Contextual Parameter Generation for Knowledge Graph Link Prediction

George Stoica,* Otilia Stretcu,* Emmanouil Antonios Platanios,*
Tom M. Mitchell, Barnabás Póczos
Carnegie Mellon University
5000 Forbes Ave,
Pittsburgh, Pennsylvania 15213
{gis, ostretcu, e.a.platanios, tom.mitchell, bapoczos}@cs.cmu.edu

Abstract

We consider the task of knowledge graph link prediction. Given a question consisting of a source entity and a relation (e.g., `Shakespeare` and `BornIn`), the objective is to predict the most likely answer entity (e.g., `England`). Recent approaches tackle this problem by learning entity and relation embeddings. However, they often constrain the relationship between these embeddings to be additive (i.e., the embeddings are concatenated and then processed by a sequence of linear functions and element-wise non-linearities). We show that this type of interaction significantly limits representational power. For example, such models cannot handle cases where a different projection of the source entity is used for each relation. We propose to use *contextual parameter generation* to address this limitation. More specifically, we treat relations as the *context* in which source entities are processed to produce predictions, by using relation embeddings to generate the parameters of a model operating over source entity embeddings. This allows models to represent more complex interactions between entities and relations. We apply our method on two existing link prediction methods, including the current state-of-the-art, resulting in significant performance gains and establishing a *new state-of-the-art* for this task. These gains are achieved while also *reducing convergence time by up to 28 times*.

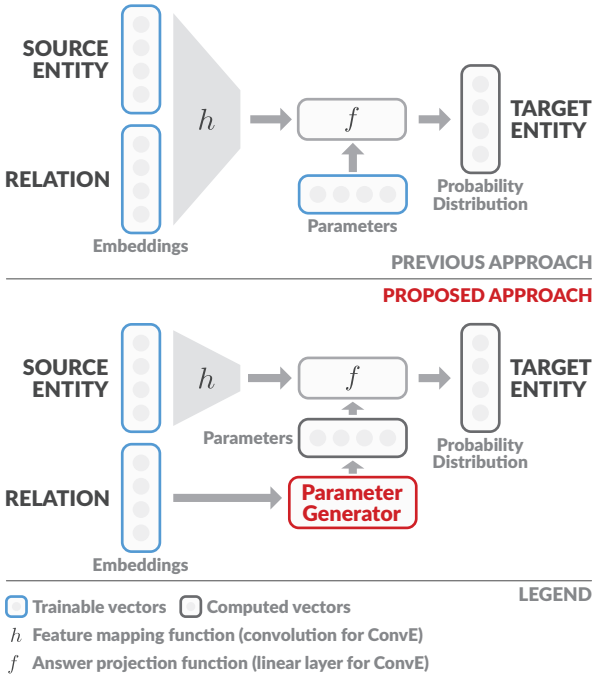

Figure 1: Overview of how our approach differs from past work. The relation is used to generate the parameters of the model that is used to transform the source entity.

1 Introduction

Many real-world applications ranging from search engines to conversational agents such as Amazon's Alexa and Apple's Siri rely on the ability to infer new facts from existing knowledge. A common means of representing such knowledge is via *knowledge graphs (KGs)*. In KGs, facts are represented by entity-relation-entity triples, (e_s, r, e_t), which encode factual relationships between graph nodes. An example triple of this form could be $(\texttt{Shakespeare}, \texttt{BornIn}, \texttt{England})$, which specifies that Shakespeare was born in England. For each triple, we refer to e_s and e_t as the source and target entities, respectively, and we refer to r as the relation between e_s and e_t.

A collection of triples is called a knowledge *graph* because the triples implicitly form a graph where entities correspond to graph nodes and relations to graph edges. There are many existing large-scale KGs, both automatically generated (e.g., the never-ending language learning system by Mitchell et al. (2018)) and human-curated (e.g., Freebase by Bollacker et al. (2008)). However, an important limitation is that they are often incomplete. For example, Freebase (Bollacker et al. 2008) is missing the place of birth for 71% of the people that exist in its graph (West et al. 2014). Despite this deficiency, many missing links are inferrable from existing knowledge in the KG. For instance, knowing who Shakespeare's parents were and where they lived could be used

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.
* Equal author contribution.

to infer the most likely place where Shakespeare was born. This motivates the task of *link prediction*, which is typically formulated as either question answering—inferring answers to questions of the form $(e_s, r, ?)$—or fact checking—evaluating the validity for statements of the form (e_s, r, e_t). While each formulation offers a different approach to link prediction, question answering can be thought of as a generalization of fact checking. This is because, in the worst case, answers can be produced by enumerating all possible entities and applying a fact checking model on each. Thus, in this paper we propose a novel method to tackle link prediction using the question answering formulation, although our core contribution can also be applied to fact-checking methods. The proposed method consistently outperforms the current state-of-the-art across multiple datasets.

The study of link prediction has gathered substantial attention in the past years, and many methods have been proposed to solve it. A significant boost in performance was observed when recent methods such as ConvE (Dettmers et al. 2018), MINERVA (Das et al. 2018), or MultiHop-KG (Lin, Socher, and Xiong 2018) combined KGs with the expressive power of neural networks. All these approaches consist of learning finite dimensional continuous vector representations (i.e., embeddings) for both the entities and the relations in the KG, and then processing them (e.g., through a neural network) in order to infer missing links in the KG. Different models process these embeddings through potentially very different types of architectures (e.g., convolutional networks or recurrent neural networks). However, they all have something in common: Entity and relation representations are combined in a way that only allows for additive interactions between them (e.g., they may be concatenated and then projected using a linear transformation). In this work, we show how this type of interaction between entities and relations significantly limits expressive power, and we propose a novel method to address this limitation. More specifically, we propose to treat the relations as the *context* in which source entities are interpreted and transformed to produce target entities. Concretely, we use the relation embeddings to generate the parameters of a model operating over entity embeddings, which then outputs a distribution over correct answers. Figure 1 shows an illustration of our method. The proposed method, **CoPER** (***Co**ntextual **P**arameters from **E**mbedded **R**elations*), has the following desirable properties:

1. **Abstract:** It can be used to enhance the representational power of several existing link prediction methods.
2. **Simple:** It can be formulated as a simple transformation for qualifying models, that can be implemented with only about 10 lines of code.
3. **Scalable:** It speeds up convergence by up to 28×.
4. **State-of-the-Art:** It outperforms competing methods by a significant margin on several established datasets.

2 Related Work

Existing approaches to perform link prediction can be classified in two categories: *single-hop* and *multi-hop* methods.

Single-Hop Methods. Given a question, single-hop methods predict answers by learning source entity and relation embeddings, and jointly transforming them to an answer entity using a fixed amount of computation. TransE (Bordes et al. 2013) produces answer entity embeddings by adding the relation and source entity embeddings together. DistMult (Yang et al. 2015) extends TransE by instead multiplying the source entity and relation embeddings element-wise. ComplEx (Trouillon et al. 2016) boosts DistMult's performance by extending the model to use complex numbers instead of real numbers, allowing for more expressive relationships between entities and relations. TransR (Lin et al. 2015) infers answer entity embeddings by first projecting the source and potential answer entity embeddings to relation space via the query relation, and then performing TransE on the resultants. CTransR (Lin et al. 2015) extends TransR by clustering distinct source and target entity pairs from triples into groups and learning distinct relation vectors for each group, thereby sharing information between group-correlated relations. TransD (Ji et al. 2015) develops on CTransR by accounting for entity types, resulting in greater method flexibility. DistMult, TransR, and CTransR empirically illustrate the benefits of multiplicative interactions between entities and relations over additive ones, and serve as an inspiration for our work. The current state-of-the-art model, ConvE (Dettmers et al. 2018), estimates a distribution over possible answers by first concatenating the source entity and relation embeddings and then feeding them through a convolutional neural network. Similar to TransE, the concatenation of entity and relation embeddings only allows for an additive interaction between the two. As we describe in Section 3.1, this significantly limits the expressive power of the model.

Multi-Hop Methods. Multi-hop approaches determine answers by finding paths connecting source entities to target entities, and consist mostly of path ranking methods (Lao, Mitchell, and Cohen 2011; Gardner et al. 2013) and neural models (Neelakantan, Roth, and McCallum 2015; Guu, Miller, and Liang 2015; Toutanova et al. 2016; Das et al. 2018; Lin, Socher, and Xiong 2018). Given a question of the form $(e_s, r, ?)$, these methods aim to find sequences of relations (e.g., (r_1, r_2, r_3)) that start at e_s and when composed, are semantically equivalent to r. NeuralLP (Yang, Yang, and Cohen 2017) learns end-to-end differentiable relation paths between source entities and targets. Similarly, NTP-λ (Rocktäschel and Riedel 2017) proposes an end-to-end differentiable backward chaining model to learn effective sequences between source entities and answers. MINERVA (Das et al. 2018) proposes a history-dependent reinforcement learning approach to KG link prediction. MultiHop-KG (Lin, Socher, and Xiong 2018) extends MINERVA by employing a pretrained single-hop method to shape the rewards used by MINERVA, which allows the model to use a more granular and informative reward policy when traversing paths. Both these latter approaches process entities and relations additively by concatenating and transforming them to obtain the next path entity. Similar to their single-hop counterparts, this limits their expressivity.

3 Background

Before describing our proposed method, we introduce the notation that we will be using for the remainder of this paper. Let e_s, r, and e_t denote one-hot encoded representations of the source entity, relation, and target entity of a KG triple. A common approach to learning abstract representations of entities and relations is to learn vector embeddings. This provides for a simple yet effective method of sharing information. The transformation from one-hot encodings to vector embeddings is modeled as follows. Let N_e and N_r denote the total number of distinct entities and relations in the KG, respectively. Given a set of entities, $E = \{e_i\}_{i=1}^{N_e}$, and a set of relations $R = \{r_i\}_{i=1}^{N_r}$, we define the following embedding matrices: $\mathbf{E} \in \mathbb{R}^{D_e \times N_e}$ and $\mathbf{R} \in \mathbb{R}^{D_r \times N_r}$, where D_e and D_r correspond to the entity and relation embedding sizes, respectively. $\mathbf{E}$ and $\mathbf{R}$ are both trainable parameters. Given a question of the form $(e_s, r, ?)$, the corresponding source entity and relation embeddings are $\mathbf{e}_s = \mathbf{E}e_s$ and $\mathbf{r} = \mathbf{R}r$, where $\mathbf{e}_s \in \mathbb{R}^{D_e}$ and $\mathbf{r} \in \mathbb{R}^{D_r}$, respectively.

Multiple existing single-hop link prediction methods (as well as a single hop in multi-hop methods) can be described in terms of the following abstract model:

$$\mathbf{e}_s = \mathbf{E}e_s, \quad \text{(embedding)} \quad (1)$$

$$\mathbf{r} = \mathbf{R}r, \quad \text{(embedding)} \quad (2)$$

$$\mathbf{z} = h_\phi(\mathbf{e}_s, \mathbf{r}, ...), \quad \text{(merge)} \quad (3)$$

$$\mathbf{ans} = f_\theta(\mathbf{z}, ...), \quad \text{(prediction)} \quad (4)$$

where $\mathbf{z}$ is a latent representation of the merged entity and relation embeddings. The merge is performed using the *merging function* h, parameterized by ϕ. Then, the answer $\mathbf{ans}$ is predicted from $\mathbf{z}$ using the *prediction function* f, which is parameterized by θ. Depending on the model, $\mathbf{ans}$ can be the predicted embedding of the target entity, $\hat{e}_t$, a probability distribution over target entities, or the probability that the fact (e_s, r, e_t) is true. Note that the functions h_ϕ and f_θ may also take other model-dependent arguments, such as e_t, which we denote through "...". In fact, in recent methods that have shown improved link prediction accuracies (ConvE, MINERVA, Multihop-KG), f_θ and h_ϕ are neural networks consisting of convolution and/or recurrent layers. Figure 1, top, shows an illustration of this abstract architecture.

While, multiple existing link prediction methods fit under this formulation, we will use ConvE (Dettmers et al. 2018) as a running example, since it is both the current state-of-the-art and one of the baseline methods used in our experiments. In ConvE, we have:

$$\mathbf{z} = \text{Conv2D}(\text{Reshape}([\mathbf{e}_s; \mathbf{r}])), \quad \text{(merge)} \quad (5)$$

$$\hat{e}_t = f_\theta(\mathbf{z}), \quad \text{(prediction)} \quad (6)$$

where $[\mathbf{e}_s; \mathbf{r}] \in \mathbb{R}^{D_e + D_r}$ represents the result of stacking the entity and relation embeddings together, followed by a reshape into a $W \times H$ rectangular matrix, where W and H are model hyperparameters such that $WH = D_e + D_r$. This matrix is then passed through a 2D convolution layer to obtain the merged representation $\mathbf{z}$. The prediction function f_θ is defined as a linear layer, where θ denotes its weight matrix and bias vector combined, followed by a dropout layer. An illustration of the ConvE model is shown in the top left quadrant of Figure 3. For further details, we refer the reader to the work of Dettmers et al. (2018). Given that h_ϕ may also include other entity-relation merge operations, more complex models such as MINERVA or MultiHop-KG can also be expressed in terms of this abstraction.

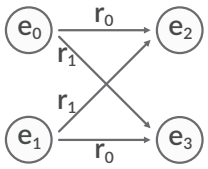

Figure 2: Toy example that cannot be modeled by additive interactions between entities and relations.

3.1 Limited Expressive Power

Most existing neural methods consist of the following steps: (i) learn entity and relation embeddings, (ii) concatenate the source entity and relation embeddings, and (iii) perform a sequence of linear transformations and element-wise non-linear operations on them (as shown in equations 1-4), in order to obtain the target entities. We now use an example to explain why this only explicitly allows for additive interactions between the source entity and relation, and why this is significantly limiting the model's expressive power. Consider a simple merging function where the source entity and relation are first concatenated into a single vector as $[\mathbf{e}_s; \mathbf{r}]$, and then projected through a linear layer:

$$h_\phi(\mathbf{e}_s, \mathbf{r}) = \phi \cdot [\mathbf{e}_s; \mathbf{r}], \quad (7)$$

where $\phi \in \mathbb{R}^{D_z \times (D_e + D_r)}$ and D_z is the size of $\mathbf{z}$. If we refer to the first D_e columns of ϕ as ϕ_e, and the last D_r columns as ϕ_r (i.e., $\phi = [\phi_e; \phi_r]$), then we can write $h_\phi(\mathbf{e}_s, \mathbf{r}) = \phi_s \mathbf{e}_s + \phi_r \mathbf{r}$. Note that, in this case the elements of the entity embedding $\mathbf{e}_s$ and the relation embedding $\mathbf{r}$ only interact in an additive way (i.e., the output $\mathbf{z}$ is a linear combination of the elements in $\mathbf{e}_s$ and $\mathbf{r}$ and it does not support more complex interactions, such as multiplicative or polynomial). The same is true for the merging function in ConvE through Conv2D (only some of the elements of ϕ are shared), as well as the merging functions of MINERVA and Multihop-KG (further explained in Section 4.3). The main implication of this is that relations cannot influence the projection matrices used to transform the entities.

Let us demonstrate this important limitation by considering the example shown in Figure 2, illustrating 4 KG facts: $(\mathtt{e}_0, \mathtt{r}_0, \mathtt{e}_2)$, $(\mathtt{e}_0, \mathtt{r}_1, \mathtt{e}_3)$, $(\mathtt{e}_1, \mathtt{r}_0, \mathtt{e}_3)$, $(\mathtt{e}_1, \mathtt{r}_1, \mathtt{e}_2)$. Suppose we want to encode these facts using a model in the form of Equation 7:

$$\mathbf{e}_2 = \phi_e \mathbf{e}_0 + \phi_r \mathbf{r}_0, \quad (8)$$

$$\mathbf{e}_3 = \phi_e \mathbf{e}_0 + \phi_r \mathbf{r}_1, \quad (9)$$

$$\mathbf{e}_3 = \phi_e \mathbf{e}_1 + \phi_r \mathbf{r}_0, \quad (10)$$

$$\mathbf{e}_2 = \phi_e \mathbf{e}_1 + \phi_r \mathbf{r}_1. \quad (11)$$

Subtracting (9) from (8), and (11) from (10), we have that:

$$(\mathbf{e}_2 - \mathbf{e}_3) = \phi_r(\mathbf{r}_0 - \mathbf{r}_1), \quad (12)$$

$$(\mathbf{e}_3 - \mathbf{e}_2) = \phi_r(\mathbf{r}_0 - \mathbf{r}_1), \quad (13)$$

which leads to a degenerate solution where (i) $\mathbf{e}_3 = \mathbf{e}_2$, and (ii) $\phi_r = 0$ or $\mathbf{r}_0 = \mathbf{r}_1$. Note that if instead we subtract (10) from (8), and (11) from (9), we achieve similar degenerate solutions that: (i) $\mathbf{e}_2 = \mathbf{e}_3$, and (ii) $\phi_e = 0$ or $\mathbf{e}_1 = \mathbf{e}_0$. This implies that additive models cannot handle this toy example.

While this is a toy example, it illustrates a more general problem. For instance, consider a case where we want to learn a different expert model for each relation. This means that given a source entity and relation, the relation determines which expert to use when processing the source entity. This example is important because related work in other areas has shown that mixtures of experts—our toy example is in fact a very simple form of a mixture of experts—can result in significant performance gains (Lengerich, Maas, and Potts 2018). Methods that combine entities and relations additively cannot learn such a mixtures of experts. Ideally, we want our model to be expressive enough such that it can learn functions that are conditional on the relation, such as the above mixture of experts example. That example is important because learning a separate model for each relation may sometimes be impossible. A common pattern for certain KGs is that for some relations we have a lot of training data, but for most we have very little. In such cases, we want to be able to leverage the fact that many relations are similar by sharing information among them. Note also that increased expressivity alone is not sufficient as it can result in overfitting. We propose to use *contextual parameter generation*, originally proposed in the context of neural machine translation by Platanios et al. (2018), which allows us to increase expressivity in a manner that, as we show using an extensive empirical evaluation, is useful to the link prediction problem. Importantly, as we show in Section 4.2 the proposed approach is able to handle the aforementioned toy example as well as more general mixtures of experts.

4 Proposed Method

In this section, we propose a new approach that addresses the limitations regarding additive interactions raised in the previous section. Our method, termed **CoPER** — *Contextual Parameters from Embedded Relations*, can be used to enhance multiple existing additive link prediction methods, by enabling them to learn more expressive relationships between entities and relations. At the core of CoPER lies the key idea that relations define *how* source entities are processed in order to produce answer entities. Specifically, when answering a question $(e_s, r, ?)$, the target entity e_t can be obtained through a transformation of the source entity e_s, and the *parameters* of this transformation are determined by the relation r.

In Figure 1, we show how a baseline model, expressed using Equations 1-4, can be transformed using CoPER. In this baseline model, the embeddings of e_s and r are merged through the additive operation h (e.g., concatenation followed by convolution), and then transformed using f (e.g., a neural network) whose parameters are *learned* (e.g., through backpropagation). In CoPER, operation h is only applied to $\mathbf{e}_s$, while $\mathbf{r}$ is now used to *generate* the parameters of f. Thus, the parameters of f are no longer learned directly, but are rather the output of a new model component—the *contextual parameter generator (CPG)*. In the following section, we propose and compare different potential architectures for the CPG module. We then explain how the proposed modification can have a large impact in the kinds of KG relationships our models can represent.

4.1 Parameter Generator Network

The contextual parameter generation (CPG) module is a function that takes as input a relation r and outputs the parameters θ of some other function f. Let $g\colon \{1, 2, ..., N_r\} \rightarrow \mathbb{R}^{D_\theta}$ be our parameter generation function, where N_r is the number of relations in the KG, and $\theta \in \mathbb{R}^{D_\theta}$. We now present three simple functional forms for g that we also use for our experiments.

Parameter Lookup Table. The simplest approach is to output an entirely different θ for each relation. This results in the following form:

$$g_{\text{lookup}}(r) = \mathbf{W}_{\text{lookup}} r, \quad (14)$$

where r here is a one-hot encoded vector representation of the relation, and $\mathbf{W}_{\text{lookup}} \in \mathbb{R}^{D_\theta \times N_r}$ is the only learnable parameter of g_{lookup}. Interestingly, this is comparable to DistMult and TransR, as each of these methods also uses relations to define distinct mappings over entity embeddings. However, the problem with this simple formulation is that information sharing across relations can only happen through the shared entity embeddings. This makes the model prone to overfitting, especially for relations which have limited training data. Moreover, in certain KGs, many of the relations may be similar (e.g., `bornIn` and `livesIn`), and it may be beneficial for them to share information. This motivates a different approach for generating parameters.

Linear Projection. Instead of using one-hot representations for the relations, we can instead learn embeddings:

$$g_{\text{linear}}(r) = \mathbf{W}_{\text{linear}} \mathbf{R} r + \mathbf{b}, \quad (15)$$

where we use the embedding lookup equation, $\mathbf{W}_{\text{linear}} \in \mathbb{R}^{D_\theta \times D_r}$, bias term $\mathbf{b} \in \mathbb{R}^{D_\theta}$, D_r is the relation embedding size, and both $\mathbf{W}_{\text{linear}}$ and $\mathbf{R}$, are trainable model parameters. Intuitively, the learned relation embeddings represent a linear combination of D_r different values for θ, allowing for shared information between relations.

Multi-Layer Perceptron. Most of our experiments are performed using g_{linear}, with which we achieve state-of-the-art results. However, we observed that g_{linear} may underperform in small datasets. We argue that the most likely reason is due to $\mathbf{W}_{\text{linear}}$ becoming too big relative to the original number of parameters. This is because, if we originally had D_θ trainable variables, g_{linear} now has $D_\theta \times D_r$ parameters, which is significantly larger. Limiting the value of D_r is not necessarily a solution as a small D_r can significantly constrain the capacity of our model. We therefore propose

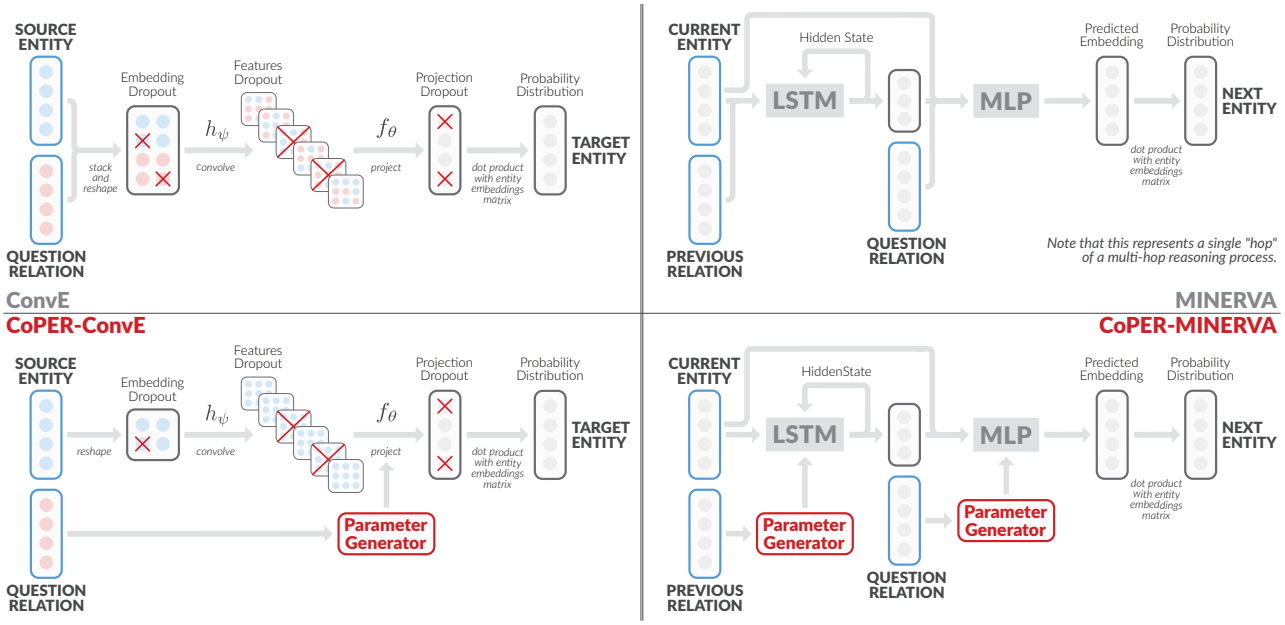

Figure 3: Overview of ConvE (top left), MINERVA (top right) and their CoPER-powered versions (bottom).

a third variant of the generator network using a multi-layer perceptron:

$$g_{\text{MLP}}(r) = \text{MLP}(\mathbf{R}r). \tag{16}$$

This can be thought of as a low-rank approximation to g_{linear}.

These are only three possible proposals for the parameter generator network and, as we show next, even a simple network such as g_{linear} already significantly increases representational power. Note however, that the idea behind CoPER is more general and can be extended to more complex architectures. Moreover, in contrast to CTransR and TransD which also learn correlations between relations, CPG learns *unrestricted* relationships between them: based on the choice of CPG module, the network can learn any arbitrary relation interactions. Furthermore, its abstract design enables it to fully benefit from the expressive power of neural networks.

4.2 Enhanced Expressive Power

Through the parameter generation component, CoPER enables link prediction methods to directly model more complex interactions between the entity and relation embeddings. A CPG module as simple as g_{linear}, combined with any typical neural network architecture for f_θ (from a single linear layer to many complex layers followed by element-wise non-linearities) allows the model to represent *multiplicative* interactions between source entities and relations.

For ease of explanation, we will illustrate this increase in representation power with a simple form of h_ϕ and f_θ, but it is easy to see that more complex architectures can only increase the expressive power further. According to the CoPER formulation, h_ϕ now only operates on $\mathbf{e}_s$, preprocessing the source entity embedding before passing through the predictor function f_θ. For simplicity, we can assume no preprocessing is necessary, so $h_\phi(\mathbf{e}_s) = \mathbf{e}_s$, and that f is a simple linear projection, $f_\theta(x) = \theta x$. The parameters θ are given by $\theta = g_{\text{linear}}(r) = \mathbf{W}\mathbf{R}r + \mathbf{b} = \mathbf{W}\mathbf{r} + \mathbf{b}$. Thus:

$$\hat{e}_t = f_\theta(h_\phi(\mathbf{e}_s)) = f_\theta(\mathbf{e}_s) = \theta\mathbf{e}_s = (\mathbf{W}\mathbf{r} + \mathbf{b})\mathbf{e}_s. \tag{17}$$

This result shows that the relation and entity embedding now interact in a multiplicative way, which means the relation itself affects the weights with which we multiply the entity embedding. This is more expressive than an additive interaction, as it now allows us to represent dependencies such as conditionals (i.e., if statements), mixtures of experts, and even the toy example we present in Figure 2.

Toy Example. Going back to our toy example that additive interactions cannot represent, we now show that a CPG module as simple as g_{linear} or g_{lookup} can encode this KG example. Applying the predictor derived in Equation 17 to the KG in Figure 2, the following equations must hold for the toy example to be representable by the model:

$$\mathbf{e}_2 = (\mathbf{W}\mathbf{r}_0 + \mathbf{b})\mathbf{e}_0, \tag{18}$$
$$\mathbf{e}_3 = (\mathbf{W}\mathbf{r}_0 + \mathbf{b})\mathbf{e}_1, \tag{19}$$
$$\mathbf{e}_2 = (\mathbf{W}\mathbf{r}_1 + \mathbf{b})\mathbf{e}_1, \tag{20}$$
$$\mathbf{e}_3 = (\mathbf{W}\mathbf{r}_1 + \mathbf{b})\mathbf{e}_0. \tag{21}$$

Subtracting (18) from (19), and (20) from (21), we have:

$$\mathbf{e}_3 - \mathbf{e}_2 = (\mathbf{W}\mathbf{r}_0 + \mathbf{b})(\mathbf{e}_1 - \mathbf{e}_0), \tag{22}$$
$$\mathbf{e}_3 - \mathbf{e}_2 = (\mathbf{W}\mathbf{r}_1 + \mathbf{b})(\mathbf{e}_0 - \mathbf{e}_1). \tag{23}$$

Avoiding the degenerate solution where $\mathbf{e}_0 = \mathbf{e}_1$, we have:

$$\mathbf{W}(\mathbf{e}_1 - \mathbf{e}_0)(\mathbf{r}_0 + \mathbf{r}_1) + 2\mathbf{b}(\mathbf{e}_1 - \mathbf{e}_0) = 0. \tag{24}$$

This equation has an infinite number of solutions. Note that although we showed here that CPG leads to multiplicative

interactions between $\mathbf{e}_s$ and $\mathbf{r}$ for a particular choice of $f_\theta(x) = \theta x$, the conclusions will stand for most neural network architectures, from multilayer perceprons to convolutional to recurrent neural networks, since they usually involve such a projection step on the inputs.

4.3 CoPER and State-of-the-Art Models

We discussed how CoPER can generally be used to extend link prediction models with additive interactions. We will now show how it can be applied to two specific models, ConvE and MINERVA, which are representative for the two main lines of recent neural link prediction methods: single-hop and multi-hop methods. While these models may have distinct complex architectures with multiple types of neural network layers, each integrates entities and relations additively in several key components of their networks. In place of each of these interactions, we substitute our CPG module to alleviate the limitations induced by additive integration. Figure 3 shows a parallel between the original ConvE and MINERVA, and their CoPER-enhanced versions, CoPER-ConvE and CoPER-MINERVA, respectively.

ConvE. The original ConvE model can be described in terms of our abstract framework as shown in Equations 5-6. In CoPER-ConvE, the first preprocessing steps in the pipeline (reshape, convolution) are only applied to the entity embedding, while the relation is used to generate the parameters of the projection layer:

$$\begin{aligned}\mathbf{z} &= \text{Conv2D}(\text{Reshape}(\mathbf{e}_s)),\\ \theta &= g(r),\\ \hat{e}_t &= f_\theta(\mathbf{z}) = \theta_1 + \theta_{2:D_\theta}\mathbf{z},\end{aligned}$$

where $\theta = [\theta_1; \theta_2]$ is the parameter vector produced by the parameter generator.

MINERVA. MINERVA is a deterministic RL-based multi-hop question-answering agent, which means that it will answer the question $(e_s, r, ?)$ by finding a path in the graph that connects e_s with the predicted answer $\hat{e}_t$. The model defines states as the entities in the KG, and actions as tuples (r, e) consisting of an outgoing relation and its destination entity, specifying a hop to a neighboring node in the KG. Given a question (e_s, r_q, e_t), MINERVA traverses the KG along its relations from e_s to the most likely target entity e_t. Each step along the graph path iteratively accumulates a history of entities and relations visited, which is aggregated together through and then stored in a Long Short-Term Memory (LSTM) network (Hochreiter and Schmidhuber 1997), as illustrated in Figure 3 (right). The hidden state of the LSTM is updated as follows:

$$\mathbf{h}_i = \text{LSTM}(\mathbf{h}_{i-1}, [\mathbf{e}_i; \mathbf{r}_{i-1}]), \qquad \text{(merge)}$$

where $\mathbf{h}_i$ denotes the accumulated history representation at the i^{th} time step, $\mathbf{h}_{i-1}$ is the history representation at the previous step and is the hidden state of the LSTM, $\mathbf{r}_{i-1}$ denotes the embedding representation of the relation taken in the previous step leading to state e_i (represented by the embedding $\mathbf{e}_i$), and $[;]$ represents vector concatenation. Additionally, $[\mathbf{e}_i; \mathbf{r}_{i-1}]$ denotes the LSTM input. Because the LSTM module consists of a series of input projections, $\mathbf{e}_i$ and $\mathbf{r}_{i-1}$ are *additively* incorporated into the agent's history.

At every time step, once the traversal history has been accumulated, the agent next determines the subsequent action to take as follows:

$$\mathbf{o}_i = \text{MLP}([\mathbf{h}_i; \mathbf{e}_i; \mathbf{r}_q]), \qquad \text{(merge)} \tag{25}$$

$$a_j = \text{Categorical}(\mathbf{A}_i \mathbf{o}_i), \qquad \text{(prediction)} \tag{26}$$

where $\mathbf{A_i}$ denotes the embedding representations of each available action from e_i, $\mathbf{o}_i$ represents the Multi-Layer Perception (MLP) output, Categorical denotes a categorical distribution decision function—such as a network policy—which operates over action distribution logits given by $\mathbf{A}_i\mathbf{o}_i$, and a_j is the selected action. Since an action is a tuple (r, e) as explained above, we represent a_j as the concatenation of the respective relation embedding and an entity embedding. $\mathbf{A}_i$ denotes then the matrix containing vector representations of all available actions from each state. Importantly, we observe that the entity and relation embeddings are concatenated as input to the MLP, which also induces an *additive* interaction between them in this component. Thus, in both components where entity and relation information is processed in MINERVA, it is done additively. As illustrated by our toy example, this limits the expressivity of the network.

In CoPER-MINERVA, we replace these additive steps with parameter generators, as illustrated in the right of Figure 3. In the fist case, the embedding of the previous relation in a step, $\mathbf{r}_{i-1}$, is used as input to a parameter generator that outputs the parameters of the LSTM component. In the second case, the query relation $\mathbf{r}$ embedding is used to generate the parameters of the MLP, which operates over the step history and current entity representations. The rest of the model remains unchanged.

5 Experiments

In this section, we empirically evaluate the performance of CoPER on several established link-prediction datasets.

Datasets. We adopt the following datasets used in prior literature: Unified Medical Language Systems (*UMLS*) (Kok and Domingos 2007), Alyawarra *Kinship*, *WN18RR* (Dettmers et al. 2018), *FB15k-237* (Toutanova and Chen 2015), and NELL-995 (Xiong, Hoang, and Wang 2017). Table 2 displays summary statistics for each dataset. To keep our train/validation/test dataset partitions consistent with those of prior literature and ensure fair comparisons, we use the published datasets from Das et al. (2018) and Lin, Socher, and Xiong (2018). Similar to prior work, we augment our training data with inverse relations (for each example (e_s, r, e_t) we introduce (e_t, r^{-1}, e_s)).

Metrics. We report results for two metrics used throughout prior work: `Hits@k` and Mean Reciprocal Rank (`MRR`). Both assess how a model ranks the correct answer compared to all other possible answers. `Hits@k`, also known as recall-at-k, is defined as the proportion of times the correct answer is ranked among the top-*k* answers, according to the probabilities assigned by the model. Similar to prior work, we report the average `Hits@1` and `Hits@10` over the test

Dataset	Metric	Models								
		DistMult	ComplEx	NeuralLP	NTP-λ	MINERVA	MultiHop-KG	ConvE	CoPER-MINERVA	CoPER-ConvE
UMLS	Hits@1	82.1	82.3	64.3	84.3	75.3	90.2	92.89	77.76†	**95.46**‡
	Hits@10	96.7	99.5	96.2	**100.0**	96.7	99.2	99.70	97.43†	99.70‡
	MRR	86.8	89.4	77.8	91.2	84.1	94.0	95.35	85.44†	**97.08**‡
Kinship	Hits@1	48.7	75.4	47.5	75.9	60.5	78.9	74.21	66.20†	**83.62**†
	Hits@10	90.4	98.0	91.2	87.8	92.4	98.2	97.86	94.23†	**98.42**†
	MRR	61.4	83.8	61.9	79.3	72.0	86.5	83.04	76.00†	**89.52**†
WN18RR	Hits@1	43.1	41.0	37.6	–	41.3	41.8	41.86	42.66†	**44.05**†
	Hits@10	52.4	51.0	**65.7**	–	51.3	51.7	52.17	50.99†	56.12†
	MRR	46.2	44.0	46.3	–	44.8	45.0	45.19	46.51†	**48.33**†
FB15k237	Hits@1	32.4	15.8	16.6	–	22.3	**32.7**	30.30	29.49†	32.18†
	Hits@10	60.0	42.8	34.8	–	44.9	56.4	60.83	50.39†	**62.92**†
	MRR	41.7	24.7	22.7	–	29.2	40.7	40.51	36.51†	**42.56**†
NELL-995	Hits@1	55.2	64.3	–	–	63.96	65.6	67.04	65.52†	**72.15**†
	Hits@10	78.3	86.0	–	–	82.35	84.4	87.96	83.24†	**88.35**†
	MRR	64.1	72.6	–	–	70.97	72.7	75.42	72.46†	**78.68**†

Table 1: Results for various link prediction models. Results for ConvE, MINERVA, CoPER-ConvE and CoPER-MINERVA are reported according to our own experiments. The remainder are taken from Das et al. (2018). All numbers are expressed as percentages. † denotes experiments performed using g_{linear}, and ‡ denotes those performed using g_{MLP}. "–" denotes missing results from the respective publications. Note that for MINERVA, we used the implementation provided by Lin, Socher, and Xiong (2018), and report our results with the provided implementation for fair comparison with our CoPER extension.

Dataset	# Train	N_e	N_r	$\bar{N}_a$	$\bar{d}$
Kinship	8,544	104	25	6.14	82.15
UMLS	5,216	135	46	7.83	26.59
FB15k237	272,115	14,541	237	3.03	17.87
WN18RR	86,835	40,945	11	1.41	2.19
NELL-995	154,213	75,492	200	3.57	4.07

Table 2: Dataset statistics. Here, # Train denotes the number of questions used for training, N_e the number of distinct entities, N_r the number of distinct relations, $\bar{N}_a$ the average number of answers per question, and $\bar{d}$ the average degree of the graph nodes in the dataset.

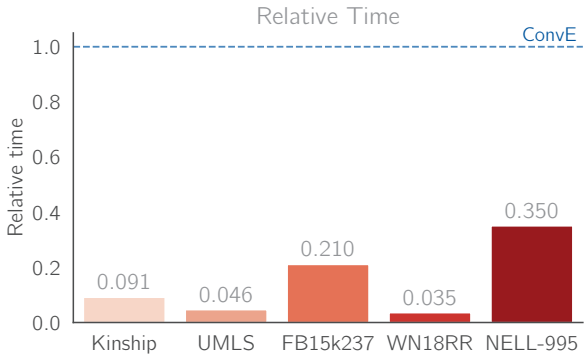

Figure 4: Time required for CoPER-ConvE to obtain its best performance on each dataset, as a fraction of the time it takes ConvE to achieve equivalent performance.

set. MRR is defined as the average value of the reciprocated rank of the correct answer for each test instance. Therefore, MRR is a measure of the overall quality of a model's predictions. Note that this evaluation method is also used in ConvE, MINERVA, and MultiHop-KG.

Models. We evaluate CoPER-ConvE and CoPER-MINERVA against their base models, ConvE and MINERVA, and multiple other link prediction methods. To ensure a fair comparison between CoPER-ConvE and its unaltered baseline, we re-implement ConvE in our environment, and retain the hyperparameters originally reported by Dettmers et al. (2018). In fact, our implementation either matches or improves upon the previously published results (possibly due to the use of negative sampling, which we cover in our supplemental material. This can be accessed from Section 6). For CoPER-MINERVA, we construct our CoPER framework within the MINERVA implementation provided by Lin, Socher, and Xiong (2018). For both CoPER extensions, we vary the relation embedding size (originally 200) based on the number of relations in each dataset, which stems from our observations that datasets with few relations (e.g., Kinship or WN18RR) perform better with smaller embeddings. We choose the dropout parameters by performing a grid search between [0,1] based on the validation set Hits@1. Regarding the parameter generation module, we perform experiments using both g_{linear} and g_{MLP}. For the MLP, we use a single hidden layer with a ReLU activation and chose the number of hidden units by also performing a grid search between. We train our models using the binary cross-entropy loss function. For each positive training example, we sample 10 negatives as described in our supplemental material (reached from Section 6), and use a label smoothing factor of 0.1. All hyperparameter values and CPG architectures utilized in our experiments can be found in our repository

at https://github.com/otiliastr/coper. We conduct all our experiments on a single Nvidia TitanX GPU.

Results. Our overall performance results are reported in Table 1. We observe that CoPER-ConvE outperforms ConvE on all datasets, with up to $+9.41\%$ Hits@1 performance gain over ConvE on Kinship. Moreover, we find that CoPER-ConvE achieves superior performance over all other existing methods on these datasets, often by a significant margin. Notably, we observe a +4.7% Hits@1 gain for Kinship over the best existing method and a +4.09% Hits@1 gain for NELL-995. To the best of our knowledge, CoPER-ConvE establishes a *new state-of-the-art*.

We also examine the effect of CoPER on *training time*. Since CoPER-ConvE is the version with the best results across all datasets, we perform this analysis for ConvE and CoPER-ConvE. Given that CoPER-ConvE consistently outperforms ConvE in terms of Hits@1 we compare the number of iterations that each method requires to reach the best Hits@1 value that ConvE achieves (e.g., we check when both ConvE and CoPER reach 92.89% Hits@1 on UMLS). Then we calculate the ratio: $\frac{\text{\# iterations CoPER}}{\text{\# iterations ConvE}}$. For instance, if a baseline model requires 10,000 steps to attain best performance, while its CoPER variant takes 3,000 steps to achieve identical performance, then our metric would be: $\frac{3{,}000}{10{,}000} = 0.3$. Analogously, this would denote a $\frac{1}{0.3} = 3.33$ factor of training speed gain. Our results, illustrated in Figure 4, show that CoPER-ConvE always requires much fewer training iterations than ConvE, yielding a speedup of between 2.9 to 28.6 times.

Recall that in Section 4.1 we proposed to use g_{linear} and g_{MLP} for the parameter generation network, instead of the arguably more straightforward parameter lookup table. We motivated this by suggesting that using g_{lookup} would more likely result in overfitting, especially in cases where there is too little training data per relation. To examine the impact of relation information sharing through the contextual parameter generator, we conduct an experiment comparing our best performing g_{lookup} CoPER models against our best performing g_{linear} or g_{MLP} methods. As g_{lookup} is an example of a generator which does not enable this kind of information sharing, pitting it against these latter generators enables to explicitly analyze the importance of information sharing through the generator across our benchmark datasets. Table 3 illustrates our results from these experiments, with the addition of ConvE for reference. In the table, g_{lookup} is referenced by "CoPER-PL", while "CoPER" on its own refers to best performing g_{linear} or g_{MLP} model. g_{linear} models are referenced by †, while g_{MLP} generators are denoted by ‡. Note also that training CoPER-PL-ConvE on NELL-995 is infeasible with our resource capabilities due to the memory required to store the parameter lookup table and entity embeddings.

Based on these experiments, we observe a strong correlation with performance disparity between CoPER-PL and CoPER and dataset size and sparsity (please refer to Table 2 for dataset statistics). These results suggest that as datasets become smaller and denser, generator functions such as g_{linear} and g_{MLP} are critical to maintaining strong results. Moreover, while the performance discrepancy between CoPER-PL and CoPER is less pronounced in larger and sparser datasets, we observe that sharing information through the generator is still important to performance.

Dataset	Metric	Models		
		ConvE	CoPER-PL-ConvE	CoPER-ConvE
UMLS	Hits@1	92.89	73.82	**95.46**‡
	Hits@10	**99.70**	99.09	**99.70**‡
	MRR	95.35	85.17	**97.08**‡
Kinship	Hits@1	74.21	74.90	**83.62**†
	Hits@10	97.86	96.63	**98.42**†
	MRR	83.04	83.22	**89.52**†
WN18RR	Hits@1	41.86	**44.10**	44.05†
	Hits@10	52.17	51.20	**56.12**†
	MRR	45.19	46.63	**48.33**†
FB15k237	Hits@1	30.30	30.72	**32.18**†
	Hits@10	60.83	60.04	**62.92**†
	MRR	40.51	40.52	**42.56**†
NELL-995	Hits@1	67.04	N/A	**72.15**†
	Hits@10	87.96	N/A	**88.34**†
	MRR	75.42	N/A	**78.68**†

Table 3: Overview of our ablation testing performances on our evaluation datasets. Results for ConvE, CoPER-PL-ConvE (using a parameter lookup generator function), and CoPER-ConvE are reported according to our own experiments. All numbers are expressed as percentages. † refers to g_{linear} generators, while ‡ refers to g_{MLP} generators. "N/A" denotes experiments outside our computational resource capabilities.

We also compare CoPER with TransR and TransD, which are simple models that allow for multiplicative interactions. Further details and results can be found our supplemental material under Section 6. CoPER consistently outperforms the other methods. Finally, visualizations showcasing the resultant relation similarities from FB15k-237 and NELL-995 our CoPER models learn can also be accessed by Section 6.

6 Supplementary Material

All supplementary material along with code to reproduce our experiments can be accessed at: https://github.com/otiliastr/coper.

7 Conclusion

We proposed CoPER, a novel framework that improves upon the current state-of-the-art methods for the task of knowledge graph link prediction. CoPER treats relations as the context in which source entities are processed to predict target entities. We showed how this significantly increases the expressive power of link prediction models by allowing them to represent multiplicative interactions between entities and relations. We also exhibited our approach's flexibility by extending both a single-hop and a multi-hop link prediction model, achieving new state-of-the-art performance for this task, while significantly speeding up convergence time over unaltered methods by up to $28\times$.

8 Acknowledgements

This material is based upon work supported by AFOSR FA95501710218, NSF IIS1563887, DARPA/AFRL FA87501720130 and Lockheed Martin. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Air Force Office of Scientific Research, the National Science Foundation, the Defense Advanced Research Projects Agency, Air Force Research Laboratory, or Lockheed Martin.

References

Bollacker, K.; Evans, C.; Paritosh, P.; Sturge, T.; and Taylor, J. 2008. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, 1247–1250. ACM.

Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*, 2787–2795.

Das, R.; Dhuliawala, S.; Zaheer, M.; Vilnis, L.; Durugkar, I.; Krishnamurthy, A.; Smola, A.; and McCallum, A. 2018. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

Dettmers, T.; Pasquale, M.; Pontus, S.; and Riedel, S. 2018. Convolutional 2d knowledge graph embeddings. In *Proceedings of the 32th AAAI Conference on Artificial Intelligence*, 1811–1818.

Gardner, M.; Talukdar, P. P.; Kisiel, B.; and Mitchell, T. 2013. 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*, 833–838.

Guu, K.; Miller, J.; and Liang, P. 2015. Traversing knowledge graphs in vector space. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 318–327.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural Comput.* 9(8):1735–1780.

Ji, G.; He, S.; Xu, L.; Liu, K.; and Zhao, J. 2015. Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 687–696.

Kok, S., and Domingos, P. 2007. Statistical predicate invention. In *Proceedings of the 24th International Conference on Machine Learning*, 433–440.

Lao, N.; Mitchell, T.; and Cohen, W. W. 2011. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 529–539. Association for Computational Linguistics.

Lengerich, B.; Maas, A.; and Potts, C. 2018. Retrofitting Distributional Embeddings to Knowledge Graphs with Functional Relations. In *Proceedings of the 27th International Conference on Computational Linguistics*, 2423–2436. Santa Fe, New Mexico, USA: Association for Computational Linguistics.

Lin, Y.; Liu, Z.; Sun, M.; Liu, Y.; and Zhu, X. 2015. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, 2181–2187. AAAI Press.

Lin, X. V.; Socher, R.; and Xiong, C. 2018. Multi-hop knowledge graph reasoning with reward shaping. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 3243–3253.

Mitchell, T.; Cohen, W.; Hruschka, E.; Talukdar, P.; Yang, B.; Betteridge, J.; Carlson, A.; Dalvi, B.; Gardner, M.; Kisiel, B.; et al. 2018. Never-Ending Learning. *Communications of the ACM* 61(5):103–115.

Neelakantan, A.; Roth, B.; and McCallum, A. 2015. Compositional vector space models for knowledge base completion. In *ACL*.

Platanios, E. A.; Sachan, M.; Neubig, G.; and Mitchell, T. 2018. Contextual Parameter Generation for Universal Neural Machine Translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Rocktäschel, T., and Riedel, S. 2017. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems*, 3788–3800.

Toutanova, K., and Chen, D. 2015. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, 57–66.

Toutanova, K.; Lin, V.; Yih, W.-t.; Poon, H.; and Quirk, C. 2016. Compositional learning of embeddings for relation paths in knowledge base and text. In *ACL*.

Trouillon, T.; Welbl, J.; Riedel, S.; Gaussier, E.; and Bouchard, G. 2016. Complex embeddings for simple link prediction. In *International Conference on Machine Learning (ICML)*, volume 48, 2071–2080.

West, R.; Gabrilovich, E.; Murphy, K.; Sun, S.; Gupta, R.; and Lin, D. 2014. Knowledge Base Completion via Search-Based Question Answering. In *WWW*.

Xiong, W.; Hoang, T.; and Wang, W. Y. 2017. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 564–573. Association for Computational Linguistics.

Yang, B.; Yih, W.-t.; He, X.; Gao, J.; and Deng, L. 2015. Embedding entities and relations for learning and inference in knowledge bases. In *International Conference on Learning Representations (ICLR)*.

Yang, F.; Yang, Z.; and Cohen, W. W. 2017. Differentiable learning of logical rules for knowledge base reasoning. In *Advances in Neural Information Processing Systems*, 2319–2328.