test set of 20 countries (test), such that every dev and test country has at least one neighbor in the training set. Subsequently, three different task datasets are created. For all tasks, the goal is to predict $\texttt{locatedIn}(c, r)$ for every test country c and all five regions r, but the access to training atoms in the KB varies (see Fig. 5.3).

S1: All ground atoms $\texttt{locatedIn}(c, r)$ where c is a test country and r is a region are removed from the KB. Since information about the subregion of test countries is still contained in the KB, this task can be solved by using the transitivity rule

$$\texttt{locatedIn}(\mathrm{X}, \mathrm{Y}) :\!- \texttt{locatedIn}(\mathrm{X}, \mathrm{Z}), \texttt{locatedIn}(\mathrm{Z}, \mathrm{Y}).$$

S2: In addition to **S1**, all ground atoms $\texttt{locatedIn}(c, s)$ are removed where c is a test country and s is a subregion. The location of test countries needs to be inferred from the location of its neighboring countries:

$$\texttt{locatedIn}(\mathrm{X}, \mathrm{Y}) :\!- \texttt{neighborOf}(\mathrm{X}, \mathrm{Z}), \texttt{locatedIn}(\mathrm{Z}, \mathrm{Y}).$$

This task is more difficult than **S1**, as neighboring countries might not be in the same region, so the rule above will not always hold.

S3: In addition to **S2**, all ground atoms $\texttt{locatedIn}(c, r)$ where r is a region and c

is a training country that has a test or dev country as a neighbor are also removed. The location of test countries can for instance be inferred using the three-hop rule

$$\texttt{locatedIn}(X, Y) \coloneq \texttt{neighborOf}(X, Z), \texttt{neighborOf}(Z, W), \texttt{locatedIn}(W, Y).$$

Kinship, Nations & UMLS We use the Nations, Alyawarra kinship (Kinship) and Unified Medical Language System (UMLS) KBs from Kok and Domingos [2007]. We left out the Animals dataset as it only contains unary predicates and can thus not be used for evaluating multi-hop reasoning. Nations contains 56 binary predicates, 111 unary predicates, 14 constants and 2565 true facts, Kinship contains 26 predicates, 104 constants and 10686 true facts, and UMLS contains 49 predicates, 135 constants and 6529 true facts. Since our baseline ComplEx cannot deal with unary predicates, we remove unary atoms from Nations. We split every KB into 80% training facts, 10% development facts and 10% test facts. For evaluation, we take a test fact and corrupt its first and second argument in all possible ways such that the corrupted fact is not in the original KB. Subsequently, we predict a ranking of every test fact and its corruptions to calculate MRR and HITS@m.

5.3.1 Training Details

We use ADAM [Kingma and Ba, 2015] with an initial learning rate of 0.001 and a mini-batch size of 50 (10 known and 40 corrupted atoms) for optimization. We apply an ℓ_2 regularization of 0.01 to all model parameters, and clip gradient values at $[-1.0, 1.0]$. All subsymbolic representations and rule parameters are initialized using Xavier initialization [Glorot and Bengio, 2010]. We train all models for 100 epochs and repeat every experiment on the Countries corpus ten times. Statistical significance is tested using the independent t-test. All models are implemented in TensorFlow [Abadi et al., 2016]. We use a maximum proof depth of $d = 2$ and add the following rule templates where the number in front of the rule template indicates how often a parameterized rule of the given structure will be instantiated. Note that a rule template such as $\#1(X, Y) \coloneq \#2(X, Z), \#2(Z, Y)$ specifies that the two predicate representations in the body are shared.

Countries S1
3 $\#1(X, Y) \coloneq \#1(Y, X).$
3 $\#1(X, Y) \coloneq \#2(X, Z), \#2(Z, Y).$

Countries S2
3 $\#1(X, Y) \coloneq \#1(Y, X).$
3 $\#1(X, Y) \coloneq \#2(X, Z), \#2(Z, Y).$

Table 5.1: AUC-PR results on Countries and MRR and HITS@m on Kinship, Nations, and UMLS.

Corpus		Metric	Model			Examples of induced rules and their confidence
			ComplEx	**NTP**	**NTPλ**	
Countries	S1	AUC-PR	99.37 ± 0.4	90.83 ± 15.4	**100.00** ± 0.0	0.90 `locatedIn`(X,Y) :– `locatedIn`(X,Z), `locatedIn`(Z,Y).
	S2	AUC-PR	87.95 ± 2.8	87.40 ± 11.7	**93.04** ± 0.4	0.63 `locatedIn`(X,Y) :– `neighborOf`(X,Z), `locatedIn`(Z,Y).
	S3	AUC-PR	48.44 ± 6.3	56.68 ± 17.6	**77.26** ± 17.0	0.32 `locatedIn`(X,Y) :– `neighborOf`(X,Z), `neighborOf`(Z,W), `locatedIn`(W,Y).
Kinship		MRR	**0.81**	0.60	0.80	0.98 `term15`(X,Y) :– `term5`(Y,X)
		HITS@1	0.70	0.48	**0.76**	0.97 `term18`(X,Y) :– `term18`(Y,X)
		HITS@3	**0.89**	0.70	0.82	0.86 `term4`(X,Y) :– `term4`(Y,X)
		HITS@10	**0.98**	0.78	0.89	0.73 `term12`(X,Y) :– `term10`(X, Z), `term12`(Z, Y).
Nations		MRR	**0.75**	**0.75**	0.74	0.68 `blockpositionindex`(X,Y) :– `blockpositionindex`(Y,X).
		HITS@1	**0.62**	**0.62**	0.59	0.46 `expeldiplomats`(X,Y) :– `negativebehavior`(X,Y).
		HITS@3	0.84	0.86	**0.89**	0.38 `negativecomm`(X,Y) :– `commonbloc0`(X,Y).
		HITS@10	**0.99**	**0.99**	**0.99**	0.38 `intergovorgs3`(X,Y) :– `intergovorgs`(Y,X).
UMLS		MRR	0.89	0.88	**0.93**	0.88 `interacts_with`(X,Y) :– `interacts_with`(X,Z), `interacts_with`(Z,Y).
		HITS@1	0.82	0.82	**0.87**	
		HITS@3	0.96	0.92	**0.98**	0.77 `isa`(X,Y) :– `isa`(X,Z), `isa`(Z,Y).
		HITS@10	**1.00**	0.97	**1.00**	0.71 `derivative_of`(X,Y) :– `derivative_of`(X,Z), `derivative_of`(Z,Y).

3 $\#1(\mathrm{X}, \mathrm{Y})$:– $\#2(\mathrm{X}, \mathrm{Z}), \#3(\mathrm{Z}, \mathrm{Y})$.

Countries S3

3 $\#1(\mathrm{X}, \mathrm{Y})$:– $\#1(\mathrm{Y}, \mathrm{X})$.
3 $\#1(\mathrm{X}, \mathrm{Y})$:– $\#2(\mathrm{X}, \mathrm{Z}), \#2(\mathrm{Z}, \mathrm{Y})$.
3 $\#1(\mathrm{X}, \mathrm{Y})$:– $\#2(\mathrm{X}, \mathrm{Z}), \#3(\mathrm{Z}, \mathrm{Y})$.
3 $\#1(\mathrm{X}, \mathrm{Y})$:– $\#2(\mathrm{X}, \mathrm{Z}), \#3(\mathrm{Z}, \mathrm{W}), \#4(\mathrm{W}, \mathrm{Y})$.

Kinship, Nations & UMLS

20 $\#1(\mathrm{X}, \mathrm{Y})$:– $\#2(\mathrm{X}, \mathrm{Y})$.
20 $\#1(\mathrm{X}, \mathrm{Y})$:– $\#2(\mathrm{Y}, \mathrm{X})$.
20 $\#1(\mathrm{X}, \mathrm{Y})$:– $\#2(\mathrm{X}, \mathrm{Z}), \#3(\mathrm{Z}, \mathrm{Y})$.

5.4 Results and Discussion

Results for the different model variants on the benchmark KBs are shown in Table 6.1. Another method for inducing rules in a differentiable way for automated KB completion has been introduced recently by Yang et al. [2017] and our evaluation setup is equivalent to their Protocol II. However, our neural link prediction baseline, ComplEx, already achieves much higher HITS@10 results (1.00 vs. 0.70 on UMLS and 0.98 vs. 0.73 on Kinship). We thus focus on the comparison of NTPs with ComplEx.

First, we note that vanilla NTPs alone do not work particularly well compared to ComplEx. They only outperform ComplEx on Countries S3 and Nations, but not on Kinship or UMLS. This demonstrates the difficulty of learning subsymbolic representations in a differentiable prover from unification alone, and the need for auxiliary losses. The NTPλ with ComplEx as auxiliary loss outperforms the other models in the majority of tasks. The difference in AUC-PR between ComplEx and NTPλ is significant for all Countries tasks ($p < 0.0001$).

A major advantage of NTPs is that we can inspect induced rules which provide us with an interpretable representation of what the model has learned. The right column in Table 6.1 shows examples of induced rules by NTPλ (note that predicates on Kinship are anonymized). For Countries, the NTP recovered those rules that are needed for solving the three different tasks. On UMLS, the NTP induced transitivity rules. Those relationships are particularly hard to encode by neural link prediction models like ComplEx, as they are optimized to locally predict the score of a fact.

5.5 Related Work

Combining neural and symbolic approaches to relational learning and reasoning has a long tradition and let to various proposed architectures over the past decades (see d'Avila Garcez et al. [2012] for a review). Early proposals for neural-symbolic networks are limited to *propositional rules* (*e.g.*, EBL-ANN [Shavlik and Towell, 1989], KBANN [Towell and Shavlik, 1994] and C-IL^{2}P [d'Avila Garcez and Zaverucha, 1999]). Other neural-symbolic approaches focus on first-order inference, but do not learn subsymbolic vector representations from training facts in a KB (*e.g.*, SHRUTI [Shastri, 1992], Neural Prolog [Ding, 1995], CLIP++ [França et al., 2014], Lifted Relational Neural Networks [Sourek et al., 2015], and TensorLog [Cohen, 2016]). Logic Tensor Networks [Serafini and d'Avila Garcez, 2016] are in spirit similar to NTPs, but need to fully ground first-order logic rules. However, they support function terms, whereas NTPs currently only support function-free terms.

Recent question-answering architectures such as [Peng et al., 2015, Weissenborn, 2016, Shen et al., 2016] translate query representations implicitly in a vector space without explicit rule representations and can thus not easily incorporate domain-specific knowledge. In addition, NTPs are related to random walk [Lao et al., 2011, 2012, Gardner et al., 2013, 2014] and path encoding models [Neelakantan et al., 2015, Das et al., 2017]. However, instead of aggregating paths from random walks or encoding paths to predict a target predicate, reasoning steps in NTPs are explicit and only unification uses subsymbolic representations. This allows us to induce interpretable rules, as well as to incorporate prior knowledge either in the form of rules or in the form of rule templates which define the structure of logical relationships that we expect to hold in a KB. Another line of work [Rocktäschel et al., 2014, Rocktäschel et al., 2015, Vendrov et al., 2016, Hu et al., 2016, Demeester et al., 2016] regularizes distributed representations via domain-specific rules, but these approaches do not learn such rules from data and only support a restricted subset of first-order logic. NTPs are constructed from Prolog's backward chaining and are thus related to Unification Neural Networks [Komendantskaya, 2011, Hölldobler, 1990]. However, NTPs operate on vector representations of symbols instead of scalar values, which are more expressive.

As NTPs can learn rules from data, they are related to ILP systems such as FOIL [Quinlan, 1990], Sherlock [Schoenmackers et al., 2010b] and meta-interpretive learning of higher-order dyadic Datalog (Metagol) [Muggleton et al., 2015]. While these ILP systems operate on symbols and search over the discrete space of logical rules, NTPs work with subsymbolic representations and induce rules using gradient descent. Recently, Yang et al. [2017] introduced a differentiable rule learning system based on TensorLog and a neural network controller similar to LSTMs [Hochreiter and Schmidhuber, 1997]. Their method is more scalable than the NTPs introduced here. However, on UMLS and Kinship our baseline already achieved stronger generalization by learning subsymbolic representations. Still, scaling NTPs to larger KBs for competing with more scalable relational learning methods is an open problem that we seek to address in future work.

5.6 Summary

We proposed an end-to-end differentiable prover for automated KB completion that operates on subsymbolic representations. To this end, we used Prolog's backward chaining algorithm as a recipe for recursively constructing neural networks that can be used to prove queries to a KB. Specifically, our contribution is the use of a differentiable unification operation between vector representations of symbols to construct such neural networks. This allowed us to compute the gradient of proof successes with respect to vector representations of symbols, and thus enabled us to train subsymbolic representations end-to-end from facts in a KB, and to induce function-free first-order logic rules using gradient descent. On benchmark KBs, our model outperformed ComplEx, a state-of-the-art neural link prediction model, on three out of four KBs while at the same time inducing interpretable rules.

Chapter 6

Recognizing Textual Entailment with Recurrent Neural Networks

"*You can't cram the meaning of a whole %&!$# sentence into a single $&!#* vector!*"

— Raymond J. Mooney

The ability to determine the logical relationship between two natural language sentences is an integral part for machines that are supposed to understand and reason with language. In previous chapters, we have discussed ways of combining symbolic logical knowledge with subsymbolic representations trained via neural networks. As first steps towards models that reason with natural language, we have used textual surface form patterns as predicates for automated Knowledge Base (KB) completion. However, for automated KB completion, we assumed surface patterns to be atomic, which does not generalize to unseen patterns as there is no compositional representation. In this chapter, we are using Recurrent Neural Networks (RNNs) for learning compositional representations of natural language sentences. Specifically, we are tackling the task of Recognizing Textual Entailment (RTE), *i.e.*, determining (i) whether two natural language sentences are contradicting each other, (ii) whether they are unrelated, or (iii) whether the first sentence (called the *premise*) entails the second sentence (called the *hypothesis*). For instance, the sentence "Two girls and a guy are involved in a pie eating contest" entails "Three people are stuffing their faces", but contradicts "Three people are drinking beer on a boat". This task is important since many Natural Language Processing (NLP) tasks, such as information extraction, relation extraction, text summarization or machine translation, rely on it explicitly or implicitly and could benefit from more accurate RTE systems [Dagan et al., 2005].

State-of-the-art systems for RTE so far relied heavily on engineered NLP pipelines, extensive manual creation of features, as well as various external resources and specialized subcomponents such as negation detection [*e.g.* Lai and Hockenmaier, 2014, Jiménez et al., 2014, Zhao et al., 2014, Beltagy et al., 2015]. Despite the success of neural networks for paraphrase detection [*e.g.* Socher et al., 2011, Hu et al., 2014, Yin and Schütze, 2015], end-to-end differentiable neural architectures so far failed to reach acceptable performance for RTE due to the lack of large high-quality datasets. An end-to-end differentiable solution to RTE is desirable since it avoids specific assumptions about the underlying language. In particular, there is no need for language features like part-of-speech tags or dependency parses. Furthermore, a generic sequence-to-sequence solution allows us to extend the concept of capturing entailment across any sequential data, not only natural language.

Recently, Bowman et al. [2015] published the Stanford Natural Language Inference (SNLI) corpus accompanied by a long short-term memory (LSTM) baseline [Hochreiter and Schmidhuber, 1997] which achieves an accuracy of 77.6% for RTE on this dataset. It is the first instance of a generic neural model without hand-crafted features that got close to the accuracy of a simple lexicalized classifier with engineered features for RTE. This can be explained by the high quality and size of SNLI compared to the two orders of magnitude smaller and partly synthetic datasets used so far to evaluate RTE systems. Bowman et al.'s LSTM encodes the premise and hypothesis independently as dense fixed-length vectors whose concatenation is subsequently used in a Multi-layer Perceptron (MLP) for classification (Section 6.1.2). In contrast, we are proposing a neural network that is capable of fine-grained comparison of pairs of words and phrases by processing the hypothesis *conditioned on the premise* and using a neural attention mechanism.

Our contributions are threefold: (i) we present a neural model based on two LSTMs that read two sentences in one go to determine entailment as opposed to mapping each sentence independently into a vector space (Section 6.2.1), (ii) we extend this model with a neural word-by-word attention mechanism to encourage more fine-grained comparison of pairs of words and phrases (Section 6.2.3), and (iii) we provide a detailed qualitative analysis of neural attention for RTE (Section 6.4.1). Our benchmark LSTM achieves an accuracy of 80.9% on SNLI, outperforming a simple lexicalized classifier tailored to RTE by 2.7 percentage points. An extension with word-by-word neural attention surpasses this strong benchmark LSTM result by 2.6 percentage points, achieving an accuracy of 83.5% for recognizing entailment on SNLI.

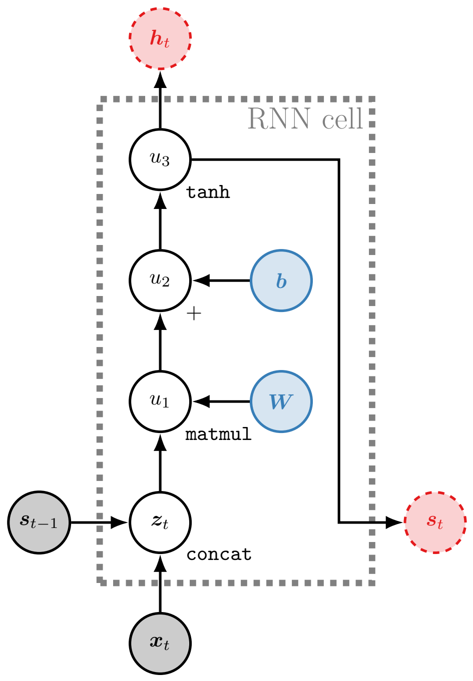


Figure 6.1: Computation graph for the fully-connected RNN cell.

6.1 Background

In this section, we describe RNNs and LSTMs for sequence modeling, before explaining how LSTMs are used in the independent sentence encoding model for RTE by Bowman et al. [2015].

6.1.1 Recurrent Neural Networks

An RNN is parameterized by a differentiable *cell function* $f_\theta : \mathbb{R}^k \times \mathbb{R}^s \rightarrow \mathbb{R}^o \times \mathbb{R}^s$ that maps an input vector $\boldsymbol{x}_t \in \mathbb{R}^k$ and previous state $\boldsymbol{s}_{t-1} \in \mathbb{R}^s$ to an output vector $\boldsymbol{h}_t \in \mathbb{R}^o$ and next state $\boldsymbol{s}_t \in \mathbb{R}^s$. For simplicity, we assume that the input size and output size are the same, *i.e.*, $\boldsymbol{x}_t, \boldsymbol{h}_t \in \mathbb{R}^k$. By applying the cell function at time-step t, we obtain an output vector $\boldsymbol{h}_t$ and the next state $\boldsymbol{s}_t$:

$$\boldsymbol{h}_t, \boldsymbol{s}_t = f_\theta(\boldsymbol{x}_t, \boldsymbol{s}_{t-1}). \tag{6.1}$$

Given a sequence of input representations $\mathrm{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T]$ and a start state $\boldsymbol{s}_0$, the output of the RNN over the entire input sequence is then obtained by recursively applying the cell function:

$$\mathrm{RNN}(f_\theta, \mathrm{X}, \boldsymbol{s}_0) = [f_\theta(\boldsymbol{x}_1, \boldsymbol{s}_0), \ldots, f_\theta(\boldsymbol{x}_T, \boldsymbol{s}_{T-1})]. \tag{6.2}$$

Usually, this output is separated into a list of output vectors $\mathrm{H} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_T]$ and states $\mathrm{S} = [\boldsymbol{s}_1, \ldots, \boldsymbol{s}_T]$. Note that RNN can be applied to input sequences of varying length, such as sequences of word representations.

6.1.1.1 Fully-connected Recurrent Neural Network

The most basic RNN is parameterized by the following cell function

$$\boldsymbol{z}_t = \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{s}_{t-1} \end{bmatrix} \tag{6.3}$$

$$\boldsymbol{h}_t = \tanh(\boldsymbol{W}\boldsymbol{z}_t + \boldsymbol{b}) \tag{6.4}$$

$$\boldsymbol{s}_t = \boldsymbol{h}_t \tag{6.5}$$

$$f_\theta^{\mathrm{RNN}}(\boldsymbol{x}_t, \boldsymbol{s}_{t-1}) = (\boldsymbol{h}_t, \boldsymbol{s}_t) \tag{6.6}$$

where $\boldsymbol{W} \in \mathbb{R}^{2k \times k}$ is a trainable transition matrix, $\boldsymbol{b} \in \mathbb{R}^k$ a trainable bias, and $\tanh$ the element-wise application of the hyperbolic function. We call this a fully-connected RNN, as the cell function is modeled by a dense layer. We illustrate the computation graph for a single application of the fully-connected RNN cell in Fig. 6.1. Note that for recurrent application of this cell function to a sequence of inputs, all parameters (transition matrix and bias) are shared between all time steps.

6.1.1.2 Long Short-Term Memory

RNNs with LSTM units [Hochreiter and Schmidhuber, 1997] have been successfully applied to a wide range of NLP tasks, such as machine translation [Sutskever et al., 2014], constituency parsing [Vinyals et al., 2015b], language modeling [Zaremba et al., 2014] and recently RTE [Bowman et al., 2015]. LSTMs encompass memory cells that can store information for a long period of time, as well as three types of gates that control the flow of information into and out of these cells: *input gates* (Eq. 6.8), *forget gates* (Eq. 6.9) and *output gates* (Eq. 6.10). Given an input vector

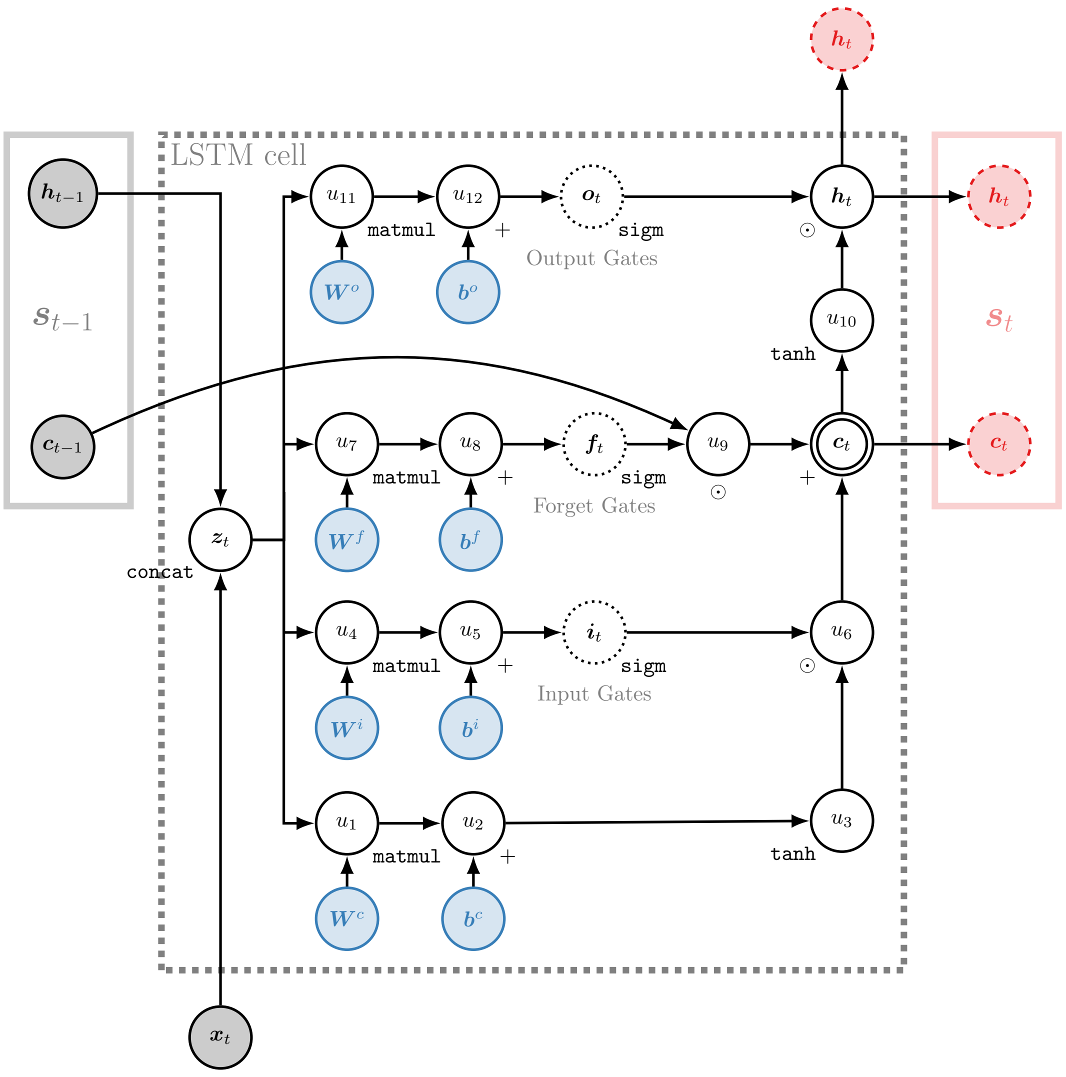


Figure 6.2: Computation graph for the LSTM cell.

$\mathbf{x}_t$ at time step t, the previous output $\mathbf{h}_{t-1}$ and cell state $\mathbf{c}_{t-1}$, an LSTM with hidden size k computes the next output $\mathbf{h}_t$ and cell state $\mathbf{c}_t$ as

$$\boldsymbol{z}_t = \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{h}_{t-1} \end{bmatrix} \tag{6.7}$$

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}^i \boldsymbol{z}_t + \boldsymbol{b}^i) \tag{6.8}$$

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}^f \boldsymbol{z}_t + \boldsymbol{b}^f) \tag{6.9}$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}^o \boldsymbol{z}_t + \boldsymbol{b}^o) \tag{6.10}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tanh(\boldsymbol{W}^c \boldsymbol{z}_t + \boldsymbol{b}^c) \tag{6.11}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t) \tag{6.12}$$

$$\boldsymbol{s}_t = \begin{bmatrix} \boldsymbol{h}_t \\ \boldsymbol{c}_t \end{bmatrix} \tag{6.13}$$

$$f_\theta^{\mathtt{LSTM}}(\boldsymbol{x}_t, \boldsymbol{s}_{t-1}) = (\boldsymbol{h}_t, \boldsymbol{s}_t) \tag{6.14}$$

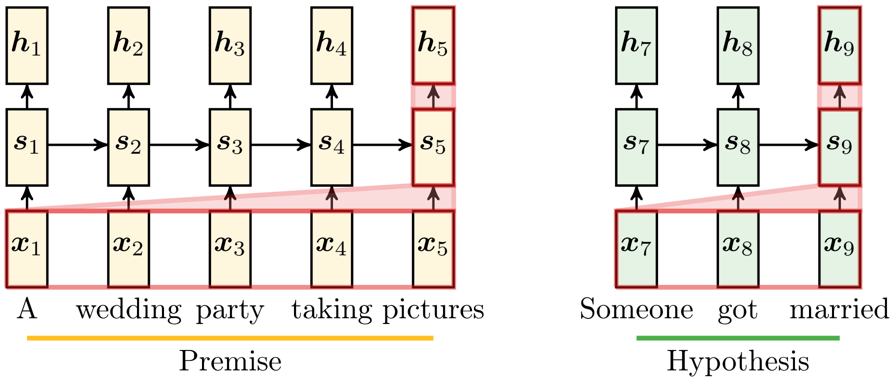


Figure 6.3: Independent encoding of the premise and hypothesis using the same LSTM. Note that both sentences are compressed as dense vectors as indicated by the red mapping.

where $\boldsymbol{W}^i, \boldsymbol{W}^f, \boldsymbol{W}^o, \boldsymbol{W}^c \in \mathbb{R}^{2k \times k}$ are trained matrices and $\boldsymbol{b}^i, \boldsymbol{b}^f, \boldsymbol{b}^o, \boldsymbol{b}^c \in \mathbb{R}^k$ trained biases that parameterize the gates and transformations of the input. As in previous chapters, σ denotes the element-wise application of the sigmoid function and $\odot$ the element-wise multiplication of two vectors. The corresponding computation graph is illustrated in Fig. 6.2.

6.1.2 Independent Sentence Encoding

LSTMs can readily be used for RTE by independently encoding the premise and hypothesis as dense vectors and taking their concatenation as input to an MLP classifier [Bowman et al., 2015].

Let $\mathrm{X}^P = [\boldsymbol{x}_1^P, \ldots, \boldsymbol{x}_N^P]$ be a sequence of words representing the premise and let $\mathrm{X}^H = [\boldsymbol{x}_1^H, \ldots, \boldsymbol{x}_M^H]$ represent the hypothesis. Both, the premise and the hypothesis, can be encoded as fixed-dimensional vectors by taking the last output vector when applying the RNN function (Eq. 6.2) with an LSTM cell function $f_\theta^{\mathtt{LSTM}}$. Subsequently, the prediction for the three RTE classes is obtained by an MLP (Eqs. 6.17 and 6.18) followed by a softmax (Eq. 6.19):

$$\mathrm{H}^P, \mathrm{S}^P = \mathrm{RNN}(f_\theta^{\mathtt{LSTM}}, \mathrm{X}^P, \boldsymbol{s}_0) \tag{6.15}$$

$$\mathrm{H}^H, \mathrm{S}^H = \mathrm{RNN}(f_\theta^{\mathtt{LSTM}}, \mathrm{X}^H, \boldsymbol{s}_0) \tag{6.16}$$

$$\boldsymbol{h} = \tanh\left(\boldsymbol{W}_2 \tanh\left(\boldsymbol{W}_1 \begin{bmatrix} \boldsymbol{h}_N^P \\ \boldsymbol{h}_M^H \end{bmatrix} + \boldsymbol{b}_1\right) + \boldsymbol{b}_2\right) \tag{6.17}$$

$$\boldsymbol{h}^* = \tanh\left(\boldsymbol{W}_3 \boldsymbol{h} + \boldsymbol{b}_3\right) \tag{6.18}$$

$$\boldsymbol{y}_i = \mathrm{softmax}(\boldsymbol{h}^*)_i = \frac{e^{\boldsymbol{h}_i^*}}{\sum_j e^{\boldsymbol{h}_j^*}} \tag{6.19}$$

where $\boldsymbol{W}_1, \boldsymbol{W}_2 \in \mathbb{R}^{2k\times 2k}$, $\boldsymbol{W}_3 \in \mathbb{R}^{2k\times 3}$, $\boldsymbol{b}_1, \boldsymbol{b}_2 \in \mathbb{R}^{2k}$ and $\boldsymbol{b}_3 \in \mathbb{R}^3$. Furthermore, $\boldsymbol{s}_0$ is a trainable start state and $\boldsymbol{h}^P_N$ denotes the last element from the list of premise output vectors (similarly for $\boldsymbol{h}^H_M$). The independent sentence encoding (Eqs. 6.15 and 6.16) is visualized in Fig. 6.3.

Given the predicted output distribution over the three RTE classes $\boldsymbol{y}$ (ENTAILMENT, NEUTRAL or CONTRADICTION) and a target one-hot vector $\hat{\boldsymbol{y}}$ encoding the correct class, a cross-entropy loss is commonly used:

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_i \hat{\boldsymbol{y}}_i \log(\boldsymbol{y}_i). \tag{6.20}$$

Independent sentence encoding is a straightforward model for RTE. However, it is questionable how efficiently an entire sentence can be represented in a single fixed-dimensional vector. Hence, in the next section, we investigate various neural architectures that are tailored towards more fine-grained comparison of the premise and hypothesis and thus do not require to represent entire sentences as fixed-sized vectors in an embedding space.

6.2 Methods

First, we propose to encode the hypothesis conditioned on a representation of the premise (Section 6.2.1). Subsequently, we introduce an extension of an LSTM for RTE with neural attention (Section 6.2.2) and word-by-word attention (Section 6.2.3). Finally, we show how such attentive models can easily be used for attending both ways: over the premise conditioned on the hypothesis and over the hypothesis conditioned on the premise (Section 6.2.4). All these models are trained using the loss in Eq. 6.20, and predict the probability of the RTE class using Eq. 6.19, but they differ in the way $\boldsymbol{h}^*$ (Eq. 6.18) is calculated.

6.2.1 Conditional Encoding

In contrast to learning sentence representations, we are interested in neural models that read both sentences to determine entailment, thereby comparing pairs of words and phrases. Figure 6.4 shows the high-level structure of this model. The premise (left) is read by an LSTM. A second LSTM with different parameters is reading a delimiter and the hypothesis (right), but its memory state is initialized with the last state of the previous LSTM ($\boldsymbol{s}_5$ in the example). That is, it processes the hypothesis

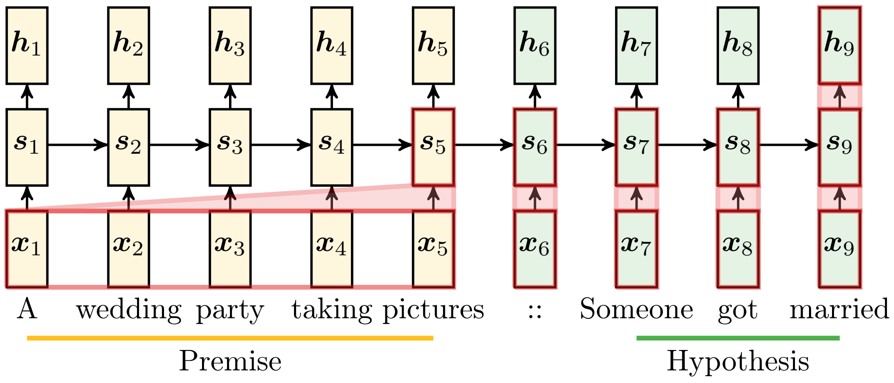

h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9
s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9
A wedding party taking pictures :: Someone got married
Premise
Hypothesis

Figure 6.4: Conditional encoding with two LSTMs. The first LSTM encodes the premise, and then the second LSTM processes the hypothesis conditioned on the representation of the premise ($\boldsymbol{s}_5$).

conditioned on the representation that the first LSTM built for the premise. Formally, we replace Eqs. 6.15 to 6.17 with

$$\mathrm{H}^P, \mathrm{S}^P = \mathrm{RNN}(f^{\mathrm{LSTM}}_{\theta_P}, \mathrm{X}^P, \boldsymbol{s}_0) \tag{6.21}$$

$$\mathrm{H}^H, \mathrm{S}^H = \mathrm{RNN}(f^{\mathrm{LSTM}}_{\theta_H}, \mathrm{X}^H, \boldsymbol{s}^P_N) \tag{6.22}$$

$$\boldsymbol{h} = \boldsymbol{h}^H_M \tag{6.23}$$

where $\boldsymbol{s}^P_N$ denotes the last state of the LSTM that encoded the premise and $\boldsymbol{h}^P_M$ denotes the last element from the list of hypothesis output vectors.

This model is illustrated for an RTE example in Fig. 6.4. Note that while the premise still has to be encoded in a fixed-dimensional vector, the LSTM that processes the hypothesis has to only keep track of whether incoming words contradict the premise, whether they are entailed by it, or whether they are unrelated. This is inspired by finite-state automata proposed for natural logic inference [Angeli and Manning, 2014].

6.2.2 Attention

Attentive neural networks have recently demonstrated success in a wide range of tasks ranging from handwriting synthesis [Graves, 2013], digit classification [Mnih et al., 2014], machine translation [Bahdanau et al., 2014], image captioning [Xu et al., 2015], speech recognition [Chorowski et al., 2015], sentence summarization [Rush et al., 2015], and code summarization [Allamanis et al., 2016] to geometric reasoning [Vinyals et al., 2015a]. The idea is to allow the model to attend over past output vectors. For LSTMs this mitigates the cell state bottleneck, *i.e.*, the fact that

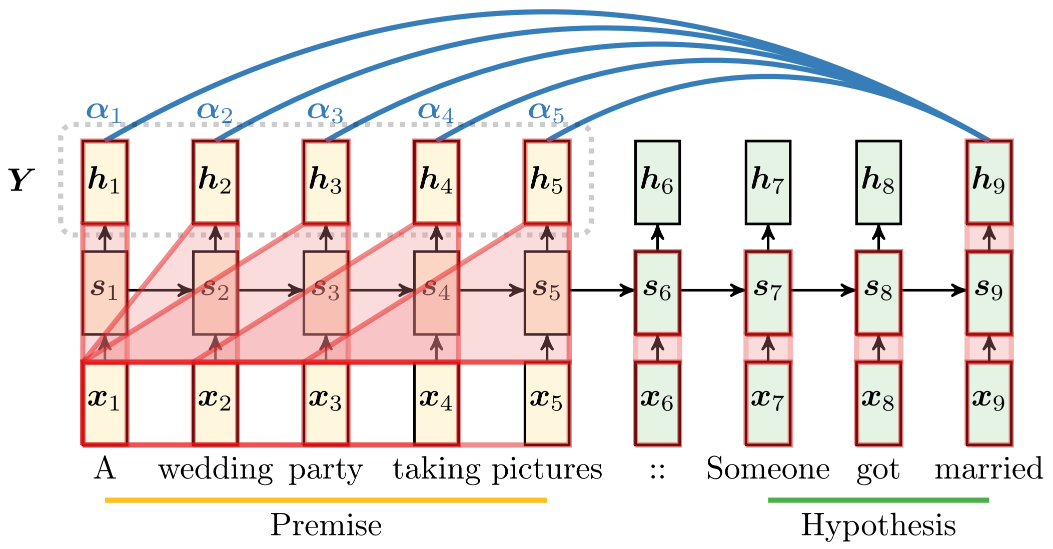

α_1
α_2
α_3
α_4
α_5
Y
h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9
s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9
A wedding party taking pictures :: Someone got married
Premise
Hypothesis

Figure 6.5: Attention model for RTE. Compared to Fig. 6.4, this model does not have to represent the entire premise in its cell state, but can instead output context representations (informally visualized by the red mapping) that are later queried by the attention mechanism (blue). Also note that now $\boldsymbol{h}_1$ to $\boldsymbol{h}_5$ are used.

a standard LSTM has to store all relevant information for future time steps in its internal memory $\boldsymbol{c}_t$ (see $\boldsymbol{c}_t$ in Fig. 6.2 and compare Fig. 6.4 with Fig. 6.5).

An LSTM with attention for RTE does not have to capture the entire content of the premise in its cell state. Instead, it is sufficient to output vectors while reading the premise (*i.e.* populating a differentiable memory of the premise) and accumulating a representation in the cell state that informs the second LSTM which of the output vectors of the premise it needs to attend over to determine the RTE class.

Formally, let $\boldsymbol{Y} \in \mathbb{R}^{k\times N}$ be a matrix consisting of output vectors $[\boldsymbol{h}_1 \cdots \boldsymbol{h}_N]$ that the first LSTM produced when reading the N words of the premise. Furthermore, let $\mathbf{1} \in \mathbb{R}^N$ be a vector of ones and $\boldsymbol{h}_M^H$ be the last output vector after the premise and hypothesis were processed by the two LSTMs. The attention mechanism will produce a vector $\boldsymbol{\alpha} \in \mathbb{R}^N$ of attention weights and a weighted representation $\boldsymbol{r}$ of the premise via

$$\boldsymbol{M} = \tanh\left(\boldsymbol{W}^y\boldsymbol{Y} + \left(\boldsymbol{W}^h\boldsymbol{h}_M^H\right)\mathbf{1}^T\right) \qquad \boldsymbol{M} \in \mathbb{R}^{k\times N} \tag{6.24}$$

$$\boldsymbol{\alpha} = \text{softmax}(\boldsymbol{w}^T\boldsymbol{M}) \qquad \boldsymbol{\alpha} \in \mathbb{R}^{1\times N} \tag{6.25}$$

$$\boldsymbol{r} = \boldsymbol{Y}\boldsymbol{\alpha}^T \qquad \boldsymbol{r} \in \mathbb{R}^k \tag{6.26}$$

where $\boldsymbol{W}^y, \boldsymbol{W}^h \in \mathbb{R}^{k\times k}$ are trainable projection matrices, and $\boldsymbol{w} \in \mathbb{R}^k$ is a trainable parameter vector. Note that the outer product $(\boldsymbol{W}^h\boldsymbol{h}_M^H)\mathbf{1}^\top$ is repeating the linearly transformed $\boldsymbol{h}_M^H$ as many times as there are words in the premise (*i.e.* N times). Hence, the intermediate attention representation $\boldsymbol{m}_i$ (ith column vector in $\boldsymbol{M}$) of the ith word in the premise is obtained from a non-linear combination of the premise's output vector $\boldsymbol{h}_i$ (ith column vector in $\boldsymbol{Y}$) and the transformed $\boldsymbol{h}_M^H$. The attention weight for the ith word in the premise is the result of a weighted combination (parameterized by $\boldsymbol{w}$) of values in $\boldsymbol{m}_i$.

The final sentence pair representation is obtained from a non-linear combination of the attention-weighted representation $\mathbf{r}$ of the premise and the last output vector $\mathbf{h}_M^H$, thus replacing Eq. 6.23 by

$$\boldsymbol{h} = \tanh(\boldsymbol{W}^p\boldsymbol{r} + \boldsymbol{W}^x\boldsymbol{h}_M^H) \tag{6.27}$$

where $\boldsymbol{W}^p, \boldsymbol{W}^x \in \mathbb{R}^{k\times k}$ are trainable projection matrices.

The attention model is illustrated in Fig. 6.5. Note that this model does not have to represent the entire premise in its cell state, but can instead output context representations that are later queried by the attention mechanism. This is informally illustrated by the red mapping of input phrases to output context representations.[1]

6.2.3 Word-by-word Attention

For determining whether one sentence entails another it is desirable to check for entailment or contradiction of individual word and phrase pairs. To encourage such behavior we employ neural word-by-word attention similar to Bahdanau et al. [2014], Hermann et al. [2015] and Rush et al. [2015]. The difference is that we do not use attention to generate words, but to obtain a sentence pair encoding from fine-grained comparison via soft-alignment of word and phrase pairs in the premise and hypothesis. In our case, this amounts to attending over the first LSTM's output vectors for the premise while the second LSTM processes the hypothesis one word at a time. Consequently, we obtain attention weight-vectors $\boldsymbol{\alpha}_t$ over premise output vectors for every word in the hypothesis. This can be modeled as follows:

$$\boldsymbol{M}_t = \tanh\left(\boldsymbol{W}^y\boldsymbol{Y} + \left(\boldsymbol{W}^h\boldsymbol{h}_t^H + \boldsymbol{W}^r\boldsymbol{r}_{t-1}\right)\mathbf{1}^T\right) \qquad \boldsymbol{M}_t \in \mathbb{R}^{k\times N} \tag{6.28}$$

$$\boldsymbol{\alpha}_t = \operatorname{softmax}\left(\boldsymbol{w}^T\boldsymbol{M}_t\right) \qquad \boldsymbol{\alpha}_t \in \mathbb{R}^{1\times N} \tag{6.29}$$

$$\boldsymbol{r}_t = \boldsymbol{Y}\boldsymbol{\alpha}_t^T + \tanh\left(\boldsymbol{W}^t\boldsymbol{r}_{t-1}\right) \qquad \boldsymbol{r}_t \in \mathbb{R}^k \tag{6.30}$$

[1]Note that the number of word representations that are used to output a context representation is not known. For illustration purposes we depicted the case that information of three words contribute to an output context representation.

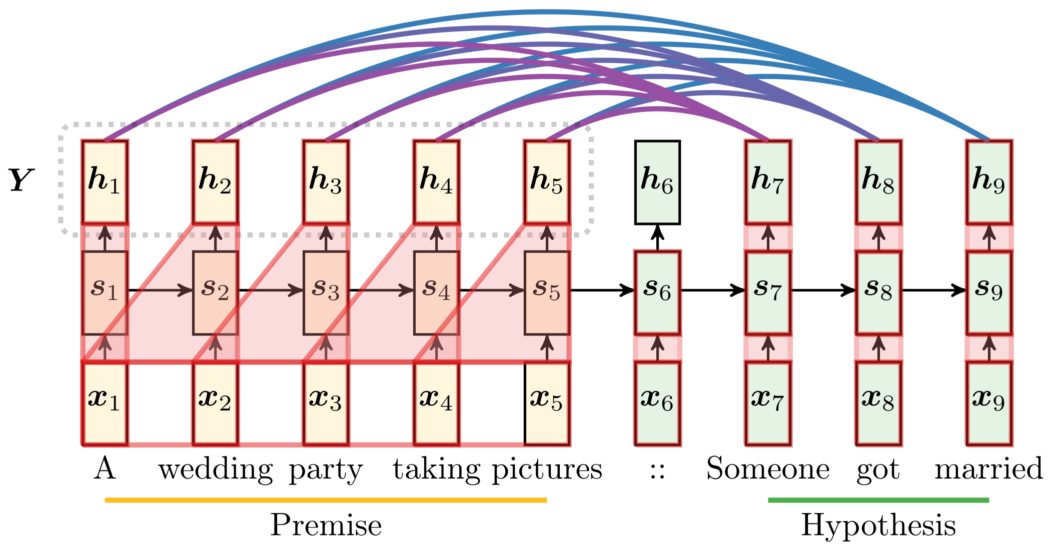


Figure 6.6: Word-by-word attention model for RTE. Compared to Fig. 6.5, querying the memory $\boldsymbol{Y}$ multiple times allows the model to store more fine-grained information in its output vectors when processing the premise. Also note that now also $\boldsymbol{h}_6$ to $\boldsymbol{h}_8$ are used.

where $\boldsymbol{W}^r, \boldsymbol{W}^t \in \mathbb{R}^{k\times k}$ are trainable projection matrices. Note that $\mathbf{r}_t$ is dependent on the previous attention representation $\mathbf{r}_{t-1}$ to inform the model about what was attended over in previous steps (see Eqs. 6.28 and 6.30).

As in the previous section, the final sentence pair representation is obtained from a non-linear combination of the last attention-weighted representation of the premise (here conditioned on the last word of the hypothesis) $\mathbf{r}_N$ and the last output vector using

$$\boldsymbol{h} = \tanh(\boldsymbol{W}^p\boldsymbol{r}_M + \boldsymbol{W}^x\boldsymbol{h}_M^H). \tag{6.31}$$

The word-by-word attention model is illustrated in Fig. 6.6. Compared to the attention model introduced earlier, querying the memory $\boldsymbol{Y}$ multiple times allows the model to store more fine-grained information in its output vectors when processing the premise. We informally illustrate this by fewer words contributing to the output context representations (red mapping).

6.2.4 Two-way Attention

Inspired by bidirectional LSTMs that read a sequence and its reverse for improved encoding [Graves and Schmidhuber, 2005], we experiment with two-way attention for RTE. The idea is to use the same model (*i.e.* same structure and weights) to attend over the premise conditioned on the hypothesis, as well as to attend over the hypothesis conditioned on the premise, by simply swapping the two sequences. This produces two sentence pair representations that we concatenate for classification.

6.3 Experiments

We conduct experiments on the Stanford Natural Language Inference corpus [SNLI, Bowman et al., 2015]. This corpus is two orders of magnitude larger than other existing RTE corpora such as Sentences Involving Compositional Knowledge [SICK, Marelli et al., 2014]. Furthermore, a large part of training examples in SICK were generated heuristically from other examples. In contrast, all sentence pairs in SNLI stem from human annotators. The size and quality of SNLI make it a suitable resource for training neural architectures such as the ones proposed in this chapter.

6.3.1 Training Details

We use pretrained word2vec vectors [Mikolov et al., 2013] as word representations, which we keep fixed during training. Out-of-vocabulary words in the training set are randomly initialized by uniformly sampling values from $[-0.05, 0.05]$ and are optimized during training.[2] Out-of-vocabulary words encountered at inference time on the validation and test corpus are set to fixed random vectors. By not tuning representations of words for which we have word2vec vectors, we ensure that at test time their representation stays close to unseen similar words that are contained in word2vec.

We use ADAM [Kingma and Ba, 2015] for optimization with a first momentum coefficient of 0.9 and a second momentum coefficient of 0.999.[3] For every model we perform a grid search over combinations of the initial learning rate [1E-4, 3E-4, 1E-3], dropout[4] [0.0, 0.1, 0.2] and ℓ_2-regularization strength [0.0, 1E-4, 3E-4, 1E-3]. Subsequently, we take the best configuration based on performance on the validation set, and evaluate only that configuration on the test set.

6.4 Results and Discussion

Results on the SNLI corpus are summarized in Table 6.1. The total number of model parameters, including tunable word representations, is denoted by $|\theta|_{\text{W+M}}$ (without word representations $|\theta|_{\text{M}}$). To ensure a comparable number of parameters to Bowman et al.'s model that encodes the premise and hypothesis independently

[2]We found 12.1k words in SNLI for which we could not obtain word2vec embeddings, resulting in 3.65M tunable parameters.

[3]Standard configuration recommended by Kingma and Ba.

[4]As in Zaremba et al. [2014], we apply dropout only on the inputs and outputs of the network.

Table 6.1: Results on the SNLI corpus.

Model	k	$\|\theta\|_{\text{W+M}}$	$\|\theta\|_{\text{M}}$	Train	Dev	Test
Lexicalized classifier [Bowman et al., 2015]	-	-	-	99.7	-	78.2
LSTM [Bowman et al., 2015]	100	$\approx$ 10M	221k	84.4	-	77.6
Conditional encoding, shared	100	3.8M	111k	83.7	81.9	80.9
Conditional encoding, shared	159	3.9M	252k	84.4	83.0	81.4
Conditional encoding	116	3.9M	252k	83.5	82.1	80.9
Attention	100	3.9M	242k	85.4	83.2	82.3
Attention, two-way	100	3.9M	242k	86.5	83.0	82.4
Word-by-word attention	100	3.9M	252k	85.3	**83.7**	**83.5**
Word-by-word attention, two-way	100	3.9M	252k	86.6	83.6	83.2

using one LSTM, we also run experiments for conditional encoding where the parameters between both LSTMs are shared ("Conditional encoding, shared" with $k = 100$), as opposed to using two independent LSTMs. In addition, we compare our attentive models to two benchmark LSTMs whose hidden sizes were chosen so that they have at least as many parameters as the attentive models (k set to 159 and 116 respectively). Since we are not tuning word vectors for which we have word2vec embeddings, the total number of parameters $|\theta|_{\text{W+M}}$ of our models is considerably smaller. We also compare our models against the benchmark lexicalized classifier used by Bowman et al., which uses features based on the BLEU score between the premise and hypothesis, length difference, word overlap, uni- and bigrams, part-of-speech tags, as well as cross uni- and bigrams.

Conditional Encoding We found that processing the hypothesis conditioned on the premise instead of encoding both sentences independently gives an improvement of 3.3 percentage points in accuracy over Bowman et al.'s LSTM. We argue this is due to information being able to flow from the part of the model that processes the premise to the part that processes the hypothesis. Specifically, the model does not waste capacity on encoding the hypothesis (in fact it does not need to encode the hypothesis at all), but can read the hypothesis in a more focused way by checking words and phrases for contradiction or entailment based on the semantic representation of the premise (see Fig. 6.4). One interpretation is that the LSTM is approximating a finite-state automaton for RTE [*c.f.* Angeli and Manning, 2014]. Another difference to Bowman et al.'s model is that we are using word2vec instead of GloVe for word representations and, more importantly, do not fine-tune these word embeddings. The drop in accuracy from the train to the test set is less severe for our models, which suggest that fine-tuning word embeddings could be a cause of overfitting.

Our LSTM outperforms a simple lexicalized classifier by 2.7 percentage points. To the best of our knowledge, at the time of publication this was the first instance of a neural end-to-end differentiable model outperforming a hand-crafted NLP pipeline on a textual entailment dataset.

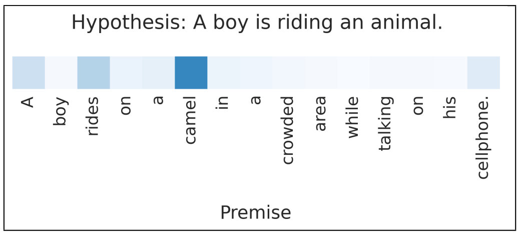
Attention By incorporating an attention mechanism we observe a 0.9 percentage point improvement over a single LSTM with a hidden size of 159 and a 1.4 percentage point increase over a benchmark model that uses two LSTMs for conditional encoding (one for the premise and one for the hypothesis conditioned on the representation of the premise). The attention model produces output vectors summarizing contextual information of the premise that is useful to attend over later when reading the hypothesis. Therefore, when reading the premise, the model does not have to build up a semantic representation of the whole premise, but instead a representation that helps attending over the premise's output vectors when processing the hypothesis (see Fig. 6.5). In contrast, the output vectors of the premise are not used by the baseline conditional model. Thus, these models have to build up a representation of the entire premise and carry it over through the cell state to the part that processes the hypothesis—a bottleneck that can be overcome to some degree by using attention.

Word-by-word Attention Enabling the model to attend over output vectors of the premise for every word in the hypothesis yields another 1.2 percentage point improvement compared to attending only once. We argue that this can be explained by the model being able to check for entailment or contradiction of individual word and phrase pairs, and we demonstrate this effect in the qualitative analysis below.

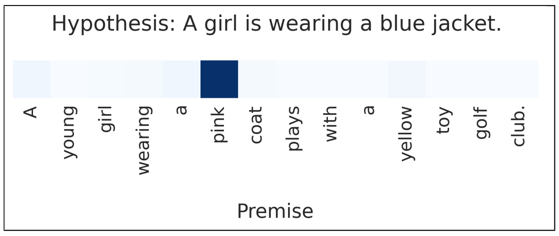
Two-way Attention Allowing the model to also attend over the hypothesis based on the premise did not improve performance for RTE in our experiments. We suspect that this is due to entailment being an asymmetric relation. Hence, using the same LSTM to encode the hypothesis (in one direction) and the premise (in the other direction) might lead to noise in the training signal. This could be addressed by training different LSTMs at the cost of doubling the number of model parameters.

6.4.1 Qualitative Analysis

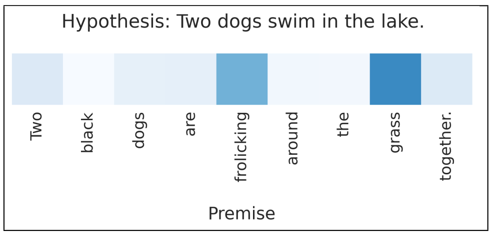
It is instructive to analyze which output representations the model is attending over when deciding the class of an RTE example. Note that interpretations based on attention weights have to be taken with care since the model is not forced to solely rely on representations obtained from attention (see $\boldsymbol{h}_M^H$ in Eqs. 6.27 and 6.31). In the following, we visualize and discuss attention patterns of the presented attentive models. For each attentive model, we hand-picked examples from ten sentence pairs that were randomly drawn from the development set.



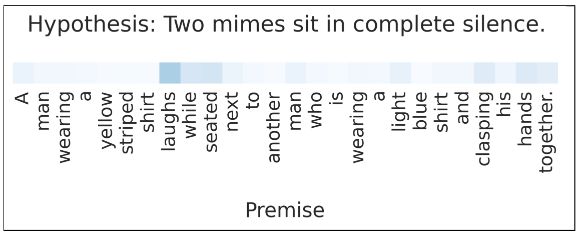
(a)



(b)



(c)



(d)

Figure 6.7: Attention visualizations.

Attention Figure 6.7 shows to what extent the attentive model focuses on contextual representations of the premise after both LSTMs processed the premise and hypothesis, respectively. Note how the model pays attention to output vectors of words that are semantically coherent with the premise ("riding" and "rides", "animal" and "camel", 6.7a) or in contradiction, as caused by a single word ("blue" vs. "pink", 6.7b) or multiple words ("swim" and "lake" vs. "frolicking" and "grass", 6.7c). Interestingly, the model shows sensitivity to context by not attending over "yellow", the color of the toy, but "pink", the color of the coat. However, for more involved examples with longer premises, we found that attention is more uniformly distributed (6.7d). This suggests that conditioning attention only on the last output representation has limitations when multiple words need to be considered for deciding the RTE class.

Word-by-word Attention Visualizations of word-by-word attention are depicted in Fig. 6.8. We found that word-by-word attention can easily detect if the hypothesis is simply a reordering of words in the premise (6.8a). Furthermore, it is able to resolve synonyms ("airplane" and "aircraft", 6.8c) and capable of matching multi-word expressions to single words ("garbage can" to "trashcan", 6.8b). It is also noteworthy that irrelevant parts of the premise, such as words capturing little meaning or whole uninformative relative clauses, are correctly neglected for determining entailment ("which also has a rope leading out of it", 6.8b).

Word-by-word attention seems to also work well when words in the premise and hypothesis are connected via deeper semantics ("snow" can be found "outside" and a “mother” is an “adult”, 6.8e and 6.8g). Furthermore, the model is able to resolve one-to-many relationships (“kids” to “boy” and “girl”, 6.8d).

Attention can fail, for example when the two sentences and their words are entirely unrelated (6.8f). In such cases, the model seems to back off to attending over function words, and the sentence pair representation is likely dominated by the last output vector ($\boldsymbol{h}_M^H$) instead of the attention-weighted representation (see Eq. 6.31).

6.5 Related Work

The methods in this chapter were published in Rocktäschel et al. [2016] and since then many new models have been proposed. They can be roughly classified into sentence encoding models which extend the independent encoding LSTM by Bowman et al. [2015] (Section 6.1.2), and models that are related to the conditional encoding architecture presented in Section 6.2.1. Results for these follow-up works are collected in a leaderboard at `http://nlp.stanford.edu/projects/snli/`. The current best result[5] is held by a bidirectional LSTM with matching and aggregation layers introduced by Wang et al. [2017]. It achieves a test accuracy of 88.8% and outperforms the best independent encoding model by 4.2 percentage points. In fact, most independent encoding models [*e.g.* Vendrov et al., 2016, Mou et al., 2016, Bowman et al., 2016, Munkhdalai and Yu, 2016a] do not reach the performance of our conditional model with word-by-word attention. Exceptions are the recently introduced two-stage bidirectional LSTM model by Liu et al. [2016] and the Neural Semantic Encoder by Munkhdalai and Yu [2016b].

6.5.1 Bidirectional Conditional Encoding

As the models presented in this chapter make little assumptions about the input data, they can be applied in other domains too. Augenstein et al. [2016] introduced a conditional encoding model for determining the stance of a tweet (*e.g.* “A foetus has rights too!”) with respect to a target (*e.g.* “Legalization of Abortion”). They

[5]Checked last on 26th of April, 2017.

further extended the conditional encoding model with bidirectional LSTMs [Graves and Schmidhuber, 2005], thus replacing Eqs. 6.21 to 6.23 with

$$\mathrm{H}^{\overrightarrow{P}}, \mathrm{S}^{\overrightarrow{P}} = \mathrm{RNN}\left(f^{\mathtt{LSTM}}_{\theta_{\overrightarrow{P}}}, \mathrm{X}^{\overrightarrow{P}}, \overrightarrow{\boldsymbol{s}}_0\right) \tag{6.32}$$

$$\mathrm{H}^{\overrightarrow{H}}, \mathrm{S}^{\overrightarrow{H}} = \mathrm{RNN}\left(f^{\mathtt{LSTM}}_{\theta_{\overrightarrow{H}}}, \mathrm{X}^{\overrightarrow{H}}, \boldsymbol{h}_N^{\overrightarrow{P}}\right) \tag{6.33}$$

$$\mathrm{H}^{\overleftarrow{H}}, \mathrm{S}^{\overleftarrow{H}} = \mathrm{RNN}\left(f^{\mathtt{LSTM}}_{\theta_{\overleftarrow{H}}}, \mathrm{X}^{\overleftarrow{H}}, \overleftarrow{\boldsymbol{s}}_0\right) \tag{6.34}$$

$$\mathrm{H}^{\overleftarrow{P}}, \mathrm{S}^{\overleftarrow{P}} = \mathrm{RNN}\left(f^{\mathtt{LSTM}}_{\theta_{\overleftarrow{P}}}, \mathrm{X}^{\overleftarrow{P}}, \boldsymbol{h}_M^{\overleftarrow{H}}\right) \tag{6.35}$$

$$\boldsymbol{h} = \tanh\left(\overrightarrow{\boldsymbol{W}}\boldsymbol{h}_M^{\overrightarrow{H}} + \overleftarrow{\boldsymbol{W}}\boldsymbol{h}_N^{\overleftarrow{P}}\right) \tag{6.36}$$

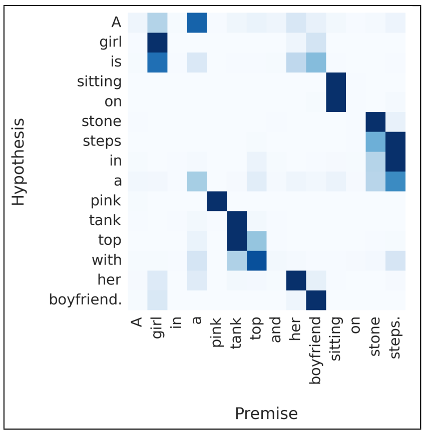
where $\overrightarrow{P}$ and $\overrightarrow{H}$ denotes the forward, and $\overleftarrow{P}$ and $\overleftarrow{H}$ the reversed sequence, $\overrightarrow{\boldsymbol{W}}, \overleftarrow{\boldsymbol{W}} \in \mathbb{R}^{k\times k}$ are trainable projection matrices, and $\overrightarrow{\boldsymbol{s}}_0$ and $\overleftarrow{\boldsymbol{s}}_0$ are the trainable forward and reverse start state, respectively. This architecture is illustrated in Fig. 6.9. It achieved the second best result on the SemEval 2016 Task 6 Twitter Stance Detection corpus [Mohammad et al., 2016].

6.5.2 Generating Entailing Sentences

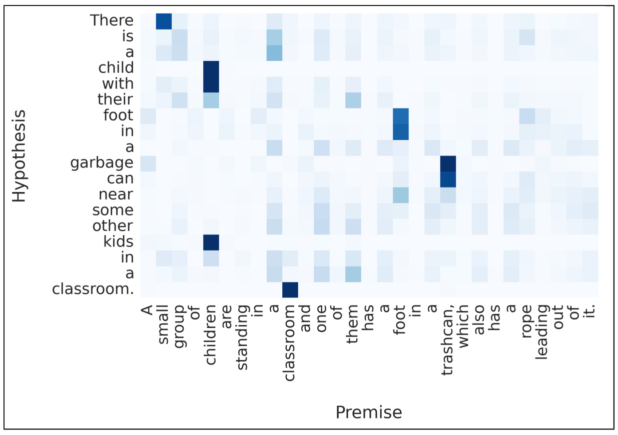
Instead of predicting the logical relationship between sentences, Kolesnyk et al. [2016] used entailment pairs of the SNLI corpus to learn to generate an entailed sentence given a premise. Their model is an encoder-decoder with attention, as used in neural machine translation [Bahdanau et al., 2014]. On a manually annotated test corpus of 100 generated sentences, their model generated correct entailed sentences in 82% of the cases. By recursively applying this encoder-decoder to the produced outputs, their model is able to generate natural language inference chains such as "A wedding party looks happy on the picture $\Rightarrow$ A bride and groom smiles $\Rightarrow$ Couple smiling $\Rightarrow$ Some people smiling."

6.6 Summary

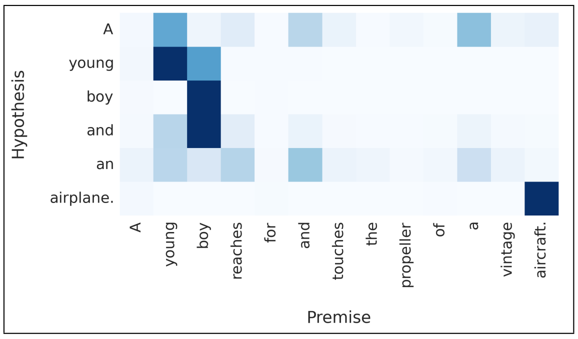
In this chapter, we demonstrated that LSTMs that read pairs of sequences to produce a final representation from which a simple classifier predicts entailment, outperform an LSTM baseline encoding the two sequences independently, as well as a classifier with hand-engineered features. Besides contributing this conditional model for RTE, our main contribution is to extend this model with attention over the premise which provides further improvements to the predictive abilities of the system. In a qualitative analysis, we showed that a word-by-word attention model is able to compare word and phrase pairs for deciding the logical relationship sentences. With an accuracy of 84.5%, it held the state-of-the-art on a large RTE corpus at the time of publication.



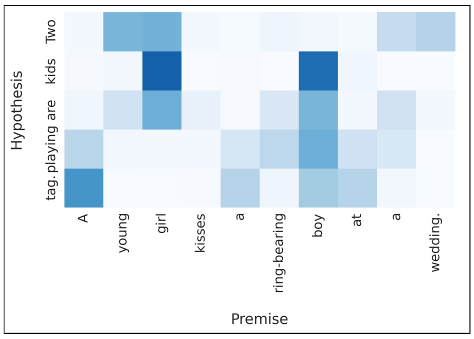
(a)



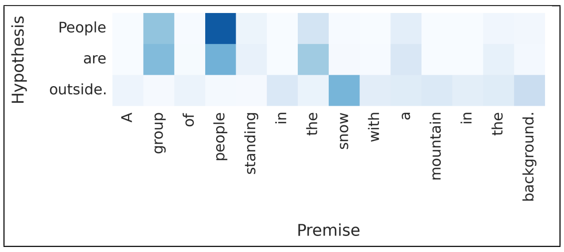
(b)



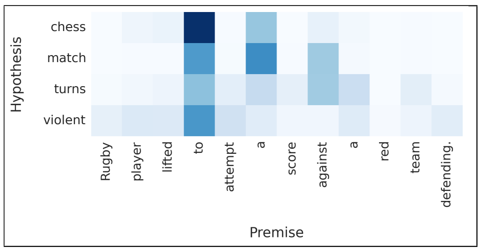
(c)



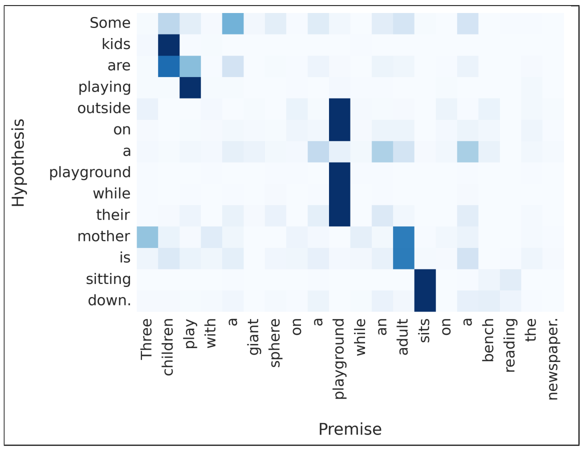
(d)



(e)



(f)



(g)

Figure 6.8: Word-by-word attention visualizations.

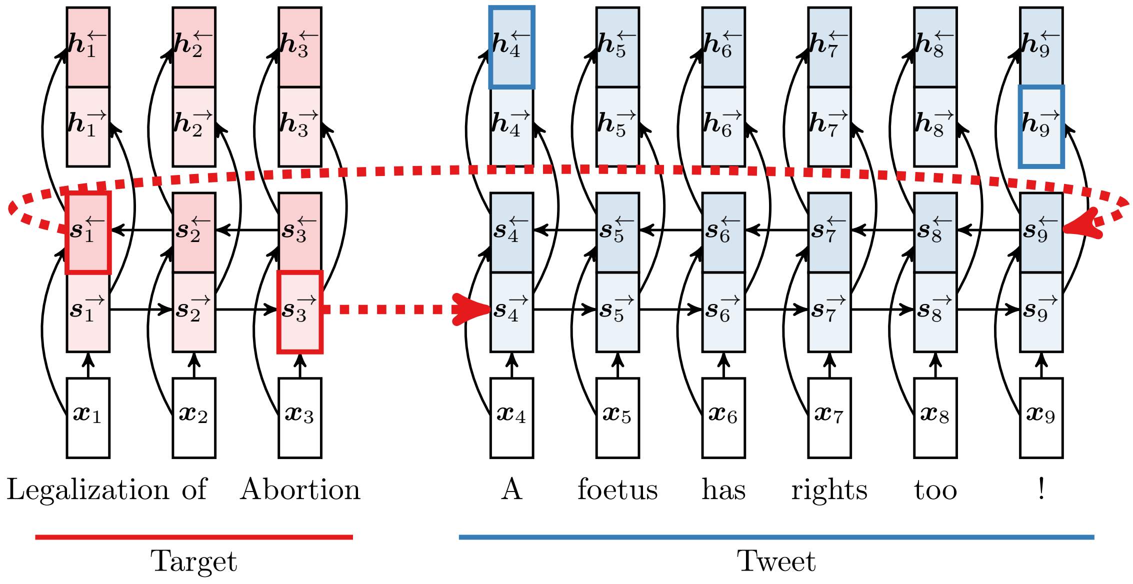

$h_1^{\leftarrow}$ $h_2^{\leftarrow}$ $h_3^{\leftarrow}$ $h_4^{\leftarrow}$ $h_5^{\leftarrow}$ $h_6^{\leftarrow}$ $h_7^{\leftarrow}$ $h_8^{\leftarrow}$ $h_9^{\leftarrow}$
$h_1^{\rightarrow}$ $h_2^{\rightarrow}$ $h_3^{\rightarrow}$ $h_4^{\rightarrow}$ $h_5^{\rightarrow}$ $h_6^{\rightarrow}$ $h_7^{\rightarrow}$ $h_8^{\rightarrow}$ $h_9^{\rightarrow}$
$s_1^{\leftarrow}$ $s_2^{\leftarrow}$ $s_3^{\leftarrow}$ $s_4^{\leftarrow}$ $s_5^{\leftarrow}$ $s_6^{\leftarrow}$ $s_7^{\leftarrow}$ $s_8^{\leftarrow}$ $s_9^{\leftarrow}$
$s_1^{\rightarrow}$ $s_2^{\rightarrow}$ $s_3^{\rightarrow}$ $s_4^{\rightarrow}$ $s_5^{\rightarrow}$ $s_6^{\rightarrow}$ $s_7^{\rightarrow}$ $s_8^{\rightarrow}$ $s_9^{\rightarrow}$
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9
Legalization of Abortion
A foetus has rights too !
Target
Tweet

Figure 6.9: Bidirectional encoding of a tweet conditioned on bidirectional encoding of a target ($[\boldsymbol{s}_3^{\rightarrow}\ \boldsymbol{s}_1^{\leftarrow}]$). The stance is predicted using the last forward and reversed output representations ($[\boldsymbol{h}_9^{\rightarrow}\ \boldsymbol{h}_4^{\leftarrow}]$).

Chapter 7

Conclusions

7.1 Summary of Contributions

In this thesis, we have presented various combinations of representation learning models with logic.

First, we proposed a way to calculate the gradient of propositional logic rules with respect to parameters of a neural link prediction model (Chapter 3). By stochastically grounding first-order logic rules, we were able to use these rules as regularizers in a matrix factorization neural link prediction model for automated Knowledge Base (KB) completion. This allowed us to embed background knowledge in form of logical rules in the vector space of predicate and entity pair representations. Using this method, we were able to train relation extractors for predicates with provided rules but little or no known training facts.

In Chapter 4, we identified various shortcomings of stochastic grounding and proposed a model in which implication rules are only used to regularize predicate representations. This has the two advantages that the method becomes independent of the size of the domain of entity pairs, and that we can guarantee that the provided rules will hold for any test entity pair. By restricting the entity pair embedding space to be non-negative, we were able to impose implications as a partial order on the predicate representation space similar to Order Embeddings [Vendrov et al., 2016]. We showed empirically that by restricting the entity pair embedding space, the model generalizes better to predicting facts in the test set, which we attribute to a regularization effect. Furthermore, we showed that incorporating implication rules with this method scales well with the number of rules.

After investigating two ways of regularizing symbol representations based on rules, in Chapter 5 we proposed a differentiable prover that performs KB inference with symbol representations in a more explicit way. To this end, we used Prolog's backward chaining algorithm as a recipe for recursively constructing neural networks that can be used to prove facts in a KB. Specifically, we proposed a differentiable unification operation between symbol representations. The constructed neural network allows us to compute the gradient of a proof success with respect to symbol representations, and thus train symbol representation end-to-end from the proof outcome. Furthermore, given templates for unknown rules of predefined structure, we can induce first-order logic rules using gradient descent. We proposed three optimizations for this model: (i) we implemented unification of multiple symbol representations as batch-operation which allows us to make use of modern Graphics Processing Units (GPUs) for efficient proving, (ii) we proposed an approximation of the gradient by only following $K \max$ proofs, and (iii) we used neural link prediction models as regularizers for the prover to learn better symbol representations more quickly. On three out of four benchmark knowledge bases, our method outperforms ComplEx, a state-of-the-art neural link prediction model, while at the same time inducing interpretable rules.

Lastly, we developed neural models for Recognizing Textual Entailment (RTE), *i.e.*, for determining the logical relationship between two natural language sentences (Chapter 6). We used one long short-term memory (LSTM) to encode the first sentence, and then conditioned on that representation encoded the second sentence using a second LSTM for deciding the label of the sentence pair. Furthermore, we extended this model with a neural word-by-word attention mechanism that enables more fine-grained comparison of word and phrase pairs. On a large RTE corpus, these models outperform a classifier with hand-engineered features and a strong LSTM baseline. In addition, we qualitatively analyzed the attention the model pays to words in the first sentence, and we were able to confirm the presence of fine-grained reasoning patterns.

7.2 Limitations and Future Work

The integration of neural representations with symbolic logic and reasoning remains an exciting and open research area, and we except to see much more systems improving representation learning models by taking inspiration from formal logic in the future. While we demonstrated the benefit of regularizing symbol representations by logical rules for automated KB completion, we were only able to do this efficiently for simple implication rules. For future work, it would be interesting to use more general first-order logic rules as regularizers on predicate representations in a lifted way, for instance, by a more informed grounding of first-order rules. However, it is likely that the approach of regularizing predicate representations using rules has theoretical limitations that need to be investigated further. Thus, we believe an interesting alternative direction is synthesizing symbolic reasoning and neural representations in more explicit ways.

The end-to-end differentiable Neural Theorem Prover (NTP) introduced in this thesis is only a first proposal towards a tight integration of symbolic reasoning systems with trainable rules and symbol representations. The major obstacle that we encountered has to do with the computational complexity of making the proof success differentiable so that we can calculate the gradient with respect to symbol representations. While it is possible to approximate the gradient by only maintaining the $K\max$ proofs for a given a query, at some point a unification of a query with all facts in a KB is necessary. As real-world KBs can contain millions of facts, this grounding becomes impossible to do efficiently without applying further heuristics even when using modern GPUs. A possible future direction could be the use of hierarchical attention [Andrychowicz et al., 2016], or recent methods for reinforcement learning such as Monte Carlo tree search [Coulom, 2006, Kocsis and Szepesvári, 2006] as used, for instance, for learning to play Go [Silver et al., 2016] or chemical synthesis planning [Segler et al., 2017]. Specifically, the idea would be to train a model that learns to select promising rules instead of trying all rules for proving a goal. Orthogonal to that, more flexible individual components of end-to-end differentiable provers are conceivable. For instance, unification, rule selection, and rule application could be modeled as parameterized functions, and thus could be used to learn a more optimal behavior from data in a KB than the behavior that we specified by closely following the backward chaining algorithm. Furthermore, while the NTP is constructed from Prolog's backward chaining, we currently only support Datalog logic programs, *i.e.*, function-free first-order logic. An open question is how we can enable support for function terms in end-to-end differentiable provers.

Another open research direction is the extension of automated provers to handle natural language sentences, questions, and perform multi-hop reasoning with natural language sentences. A starting point could be the combination of models that we proposed for determining the logical relationship between two natural language sentences, and the differentiable prover. As the end-to-end differentiable prover introduced in this thesis can be used to calculate the gradient of proof success with respect to symbol representations, these symbol representations can itself be composed by an RNN encoder that is trained jointly with the prover. The vision is a prover that directly operates on natural language statements and explanations, avoiding the need for semantic parsing [Zettlemoyer and Collins, 2005], *i.e.*, parsing text into logical form. As NTPs decompose inference in a more explicit way, it would be worthwhile to investigate whether we can obtain interpretable natural language proofs. Furthermore, it would be interesting to scale the methods presented here to larger units of text such as entire documents. Again, this needs model extensions such as hierarchical attention to ensure computational efficiency. In addition, it would be worthwhile exploring how other, more structured forms of attention [*e.g.* Graves et al., 2014, Sukhbaatar et al., 2015], or other forms of differentiable memory [*e.g.* Grefenstette et al., 2015, Joulin and Mikolov, 2015] could help improve performance of neural networks for RTE and differentiable proving. Lastly, we are interested in applying NTPs to automated proving of mathematical theorems, either in logical or natural language form, similar to the recent work by Kaliszyk et al. [2017] and Loos et al. [2017].

Appendix A

Annotated Rules

Manually filtered rules and their score for experiments in Chapter 3.

0.97 `organization/parent/child`(X,Y) :–
    `#2-nn<-unit->prep->of->pobj-#1`(X,Y).

0.97 `organization/parent/child`(X,Y) :–
    `#2->appos->subsidiary->prep->of->pobj-#1`(X,Y).

0.97 `organization/parent/child`(X,Y) :–
    `#2->rcmod->own->prep->by->pobj-#1`(X,Y).

0.97 `location/location/containedby`(X,Y) :–
    `#2-nn<-city->prep->of->pobj-#1`(X,Y).

0.97 `organization/parent/child`(X,Y) :–
    `#2->appos->subsidiary->nn-#1`(X,Y).

0.97 `people/person/nationality`(X,Y) :–
    `#2-poss<-minister->appos-#1`(X,Y).

0.97 `organization/parent/child`(X,Y) :–
    `#2->appos->unit->prep->of->pobj-#1`(X,Y).

0.96 `organization/parent/child`(X,Y) :–
    `#2->appos->division->prep->of->pobj-#1`(X,Y).

0.96 `business/person/company`(X,Y) :–
    `#2-poss<-executive->appos-#1`(X,Y).

0.96 `business/company/founders`(X,Y) :–
    `#2->appos->co-founder->prep->of->pobj-#1`(X,Y).

0.96 `book/author/works_written`(X,Y) :–
    `#2-dobj<-review->prep->by->pobj-#1`(X,Y).

0.95 `business/company/founders`(X,Y) :–
    `#2->appos->founder->prep->of->pobj-#1`(X,Y).

0.95 `location/location/containedby`(X,Y) :–

#2-nn<-town->prep->of->pobj-#1(X,Y).

0.95 location/neighborhood/neighborhood_of(X,Y) :–
#2-nn<-neighborhood->prep->of->pobj-#1(X,Y).

0.95 film/film/directed_by(X,Y) :–
#2->appos->director->dep-#1(X,Y).

0.95 location/location/containedby(X,Y) :–
#2-poss<-region->nn-#1(X,Y).

0.94 film/film/produced_by(X,Y) :–
#2->appos->producer->dep-#1(X,Y).

0.94 film/film/directed_by(X,Y) :–
#2-poss<-film->dep-#1(X,Y).

0.94 location/location/containedby(X,Y) :–
#2-nsubj<-professor->prep->at->pobj-#1(X,Y).

0.94 film/film/directed_by(X,Y) :–
#2-poss<-movie->dep-#1(X,Y).

0.93 people/person/nationality(X,Y) :–
#2-poss<-leader->appos-#1(X,Y).

0.93 film/film/directed_by(X,Y) :–
#2-nn<-film->dep-#1(X,Y).

0.93 location/location/containedby(X,Y) :–
#2-nn<-suburb->prep->of->pobj-#1(X,Y).

0.93 people/person/parents(X,Y) :–
#1->appos->daughter->prep->of->pobj-#2(X,Y).

0.93 business/person/company(X,Y) :–
#2-poss<-chairman->appos-#1(X,Y).

0.93 location/location/containedby(X,Y) :–
#2-nn<-side->prep->of->pobj-#1(X,Y).

0.93 people/deceased_person/place_of_death(X,Y) :–
#1-nsubj<-die->prep->in->pobj-#2(X,Y).

0.93 location/neighborhood/neighborhood_of(X,Y) :–
#2-poss<-neighborhood->nn-#1(X,Y).

0.91 location/location/containedby(X,Y) :–
#2-nsubj<-professor->appos-#1(X,Y).

0.91 people/deceased_person/place_of_death(X,Y) :–
#1-nsubj<-die->prep->at->pobj->hospital->prep->in->pobj-#2(X,Y).

0.91 book/author/works_written(X,Y) :–
#1-poss<-book->dep-#2(X,Y).

0.90 `business/person/company`(X,Y) :–

    `#2-nsubj<-name->dobj-#1`(X,Y).

0.90 `people/person/place_of_birth`(X,Y) :–

    `#1-nsubjpass<-bear->prep->in->pobj-#2`(X,Y).

0.90 `people/person/nationality`(X,Y) :–

    `#1->appos->minister->poss-#2`(X,Y).

0.88 `location/location/containedby`(X,Y) :–

    `#1->appos->capital->prep->of->pobj-#2`(X,Y).

0.87 `location/location/containedby`(X,Y) :–

    `#1-nsubj<-city->prep->in->pobj-#2`(X,Y).

Appendix B

Annotated WordNet Rules

Manually filtered rules derived from WordNet for experiments in Chapter 4.

```
#1->appos->organization->amod-#2(X,Y) :-
      #1->appos->party->amod-#2(X,Y).
#1-nsubj<-push->dobj-#2(X,Y) :-
      #1-nsubj<-press->dobj-#2(X,Y).
#1->appos->artist->amod-#2(X,Y) :-
      #1->appos->painter->amod-#2(X,Y).
#1->appos->artist->nn-#2(X,Y) :-
      #1->appos->painter->nn-#2(X,Y).
#1->appos->writer->amod-#2(X,Y) :-
      #1->appos->journalist->amod-#2(X,Y).
#1->appos->writer->amod-#2(X,Y) :-
      #1->appos->poet->amod-#2(X,Y).
#1-poss<-parent->appos-#2(X,Y) :-
      #1-poss<-father->appos-#2(X,Y).
#1->appos->lawyer->nn-#2(X,Y) :-
      #1->appos->prosecutor->nn-#2(X,Y).
#1->appos->expert->nn-#2(X,Y) :-
      #1->appos->specialist->nn-#2(X,Y).
#1->appos->newspaper->amod-#2(X,Y) :-
      #1->appos->daily->amod-#2(X,Y).
#1->appos->leader->nn-#2(X,Y) :-
      #1->appos->boss->nn-#2(X,Y).
#1->appos->firm->nn-#2(X,Y) :-
      #1->appos->publisher->nn-#2(X,Y).
#1->appos->journalist->nn-#2(X,Y) :-
```

```
    #1->appos->correspondent->nn-#2(X,Y).
#1->appos->company->amod-#2(X,Y) :-
    #1->appos->subsidiary->amod-#2(X,Y).
#1-nsubj<-purchase->dobj-#2(X,Y) :-
    #1-nsubj<-buy->dobj-#2(X,Y).
#1->appos->leader->nn-#2(X,Y) :-
    #1->appos->chief->nn-#2(X,Y).
#1->appos->player->poss-#2(X,Y) :-
    #1->appos->scorer->poss-#2(X,Y).
#1->appos->organization->nn-#2(X,Y) :-
    #1->appos->institution->nn-#2(X,Y).
#1->appos->center->amod-#2(X,Y) :-
    #1->appos->capital->amod-#2(X,Y).
#1->appos->center->poss-#2(X,Y) :-
    #1->appos->capital->poss-#2(X,Y).
#1->appos->representative->poss-#2(X,Y) :-
    #1->appos->envoy->poss-#2(X,Y).
#1->appos->expert->amod-#2(X,Y) :-
    #1->appos->specialist->amod-#2(X,Y).
#1->appos->center->nn-#2(X,Y) :-
    #1->appos->capital->nn-#2(X,Y).
#1->appos->writer->amod-#2(X,Y) :-
    #1->appos->novelist->amod-#2(X,Y).
#1->appos->diplomat->amod-#2(X,Y) :-
    #1->appos->ambassador->amod-#2(X,Y).
#1->appos->expert->amod-#2(X,Y) :-
    #1->appos->analyst->amod-#2(X,Y).
#1->appos->scholar->nn-#2(X,Y) :-
    #1->appos->historian->nn-#2(X,Y).
#1->appos->maker->amod-#2(X,Y) :-
    #1->appos->producer->amod-#2(X,Y).
#1->appos->maker->amod-#2(X,Y) :-
    #1->appos->manufacturer->amod-#2(X,Y).
#1->appos->official->amod-#2(X,Y) :-
    #1->appos->diplomat->amod-#2(X,Y).
#1->appos->trainer->poss-#2(X,Y) :-
    #1->appos->coach->poss-#2(X,Y).
```

```
#1->appos->member->amod-#2(X,Y) :–
      #1->appos->commissioner->amod-#2(X,Y).
#1->appos->institution->nn-#2(X,Y) :–
      #1->appos->company->nn-#2(X,Y).
#1->appos->representative->amod-#2(X,Y) :–
      #1->appos->envoy->amod-#2(X,Y).
#1->appos->scientist->nn-#2(X,Y) :–
      #1->appos->physicist->nn-#2(X,Y).
#1->appos->representative->nn-#2(X,Y) :–
      #1->appos->envoy->nn-#2(X,Y).
```

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016. URL `http://arxiv.org/abs/1603.04467`.

Yaser S. Abu-Mostafa. Learning from hints in neural networks. *J. Complexity*, 6(2): 192–198, 1990. doi: 10.1016/0885-064X(90)90006-Y. URL `http://dx.doi.org/10.1016/0885-064X(90)90006-Y`.

Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermüller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, Yoshua Bengio, Arnaud Bergeron, James Bergstra, Valentin Bisson, Josh Bleecher Snyder, Nicolas Bouchard, Nicolas Boulanger-Lewandowski, Xavier Bouthillier, Alexandre de Brébisson, Olivier Breuleux, Pierre Luc Carrier, Kyunghyun Cho, Jan Chorowski, Paul Christiano, Tim Cooijmans, Marc-Alexandre Côté, Myriam Côté, Aaron C. Courville, Yann N. Dauphin, Olivier Delalleau, Julien Demouth, Guillaume Desjardins, Sander Dieleman, Laurent Dinh, Melanie Ducoffe, Vincent Dumoulin, Samira Ebrahimi Kahou, Dumitru Erhan, Ziye Fan, Orhan Firat, Mathieu Germain, Xavier Glorot, Ian J. Goodfellow, Matthew Graham, Çaglar Gülçehre, Philippe Hamel, Iban Harlouchet, Jean-Philippe Heng, Balázs Hidasi, Sina Honari, Arjun Jain, Sébastien Jean, Kai Jia, Mikhail Korobov, Vivek Kulkarni, Alex Lamb, Pascal Lamblin, Eric Larsen, César Laurent, Sean Lee, Simon Lefrançois, Simon Lemieux, Nicholas Léonard, Zhouhan Lin, Jesse A. Livezey, Cory Lorenz, Jeremiah Lowin, Qianli Ma, Pierre-Antoine Manzagol, Olivier Mastropietro, Robert McGibbon, Roland Memisevic, Bart van Merriënboer, Vincent Michalski, Mehdi Mirza, Alberto Orlandi, Christopher Joseph Pal, Razvan Pascanu, Mohammad Pezeshki, Colin Raffel, Daniel Renshaw, Matthew Rocklin, Adriana Romero, Markus Roth, Peter Sadowski, John Salvatier, François Savard, Jan Schlüter, John Schulman, Gabriel Schwartz, Iulian Vlad Serban, Dmitriy Serdyuk, Samira Shabanian, Étienne Simon, Sigurd Spieckermann, S. Ramana Subramanyam, Jakub Sygnowski, Jérémie Tanguay, Gijs van Tulder, Joseph P. Turian, Sebastian Urban, Pascal Vincent, Francesco Visin, Harm de Vries, David Warde-Farley, Dustin J. Webb, Matthew Willson, Kelvin Xu, Lijun Xue, Li Yao, Saizheng Zhang, and Ying Zhang. Theano: A python framework for fast computation of mathematical expressions. *CoRR*, abs/1605.02688, 2016. URL `http://arxiv.org/abs/1605.02688`.

Miltiadis Allamanis, Hao Peng, and Charles A. Sutton. A convolutional attention network for extreme summarization of source code. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 2091–2100, 2016. URL `http://jmlr.org/proceedings/papers/v48/allamanis16.html`.

Miltiadis Allamanis, Pankajan Chanthirasegaran, Pushmeet Kohli, and Charles A. Sutton. Learning continuous semantic representations of symbolic expressions. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 80–88, 2017. URL `http://proceedings.mlr.press/v70/allamanis17a.html`.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Learning to compose neural networks for question answering. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 1545–1554, 2016. URL `http://aclweb.org/anthology/N/N16/N16-1181.pdf`.

Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3981–3989,

2016. URL http://papers.nips.cc/paper/6461-learning-to-learn-by-gradient-descent-by-gradient-descent.

Gabor Angeli and Christopher D. Manning. Naturalli: Natural logic inference for common sense reasoning. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 534–545, 2014. URL http://aclweb.org/anthology/D/D14/D14-1059.pdf.

Isabelle Augenstein, Tim Rocktäschel, Andreas Vlachos, and Kalina Bontcheva. Stance detection with bidirectional conditional encoding. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 876–885, 2016. URL http://aclweb.org/anthology/D/D16/D16-1084.pdf.

Franz Baader, Bernhard Ganter, Baris Sertkaya, and Ulrike Sattler. Completing description logic knowledge bases using formal concept analysis. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 230–235, 2007. URL http://ijcai.org/Proceedings/07/Papers/035.pdf.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL http://arxiv.org/abs/1409.0473.

Islam Beltagy, Cuong Chau, Gemma Boleda, Dan Garrette, Katrin Erk, and Raymond J. Mooney. Montague Meets Markov: Deep Semantics with Probabilistic Logical Form. In *Proceedings of the Second Joint Conference on Lexical and Computational Semantics, *SEM 2013, June 13-14, 2013, Atlanta, Georgia, USA.*, pages 11–21, 2013. URL http://aclweb.org/anthology/S/S13/S13-1002.pdf.

Islam Beltagy, Katrin Erk, and Raymond J. Mooney. Probabilistic soft logic for semantic textual similarity. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 1210–1219, 2014. URL http://aclweb.org/anthology/P/P14/P14-1114.pdf.

Islam Beltagy, Stephen Roller, Pengxiang Cheng, Katrin Erk, and Raymond J. Mooney. Representing meaning with a combination of logical form and vectors. *CoRR*, abs/1505.06816, 2015. URL `http://arxiv.org/abs/1505.06816`.

Islam Beltagy, Stephen Roller, Pengxiang Cheng, Katrin Erk, and Raymond J Mooney. Representing meaning with a combination of logical and distributional models. *Computational Linguistics*, 2017.

Kurt D. Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1247–1250, 2008. doi: 10.1145/1376616.1376746. URL `http://doi.acm.org/10.1145/1376616.1376746`.

Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*, 2011. URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3659`.

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 2787–2795, 2013. URL `http://papers.nips.cc/paper/5071-translating-embeddings-for-modeling-multi-relational-data`.

Johan Bos. Wide-coverage semantic analysis with boxer. In *Semantics in Text Processing. STEP 2008 Conference Proceedings, volume 1 of Research in Computational Semantics*, pages 277–286. College Publications, 2008.

Johan Bos and Katja Markert. Recognising textual entailment with logical inference. In *HLT/EMNLP 2005, Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference, 6-8 October 2005, Vancouver, British Columbia, Canada*, pages 628–635, 2005. URL `http://aclweb.org/anthology/H/H05/H05-1079.pdf`.

Matko Bosnjak, Tim Rocktäschel, Jason Naradowsky, and Sebastian Riedel. Programming with a differentiable forth interpreter. In *International Conference on Machine Learning (ICML)*, 2017. URL `http://arxiv.org/abs/1605.06640`.

Guillaume Bouchard, Sameer Singh, and Theo Trouillon. On approximate reasoning capabilities of low-rank vector spaces. In *Proceedings of the 2015 AAAI Spring Symposium on Knowledge Representation and Reasoning (KRR): Integrating Symbolic and Neural Approaches*, 2015.

Samuel R. Bowman. Can recursive neural tensor networks learn logical reasoning? volume abs/1312.6192, 2013. URL `http://arxiv.org/abs/1312.6192`.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 632–642, 2015. URL `http://aclweb.org/anthology/D/D15/D15-1075.pdf`.

Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. A fast unified model for parsing and sentence understanding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016. URL `http://aclweb.org/anthology/P/P16/P16-1139.pdf`.

Rodrigo De Salvo Braz. *Lifted First-order Probabilistic Inference*. PhD thesis, Champaign, IL, USA, 2007. AAI3290183.

David S Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, DTIC Document, 1988.

Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Scientific Computing*, 16 (5):1190–1208, 1995. doi: 10.1137/0916069. URL `http://dx.doi.org/10.1137/0916069`.

Andrew Carlson, Justin Betteridge, Richard C. Wang, Estevam R. Hruschka Jr., and Tom M. Mitchell. Coupled semi-supervised learning for information extraction. In

Proceedings of the Third International Conference on Web Search and Web Data Mining, WSDM 2010, New York, NY, USA, February 4-6, 2010, pages 101–110, 2010. doi: 10.1145/1718487.1718501. URL `http://doi.acm.org/10.1145/1718487.1718501`.

Kai-Wei Chang, Wen-tau Yih, Bishan Yang, and Christopher Meek. Typed tensor decomposition of knowledge bases for relation extraction. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1568–1579, 2014. URL `http://aclweb.org/anthology/D/D14/D14-1165.pdf`.

Ming-Wei Chang, Lev-Arie Ratinov, and Dan Roth. Guiding semi-supervision with constraint-driven learning. In *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic*, 2007. URL `http://aclweb.org/anthology-new/P/P07/P07-1036.pdf`.

Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 577–585, 2015. URL `http://papers.nips.cc/paper/5847-attention-based-models-for-speech-recognition`.

Stephen Clark and Stephen Pulman. Combining symbolic and distributional models of meaning. In *Quantum Interaction, Papers from the 2007 AAAI Spring Symposium, Technical Report SS-07-08, Stanford, California, USA, March 26-28, 2007*, pages 52–55, 2007. URL `http://www.aaai.org/Library/Symposia/Spring/2007/ss07-08-008.php`.

Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. Mathematical foundations for a compositional distributional model of meaning. *CoRR*, abs/1003.4394, 2010. URL `http://arxiv.org/abs/1003.4394`.

William W. Cohen. Tensorlog: A differentiable deductive database. *CoRR*, abs/1605.06523, 2016. URL `http://arxiv.org/abs/1605.06523`.

Michael Collins, Sanjoy Dasgupta, and Robert E. Schapire. A generalization of principal components analysis to the exponential family. In

Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada], pages 617–624, 2001. URL http://papers.nips.cc/paper/2078-a-generalization-of-principal-components-analysis-to-the-exponential-family.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

Alain Colmerauer. An introduction to prolog III. *Commun. ACM*, 33(7):69–90, 1990. doi: 10.1145/79204.79210. URL http://doi.acm.org/10.1145/79204.79210.

Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers*, pages 72–83, 2006. doi: 10.1007/978-3-540-75538-8_7. URL http://dx.doi.org/10.1007/978-3-540-75538-8_7.

Council of the European Union. Position of the council on general data protection regulation. http://www.europarl.europa.eu/sed/doc/news/document/CONS_CONS(2016)05418(REV1)_EN.docx, 2016. 8 April 2016.

Ido Dagan, Oren Glickman, and Bernardo Magnini. The PASCAL recognising textual entailment challenge. In *Machine Learning Challenges, Evaluating Predictive Uncertainty, Visual Object Classification and Recognizing Textual Entailment, First PASCAL Machine Learning Challenges Workshop, MLCW 2005, Southampton, UK, April 11-13, 2005, Revised Selected Papers*, pages 177–190, 2005. doi: 10.1007/11736790_9. URL http://dx.doi.org/10.1007/11736790_9.

Rajarshi Das, Arvind Neelakantan, David Belanger, and Andrew McCallum. Chains of reasoning over entities, relations, and text using recurrent neural networks. In *Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2017. URL http://arxiv.org/abs/1607.01426.

Artur S. d'Avila Garcez and Gerson Zaverucha. The connectionist inductive learning and logic programming system. *Appl. Intell.*, 11(1):59–77, 1999. doi: 10.1023/A:1008328630915. URL `http://dx.doi.org/10.1023/A:1008328630915`.

Artur S. d'Avila Garcez, Krysia Broda, and Dov M. Gabbay. *Neural-symbolic learning systems: foundations and applications*. Springer Science & Business Media, 2012.

Oier Lopez de Lacalle and Mirella Lapata. Unsupervised relation extraction with general domain knowledge. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 415–425, 2013. URL `http://aclweb.org/anthology/D/D13/D13-1040.pdf`.

Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. Lifted rule injection for relation embeddings. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 1389–1399, 2016. URL `http://aclweb.org/anthology/D/D16/D16-1146.pdf`.

Liya Ding. Neural prolog-the concepts, construction and mechanism. In *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*, volume 4, pages 3603–3608. IEEE, 1995.

Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 601–610, 2014. doi: 10.1145/2623330.2623623. URL `http://doi.acm.org/10.1145/2623330.2623623`.

Gregory Druck, Gideon S. Mann, and Andrew McCallum. Semi-supervised learning of dependency parsers using generalized expectation criteria. In *ACL 2009, Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP, 2-7 August 2009, Singapore*, pages 360–368, 2009. URL `http://www.aclweb.org/anthology/P09-1041`.

John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011. URL http://dl.acm.org/citation.cfm?id=2021068.

Saso Dzeroski. Inductive logic programming in a nutshell. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*, chapter 3, pages 57–92. 2007.

Oren Etzioni, Michele Banko, Stephen Soderland, and Daniel S. Weld. Open information extraction from the web. *Commun. ACM*, 51(12):68–74, 2008. doi: 10.1145/1409360.1409378. URL http://doi.acm.org/10.1145/1409360.1409378.

Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard H. Hovy, and Noah A. Smith. Retrofitting word vectors to semantic lexicons. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 1606–1615, 2015. URL http://aclweb.org/anthology/N/N15/N15-1184.pdf.

John R Firth. A synopsis of linguistic theory 1930–55, 1957.

Manoel V. M. França, Gerson Zaverucha, and Artur S. d'Avila Garcez. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine Learning*, 94(1):81–104, 2014. doi: 10.1007/s10994-013-5392-1. URL http://dx.doi.org/10.1007/s10994-013-5392-1.

Luis Antonio Galárraga, Nicoleta Preda, and Fabian M. Suchanek. Mining rules to align knowledge bases. In *Proceedings of the 2013 workshop on Automated knowledge base construction, AKBC@CIKM 13, San Francisco, California, USA, October 27-28, 2013*, pages 43–48, 2013. doi: 10.1145/2509558.2509566. URL http://doi.acm.org/10.1145/2509558.2509566.

Hervé Gallaire and Jack Minker, editors. *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, 1977*, Advances in Data Base Theory, New York, 1978. Plemum Press. ISBN 0-306-40060-X.

Kuzman Ganchev, João Graça, Jennifer Gillenwater, and Ben Taskar. Posterior regularization for structured latent variable models. *Journal of Machine Learning Re-*

search, 11:2001–2049, 2010. URL http://portal.acm.org/citation.cfm?id=1859918.

Matt Gardner, Partha Pratim Talukdar, Bryan Kisiel, and Tom M. Mitchell. Improving learning and inference in a large knowledge-base using latent syntactic cues. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 833–838, 2013. URL http://aclweb.org/anthology/D/D13/D13-1080.pdf.

Matt Gardner, Partha Pratim Talukdar, Jayant Krishnamurthy, and Tom M. Mitchell. Incorporating vector space similarity in random walk inference over knowledge bases. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 397–406, 2014. URL http://aclweb.org/anthology/D/D14/D14-1044.pdf.

Dan Garrette, Katrin Erk, and Raymond J. Mooney. Integrating logical representations with probabilistic information using markov logic. In *Proceedings of the Ninth International Conference on Computational Semantics, IWCS 2011, January 12-14, 2011, Oxford, UK*, 2011. URL http://aclweb.org/anthology/W/W11/W11-0112.pdf.

Allen Van Gelder. Efficient loop detection in prolog using the tortoise-and-hare technique. *J. Log. Program.*, 4(1):23–31, 1987. doi: 10.1016/0743-1066(87)90020-3. URL https://doi.org/10.1016/0743-1066(87)90020-3.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, pages 249–256, 2010. URL http://www.jmlr.org/proceedings/papers/v9/glorot10a.html.

Kurt Gödel. Zum intuitionistischen aussagenkalkül. *Anzeiger der Akademie der Wissenschaften in Wien*, 69:65–66, 1932.

Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. ISBN 978-0-262-03561-3. URL `http://www.deeplearningbook.org/`.

Bryce Goodman and Seth Flaxman. EU regulations on algorithmic decision-making and a "right to explanation". *CoRR*, abs/1606.08813, 2016. URL `http://arxiv.org/abs/1606.08813`.

Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013. URL `http://arxiv.org/abs/1308.0850`.

Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5-6):602–610, 2005. doi: 10.1016/j.neunet.2005.06.042. URL `http://dx.doi.org/10.1016/j.neunet.2005.06.042`.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014. URL `http://arxiv.org/abs/1410.5401`.

Edward Grefenstette. Towards a formal distributional semantics: Simulating logical calculi with tensors. In *Proceedings of the Second Joint Conference on Lexical and Computational Semantics, *SEM 2013, June 13-14, 2013, Atlanta, Georgia, USA.*, pages 1–10, 2013. URL `http://aclweb.org/anthology/S/S13/S13-1001.pdf`.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1828–1836, 2015. URL `http://papers.nips.cc/paper/5648-learning-to-transduce-with-unbounded-memory`.

Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Jointly embedding knowledge graphs and logical rules. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 192–202, 2016. URL `http://aclweb.org/anthology/D/D16/D16-1019.pdf`.

Karl Moritz Hermann and Phil Blunsom. The role of syntax in vector space models of compositional semantics. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August*

2013, Sofia, Bulgaria, Volume 1: Long Papers, pages 894–904, 2013. URL `http://aclweb.org/anthology/P/P13/P13-1088.pdf`.

Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1693–1701, 2015. URL `http://papers.nips.cc/paper/5945-teaching-machines-to-read-and-comprehend`.

Jochen Hipp, Ulrich Güntzer, and Gholamreza Nakhaeizadeh. Algorithms for association rule mining - A general survey and comparison. *SIGKDD Explorations*, 2(1):58–64, 2000. doi: 10.1145/360402.360421. URL `http://doi.acm.org/10.1145/360402.360421`.

Pascal Hitzler, Steffen Hölldobler, and Anthony Karel Seda. Logic programs and connectionist networks. *J. Applied Logic*, 2(3):245–272, 2004. doi: 10.1016/j.jal.2004.03.002. URL `http://dx.doi.org/10.1016/j.jal.2004.03.002`.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL `http://dx.doi.org/10.1162/neco.1997.9.8.1735`.

Steffen Hölldobler. A structured connectionist unification algorithm. In *Proceedings of the 8th National Conference on Artificial Intelligence. Boston, Massachusetts, July 29 - August 3, 1990, 2 Volumes.*, pages 587–593, 1990. URL `http://www.aaai.org/Library/AAAI/1990/aaai90-088.php`.

Steffen Hölldobler, Yvonne Kalinke, and Hans-Peter Störr. Approximating the semantics of logic programs by recurrent neural networks. *Appl. Intell.*, 11(1):45–58, 1999. doi: 10.1023/A:1008376514077. URL `http://dx.doi.org/10.1023/A:1008376514077`.

Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. Convolutional neural network architectures for matching natural language sentences. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2042–2050, 2014.

URL `http://papers.nips.cc/paper/5550-convolutional-neural-network-architectures-for-matching-natural-language-sentences`.

Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard H. Hovy, and Eric P. Xing. Harnessing deep neural networks with logic rules. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016. URL `http://aclweb.org/anthology/P/P16/P16-1228.pdf`.

Sergio Jiménez, George Dueñas, Julia Baquero, and Alexander F. Gelbukh. UNAL-NLP: combining soft cardinality features for semantic textual similarity, relatedness and entailment. In *Proceedings of the 8th International Workshop on Semantic Evaluation, SemEval@COLING 2014, Dublin, Ireland, August 23-24, 2014.*, pages 732–742, 2014. URL `http://aclweb.org/anthology/S/S14/S14-2131.pdf`.

Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 190–198, 2015. URL `http://papers.nips.cc/paper/5857-inferring-algorithmic-patterns-with-stack-augmented-recurrent-nets`.

Cezary Kaliszyk, François Chollet, and Christian Szegedy. Holstep: A machine learning dataset for higher-order logic theorem proving. In *International Conference on Learning Representations (ICLR)*, 2017. URL `http://arxiv.org/abs/1703.00426`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. URL `http://arxiv.org/abs/1412.6980`.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, pages 282–293, 2006. doi: 10.1007/11871842_29. URL `http://dx.doi.org/10.1007/11871842_29`.

Stanley Kok and Pedro M. Domingos. Statistical predicate invention. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, pages 433–440, 2007. doi: 10.1145/1273496.1273551. URL `http://doi.acm.org/10.1145/1273496.1273551`.

Vladyslav Kolesnyk, Tim Rocktäschel, and Sebastian Riedel. Generating natural language inference chains. *CoRR*, abs/1606.01404, 2016. URL `http://arxiv.org/abs/1606.01404`.

Ekaterina Komendantskaya. Unification neural networks: unification by error-correction learning. *Logic Journal of the IGPL*, 19(6):821–847, 2011. doi: 10.1093/jigpal/jzq012. URL `http://dx.doi.org/10.1093/jigpal/jzq012`.

Robert A. Kowalski and Donald Kuehner. Linear resolution with selection function. *Artif. Intell.*, 2(3/4):227–260, 1971. doi: 10.1016/0004-3702(71)90012-9. URL `http://dx.doi.org/10.1016/0004-3702(71)90012-9`.

Germán Kruszewski, Denis Paperno, and Marco Baroni. Deriving boolean structures from distributional vectors. *TACL*, 3:375–388, 2015. URL `https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/616`.

Alice Lai and Julia Hockenmaier. Illinois-lh: A denotational and distributional approach to semantics. In *Proceedings of the 8th International Workshop on Semantic Evaluation, SemEval@COLING 2014, Dublin, Ireland, August 23-24, 2014.*, pages 329–334, 2014. URL `http://aclweb.org/anthology/S/S14/S14-2055.pdf`.

Ni Lao and William W. Cohen. Relational retrieval using a combination of path-constrained random walks. *Machine Learning*, 81(1):53–67, 2010. doi: 10.1007/s10994-010-5205-8. URL `http://dx.doi.org/10.1007/s10994-010-5205-8`.

Ni Lao, Tom M. Mitchell, and William W. Cohen. Random walk inference and learning in A large scale knowledge base. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John McIntyre Conference Centre, Edinburgh, UK, A meeting of*

SIGDAT, a Special Interest Group of the ACL, pages 529–539, 2011. URL http://www.aclweb.org/anthology/D11-1049.

Ni Lao, Amarnag Subramanya, Fernando C. N. Pereira, and William W. Cohen. Reading the web with learned syntactic-semantic inference rules. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, EMNLP-CoNLL 2012, July 12-14, 2012, Jeju Island, Korea*, pages 1017–1026, 2012. URL http://www.aclweb.org/anthology/D12-1093.

Mike Lewis and Mark Steedman. Combined distributional and logical semantics. *TACL*, 1:179–192, 2013. URL https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/93.

Yang Liu, Chengjie Sun, Lei Lin, and Xiaolong Wang. Learning natural language inference using bidirectional LSTM model and inner-attention. *CoRR*, abs/1605.09090, 2016. URL http://arxiv.org/abs/1605.09090.

John W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987. ISBN 3-540-18199-7.

Sarah M. Loos, Geoffrey Irving, Christian Szegedy, and Cezary Kaliszyk. Deep network guided proof search. In *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, pages 85–105, 2017. URL http://www.easychair.org/publications/paper/340345.

Will Lowe and Scott McDonald. The direct route: Mediated priming in semantic space. Technical report, The University of Edinburgh, 2000.

Jan Lukasiewicz. O logice trojwartosciowej. *Ruch filozoficzny*, 5(169-171):32, 1920.

Gideon S. Mann and Andrew McCallum. Generalized expectation criteria for semi-supervised learning of conditional random fields. In *ACL 2008, Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics, June 15-20, 2008, Columbus, Ohio, USA*, pages 870–878, 2008. URL http://www.aclweb.org/anthology/P08-1099.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008. ISBN 978-0-521-86571-5.

Marco Marelli, Luisa Bentivogli, Marco Baroni, Raffaella Bernardi, Stefano Menini, and Roberto Zamparelli. Semeval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. In *Proceedings of the 8th International Workshop on Semantic Evaluation, SemEval@COLING 2014, Dublin, Ireland, August 23-24, 2014.*, pages 1–8, 2014. URL `http://aclweb.org/anthology/S/S14/S14-2001.pdf`.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3111–3119, 2013. URL `http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality`.

George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11): 39–41, 1995. doi: 10.1145/219717.219748. URL `http://doi.acm.org/10.1145/219717.219748`.

Jeff Mitchell and Mirella Lapata. Vector-based models of semantic composition. In *ACL 2008, Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics, June 15-20, 2008, Columbus, Ohio, USA*, pages 236–244, 2008. URL `http://www.aclweb.org/anthology/P08-1028`.

Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2204–2212, 2014. URL `http://papers.nips.cc/paper/5542-recurrent-models-of-visual-attention`.

Saif Mohammad, Svetlana Kiritchenko, Parinaz Sobhani, Xiao-Dan Zhu, and Colin Cherry. Semeval-2016 task 6: Detecting stance in tweets. In *Proceedings of the 10th International Workshop on Semantic Evaluation, SemEval@NAACL-HLT 2016, San Diego, CA, USA, June 16-17, 2016*, pages 31–41, 2016. URL `http://aclweb.org/anthology/S/S16/S16-1003.pdf`.

Lili Mou, Rui Men, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. Natural language inference by tree-based convolution and heuristic matching. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*, 2016. URL `http://aclweb.org/anthology/P/P16/P16-2022.pdf`.

Stephen Muggleton. Inductive logic programming. *New Generation Comput.*, 8(4): 295–318, 1991. doi: 10.1007/BF03037089. URL `http://dx.doi.org/10.1007/BF03037089`.

Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *J. Log. Program.*, 19/20:629–679, 1994. doi: 10.1016/0743-1066(94)90035-3. URL `http://dx.doi.org/10.1016/0743-1066(94)90035-3`.

Stephen H Muggleton, Dianhuan Lin, and Alireza Tamaddoni-Nezhad. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. *Machine Learning*, 100(1):49–73, 2015.

Tsendsuren Munkhdalai and Hong Yu. Neural tree indexers for text understanding. *CoRR*, abs/1607.04492, 2016a. URL `http://arxiv.org/abs/1607.04492`.

Tsendsuren Munkhdalai and Hong Yu. Neural semantic encoders. *CoRR*, abs/1607.04315, 2016b. URL `http://arxiv.org/abs/1607.04315`.

Arvind Neelakantan, Benjamin Roth, and Andrew McCallum. Compositional vector space models for knowledge base completion. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 156–166, 2015. URL `http://aclweb.org/anthology/P/P15/P15-1016.pdf`.

Arvind Neelakantan, Quoc V. Le, and Ilya Sutskever. Neural programmer: Inducing latent programs with gradient descent. In *International Conference on Learning Representations (ICLR)*, 2016. URL `http://arxiv.org/abs/1511.04834`.

A Nemirovski and D Yudin. On Cezari's convergence of the steepest descent method for approximating saddle point of convex-concave functions. In *Soviet Mathematics Doklady*, volume 19, 1978.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 809–816, 2011.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. Factorizing YAGO: scalable machine learning for linked data. In *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012*, pages 271–280, 2012. doi: 10.1145/2187836.2187874. URL http://doi.acm.org/10.1145/2187836.2187874.

Maximilian Nickel, Lorenzo Rosasco, and Tomaso A. Poggio. Holographic embeddings of knowledge graphs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 1955–1961, 2016. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12484.

Ulf Nilsson and Jan Maluszynski. *Logic, programming and Prolog*. Wiley, 1990. ISBN 978-0-471-92625-2.

Sebastian Padó and Mirella Lapata. Dependency-based construction of semantic space models. *Computational Linguistics*, 33(2):161–199, 2007. doi: 10.1162/coli.2007.33.2.161. URL http://dx.doi.org/10.1162/coli.2007.33.2.161.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

Baolin Peng, Zhengdong Lu, Hang Li, and Kam-Fai Wong. Towards neural network-based reasoning. *CoRR*, abs/1508.05508, 2015. URL http://arxiv.org/abs/1508.05508.

David Poole. First-order probabilistic inference. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 985–991, 2003. URL http://ijcai.org/Proceedings/03/Papers/142.pdf.

J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5: 239–266, 1990. doi: 10.1007/BF00117105. URL `http://dx.doi.org/10.1007/BF00117105`.

Luc De Raedt. Logical settings for concept-learning. *Artif. Intell.*, 95(1):187–201, 1997. doi: 10.1016/S0004-3702(97)00041-6. URL `http://dx.doi.org/10.1016/S0004-3702(97)00041-6`.

Scott E. Reed and Nando de Freitas. Neural programmer-interpreters. In *International Conference on Learning Representations (ICLR)*, 2016. URL `http://arxiv.org/abs/1511.06279`.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: bayesian personalized ranking from implicit feedback. In *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009*, pages 452–461, 2009. URL `https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1630&proceeding_id=25`.

Matthew Richardson and Pedro M. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006. doi: 10.1007/s10994-006-5833-1. URL `http://dx.doi.org/10.1007/s10994-006-5833-1`.

Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M. Marlin. Relation extraction with matrix factorization and universal schemas. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*, pages 74–84, 2013. URL `http://aclweb.org/anthology/N/N13/N13-1008.pdf`.

Tim Rocktäschel and Sebastian Riedel. Learning knowledge base inference with neural theorem provers. In *Proceedings of the 5th Workshop on Automated Knowledge Base Construction, AKBC@NAACL-HLT 2016, San Diego, CA, USA, June 17, 2016*, pages 45–50, 2016. URL `http://aclweb.org/anthology/W/W16/W16-1309.pdf`.

Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long*

Beach, California, United States, volume abs/1705.11040, 2017. URL `http://arxiv.org/abs/1705.11040`.

Tim Rocktäschel, Matko Bosnjak, Sameer Singh, and Sebastian Riedel. Low-Dimensional Embeddings of Logic. In *ACL Workshop on Semantic Parsing (SP'14)*, 2014.

Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 1119–1129, 2015. URL `http://aclweb.org/anthology/N/N15/N15-1118.pdf`.

Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomas Kocisky, and Phil Blunsom. Reasoning about entailment with neural attention. In *International Conference on Learning Representations (ICLR)*, 2016.

Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 379–389, 2015. URL `http://aclweb.org/anthology/D/D15/D15-1044.pdf`.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010. ISBN 978-0-13-207148-2.

Ivan Sanchez, Tim Rocktäschel, Sebastian Riedel, and Sameer Singh. Towards Extracting Faithful and Descriptive Representations of Latent Variable Models. In *AAAI Spring Symposium on Knowledge Representation and Reasoning (KRR)*, 2015.

Evan Sandhaus. The New York Times annotated corpus. *Linguistic Data Consortium*, 2008.

Stefan Schoenmackers, Oren Etzioni, and Daniel S. Weld. Scaling textual inference to the web. In *2008 Conference on Empirical Methods in Natural Language Processing, EMNLP 2008, Proceedings of the Conference, 25-27 October 2008, Honolulu, Hawaii, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 79–88, 2008. URL `http://www.aclweb.org/anthology/D08-1009`.

Stefan Schoenmackers, Jesse Davis, Oren Etzioni, and Daniel Weld. Learning first-order horn clauses from web text. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2010a. URL `http://www.aclweb.org/anthology/D10-1106`.

Stefan Schoenmackers, Jesse Davis, Oren Etzioni, and Daniel S. Weld. Learning first-order horn clauses from web text. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, EMNLP 2010, 9-11 October 2010, MIT Stata Center, Massachusetts, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1088–1098, 2010b. URL `http://www.aclweb.org/anthology/D10-1106`.

Marwin H. S. Segler, Mike Preuß, and Mark P. Waller. Towards ”alphachem”: Chemical synthesis planning with tree search and deep neural network policies. *CoRR*, abs/1702.00020, 2017. URL `http://arxiv.org/abs/1702.00020`.

Luciano Serafini and Artur S. d'Avila Garcez. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. In *Proceedings of the 11th International Workshop on Neural-Symbolic Learning and Reasoning (NeSy'16) co-located with the Joint Multi-Conference on Human-Level Artificial Intelligence (HLAI 2016), New York City, NY, USA, July 16-17, 2016.*, 2016. URL `http://ceur-ws.org/Vol-1768/NESY16_paper3.pdf`.

Lokendra Shastri. Neurally motivated constraints on the working memory capacity of a production system for parallel processing: Implications of a connectionist model based on temporal synchrony. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society: July 29 to August 1, 1992, Cognitive Science Program, Indiana University, Bloomington*, volume 14, page 159. Psychology Press, 1992.

Jude W Shavlik and Geoffrey G Towell. An approach to combining explanation-based and neural learning algorithms. *Connection Science*, 1(3):231–253, 1989.

Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. In *Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, December 9, 2016.*, 2016. URL `http://ceur-ws.org/Vol-1773/CoCoNIPS_2016_paper2.pdf`.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. doi: 10.1038/nature16961. URL `http://dx.doi.org/10.1038/nature16961`.

Sameer Singh, Dustin Hillard, and Chris Leggetter. Minimally-supervised extraction of entities from text advertisements. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 2-4, 2010, Los Angeles, California, USA*, pages 73–81, 2010. URL `http://www.aclweb.org/anthology/N10-1009`.

Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.*, pages 801–809, 2011. URL `http://papers.nips.cc/paper/4204-dynamic-pooling-and-unfolding-recursive-autoencoders-for-paraphrase-detection`.

Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Y. Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 926–934, 2013. URL `http://papers.nips.cc/paper/5028-reasoning-with-neural-tensor-networks-for-knowledge-base-completion`.

Gustav Sourek, Vojtech Aschenbrenner, Filip Zelezný, and Ondrej Kuzelka. Lifted relational neural networks. In *Proceedings of the NIPS Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches co-located with the 29th Annual Conference on Neural Information Processing Systems (NIPS 2015), Montreal, Canada, December 11-12, 2015.*, 2015. URL `http://ceur-ws.org/Vol-1583/CoCoNIPS_2015_paper_7.pdf`.

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2440–2448, 2015. URL `http://papers.nips.cc/paper/5846-end-to-end-memory-networks`.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014. URL `http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks`.

Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1499–1509, 2015. URL `http://aclweb.org/anthology/D/D15/D15-1174.pdf`.

Geoffrey G. Towell and Jude W. Shavlik. Knowledge-based artificial neural networks. *Artif. Intell.*, 70(1-2):119–165, 1994. doi: 10.1016/0004-3702(94)90105-8. URL `http://dx.doi.org/10.1016/0004-3702(94)90105-8`.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 2071–2080, 2016. URL `http://jmlr.org/proceedings/papers/v48/trouillon16.html`.

Ivan Vendrov, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. Order-embeddings of images and language. In *International Conference on Learning Representations (ICLR)*, 2016. URL `http://arxiv.org/abs/1511.06361`.

Patrick Verga, David Belanger, Emma Strubell, Benjamin Roth, and Andrew McCallum. Multilingual relation extraction using compositional universal schema. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 886–896, 2016a. URL `http://aclweb.org/anthology/N/N16/N16-1103.pdf`.

Patrick Verga, Arvind Neelakantan, and Andrew McCallum. Generalizing to unseen entities and entity pairs with row-less universal schema. *CoRR*, abs/1606.05804, 2016b. URL `http://arxiv.org/abs/1606.05804`.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2692–2700, 2015a. URL `http://papers.nips.cc/paper/5866-pointer-networks`.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2773–2781, 2015b. URL `http://papers.nips.cc/paper/5635-grammar-as-a-foreign-language`.

Johanna Völker and Mathias Niepert. Statistical schema induction. In *The Semantic Web: Research and Applications - 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, May 29-June 2, 2011, Proceedings, Part I*, pages 124–138, 2011. doi: 10.1007/978-3-642-21034-1_9. URL `http://dx.doi.org/10.1007/978-3-642-21034-1_9`.

Quan Wang, Bin Wang, and Li Guo. Knowledge base completion using embeddings and rules. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1859–1866, 2015. URL `http://ijcai.org/Abstract/15/264`.

William Yang Wang and William W. Cohen. Joint information extraction and reasoning: A scalable statistical relational learning approach. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 355–364, 2015. URL `http://aclweb.org/anthology/P/P15/P15-1035.pdf`.

William Yang Wang and William W. Cohen. Learning first-order logic embeddings via matrix factorization. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July*

2016, pages 2132–2138, 2016. URL `http://www.ijcai.org/Abstract/16/304`.

William Yang Wang, Kathryn Mazaitis, and William W. Cohen. Programming with personalized pagerank: a locally groundable first-order probabilistic logic. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 2129–2138, 2013. doi: 10.1145/2505515.2505573. URL `http://doi.acm.org/10.1145/2505515.2505573`.

Zhiguo Wang, Wael Hamza, and Radu Florian. Bilateral multi-perspective matching for natural language sentences. *CoRR*, abs/1702.03814, 2017. URL `http://arxiv.org/abs/1702.03814`.

Zhuoyu Wei, Jun Zhao, Kang Liu, Zhenyu Qi, Zhengya Sun, and Guanhua Tian. Large-scale knowledge base completion: Inferring via grounding network sampling over selected instances. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management, CIKM 2015, Melbourne, VIC, Australia, October 19 - 23, 2015*, pages 1331–1340, 2015. doi: 10.1145/2806416.2806513. URL `http://doi.acm.org/10.1145/2806416.2806513`.

Dirk Weissenborn. Separating answers from queries for neural reading comprehension. *CoRR*, abs/1607.03316, 2016. URL `http://arxiv.org/abs/1607.03316`.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *CoRR*, abs/1410.3916, 2014. URL `http://arxiv.org/abs/1410.3916`.

Fei Wu, Jun Song, Yi Yang, Xi Li, Zhongfei (Mark) Zhang, and Yueting Zhuang. Structured embedding via pairwise relations and long-range interactions in knowledge base. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 1663–1670, 2015. URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9342`.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille,* *France, 6-11 July 2015*, pages 2048–2057, 2015. URL http://jmlr.org/proceedings/papers/v37/xuc15.html.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *International Conference on Learning Representations (ICLR)*, 2015. URL http://arxiv.org/abs/1412.6575.

Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base completion. *CoRR*, abs/1702.08367, 2017. URL http://arxiv.org/abs/1702.08367.

Wenpeng Yin and Hinrich Schütze. Convolutional neural network for paraphrase identification. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 901–911, 2015. URL http://aclweb.org/anthology/N/N15/N15-1091.pdf.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014. URL http://arxiv.org/abs/1409.2329.

Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence, Edinburgh, Scotland, July 26-29, 2005*, pages 658–666, 2005. URL https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1209&proceeding_id=21.

Jiang Zhao, Tiantian Zhu, and Man Lan. ECNU: one stone two birds: Ensemble of heterogenous measures for semantic relatedness and textual entailment. In *Proceedings of the 8th International Workshop on Semantic Evaluation, SemEval@COLING 2014, Dublin, Ireland, August 23-24, 2014.*, pages 271–277, 2014. URL http://aclweb.org/anthology/S/S14/S14-2044.pdf.