EPiC Series in Computing
Volume 72, 2020, Pages 107–120
GCAI 2020. 6th Global Conference on Artificial Intelligence (GCAI 2020)

Topical Neural Theorem Prover that Induces Rules

Shuang Xia, Krysia Broda, and Alessandra Russo

Imperial College London, London, U.K.
{shuang.xia13, k.broda, a.russo}@imperial.ac.uk

Abstract

Various sub-symbolic approaches for reasoning and learning have been proposed in the literature. Among these approaches, the neural theorem prover (*NTP*) approach uses a backward chaining reasoning mechanism to guide a machine learning architecture to learn vector embedding representations of predicates and to induce first-order clauses from a given knowledge base. *NTP* is however known for being not scalable, as the computation trees generated by the backward chaining process can grow exponentially with the size of the given knowledge base. In this paper we address this limitation by extending the *NTP* approach with a topic-based method for controlling the induction of first-order clauses. Our proposed approach, called *TNTP* for *Topical NTP*, identifies topic-based clusters over a large knowledge-base and uses these clusters to control the soft unification of predicates during the learning process with the effect of reducing the size of the computation tree needed to induce first-order clauses. Our *TNTP* framework is capable of learning a diverse set of induced rules that have improved predictive accuracy, whilst reducing the computational time by several orders of magnitude. We demonstrated this by evaluating our approach on three different datasets (UMLS, Kinship and Nations) and comparing our results with that of the *NTP* method, chosen here as our baseline.

1 Introduction

A symbolic knowledge base is a human-interpretable way to model real-world data. Despite often being incomplete or containing some incorrect data, the information can be used to induce human-interpretable first-order rules that can capture missing data and new relationships among the data. Typically, information in large knowledge bases is expressed as triples of the form <*subject, relation, object*>, for example <*aspirin, is-a-cure-for, headache*>. The noisy characteristic of these knowledge bases requires reasoning and learning methods capable of performing inference in a "soft" manner without assuming precise unification of constants and predicates. Symbolic rule learning, although capable of handling some form of noise in the data ([13, 20, 14]), assumes exact syntax and semantics of the symbols used in the representation of the knowledge. On the other hand, in sub-symbolic (differentiable) computations, the symbols used in the triples can be represented as high-dimensional vector embeddings, which are amenable to a notion of "soft-unification" defined in terms of some vector distance metric, e.g. Euclidean or Cosine distance between embeddings [18]. Such soft-unification has the potential to improve the robustness of first-order rule induction from large noisy knowledge bases.

Various approaches have been proposed to address the problem of learning accurate embedding representation of symbols in a given knowledge based by integrating neural network and symbolic reasoning (see [1] for a survey). In [25] the learning of such representations is targeted to the inference of new facts whereas in [26, 28] it is targeted to the learning of binary relations. More recently, integrated sub-symbolic and symbolic approaches have been proposed as differentiable end-to-end learning methods to perform rule induction over incomplete knowledge bases. Specifically, [8] represents knowledge bases symbolically and the differentiable aspect is for the search process of inductive rules over a given set of proposed templates, [4] represents predicates as embeddings and constants as symbols, and [22, 19] represent triples of the knowledge bases as embeddings. Among these approaches, *Neural Theorem Prover*, *NTP* [22], learns vector embedding representations for both predicates and constants. A separate set of predicates is used to express rule templates and the learned representation of these predicates yields new induced rules over a given knowledge base. The learning of predicate embeddings is controlled via a backward chaining reasoning mechanism over a given set of rule templates and makes use of a notion of "soft unification" between the representations of the predicates in these templates and that of the predicates in the given knowledge base, and variable substitution [22]. As stated in [22], *NTP* is however not very scalable: the computation tree generated by the backward chaining process grows with the size of the given knowledge base as it considers all possible predicates in the given knowledge base as potential candidates for soft unification with a current (unknown) predicate in the computation tree.

In this paper we address this limitation by extending the *NTP* approach with a topic-based method for controlling the induction of first-order clauses. Our proposed approach, called *TNTP* for *Topical NTP*, focuses on the problem of rule induction by learning embedding representations of a given knowledge base and set of rule templates in a (semantically) controlled manner. The approach makes use of the notion of *topics*. A topic clusters predicates that are semantically related to each other and given that the learned embedding representation of predicates captures the semantics of the predicate in high-dimensional space, a cluster is defined in terms of vector distance in high-dimensional space. The underlying idea is to compute topic clusters over predicate embedding representations and use these clusters to control the soft unification of predicates during the learning process with the effect of reducing the size of the computation tree needed to induce the rules. Our experiments show that our approach is capable of learning a diverse set of induced rules that have improved predictive accuracy, whilst reducing the computational time by several orders of magnitude. We demonstrated this by evaluating our approach on three different datasets widely used as benchmarks in the literature (UMLS, Kinship and Nations) and comparing our results against the three systems ComplEx [28], *NTP* [22] and *NTP2.0* [19].

The rest of the paper is structured as follows. Section 2 introduces reformulation of existing background terminology and concepts related to the *NTP* approach. In Section 3 and 4 we introduce our proposed topical *NTP*, its implementation and evaluation. Section 5 considers related work and Section 6 concludes with suggestions for future work.

2 Background

We introduce basic notions and terminologies needed to define the rule induction task addressed by our TNTP approach, together with a reformulation of the basic *NTP* architecture.

Terminology. We assume that a knowledge base $\mathcal{K}$ is expressed using a given *vocabulary* consisting of a set $\mathcal{P}$ of binary predicates and a set $\mathcal{C}$ of constants. (In this paper we assume all predicates are binary, although our system supports other arities.) A *rule* is of the form $h(\bar{V})$:- $b_1(\bar{V_1})\dots b_n(\bar{V_n})$, $n \geq 1$, where $h(\bar{V})$ and $b_i(\bar{V_i})$ are atoms. The atom $h(\bar{V})$ is called *head* or *head atom* of the rule, $b_1(\bar{V_1})\dots b_n(\bar{V_n})$ is called *body* of the rule and each $b_i(\bar{V_i})$ is called a *body atom*. $\bar{V}$ (respectively $\bar{V_i}$) represents the two (variable) arguments of the head (respectively ith body atom). All variables in a rule are universally quantified, each occurring in at least two atoms in the rule, and the variables in $h(\bar{V})$ are distinct. A *ground fact* is an atom with constant arguments. We will write $p(\bar{c})$ to denote a ground atom where $\bar{c}$ indicates the two constant arguments. A knowledge base $\mathcal{K}$ is a set of rules, including either just facts or facts and rules. An *assignment* of a constant c to a variable V is denoted as V/c, and a *substitution* is a set of assignments of constants to unique variables, denoted as $d = \{V_i/c_i\}$. A specific assignment to a variable V_i in a given substitution d, will be denoted as $d[V_i] = c_i$.

Given a vocabulary $\mathcal{P} \cup \mathcal{C}$, we consider a set $\#\mathcal{P}$ of *unknown* (or placeholder) predicates disjoint from $\mathcal{P}$. We refer to these unknown predicates as *induced* predicates. A *template* rule is a rule defined using only induced predicates (i.e. predicates from $\#\mathcal{P}$). For instance $\#p(X,Y)$:- $\#q_1(X,Z), \#q_2(Z,Y)$ is a template rule. The learning process considers a given set $\mathcal{I}$ of template rules and for each template rule $I \in \mathcal{I}$ it takes as input a number (m_I) of clone template rules, whose predicates are unique. Each predicate p, induced predicate $\#p$, and constant c, is represented by a k-dimension ($k = 100$) embedding θ_p, $\theta_{\#p}$, and θ_c, which is learned during training as part of the rule induction process. To simplify our notation, unless specified, we will denote the embedding representation of a symbol x as θ_x. A ground fact $p(c1, c2)$ is represented by the tuple of embeddings $\langle\theta_p, \theta_{c1}, \theta_{c2}\rangle$. Given a vocabulary $\mathcal{P} \cup \mathcal{C}$ and a set $\#\mathcal{P}$ of induced predicates, the *embedding matrix* of a learning task S is given by $\theta_S \in \mathbb{R}^{Z*k}$, where $Z = |\mathcal{P}| + |\#\mathcal{P}| + |\mathcal{C}|$.

Given a knowledge base $\mathcal{K}$ and a ground fact $q(\bar{c})$ in $\mathcal{K}$, a *corruption* of $q(\bar{c})$, denoted as $q(\hat{c})$, is a ground fact constructed from $q(\bar{c})$ by changing one or more of its constants so that $q(\hat{c}) \notin \mathcal{K}$. For instance, given a ground fact $p(c_1, c_2)$ in $\mathcal{K}$, corrupted facts can be generated by either changing the first constant $p(\hat{c_1}, c_2)$ or by changing the second constant, $p(c_1, \hat{c_2})$ or by changing both constants $p(\hat{c_1}, \hat{c_2})$ such that $\hat{c_i} \in \mathcal{C}$ and $p(\hat{c}) \notin \mathcal{K}$ [3].

Given a knowledge base $\mathcal{K}$, a training set τ is composed by (the embedding representation of the) facts in $\mathcal{K}$ and their respective corruptions. We can now informally define the learning task of the *NTP* (and our *TNTP*) approach. Starting from a knowledge base $\mathcal{K}$ and a set $\mathcal{I}$ of template rules, an embedding matrix is randomly initialised, containing embeddings for each predicate and constant in $\mathcal{K}$ and $\mathcal{I}$. The learning task computes an updated embedding matrix that captures associations between predicates and constants used in the knowledge base so that the system minimises the error in answering queries from the training set. The learned embeddings for the induced rules can then be used to answer unseen queries.

A Reformulation of NTP. We summarise here a reformulation of the key concepts of the original *NTP* framework [22] that are relevant to our *TNTP* approach (for more information see [22]). An example *NTP* computation tree is given in Figure 1 and elaborated below.

The *NTP* approach uses a backward chaining reasoning mechanism similar to Prolog's backward chaining [5, 10] to classify the truth value of a ground query (a labelled ground atom from the training set). Whereas *Prolog* uses *symbol-level unification*, the backward chaining reasoning of *NTP* uses *subsymbolic soft-unification*, as given by the function *uni_c* in Definition 1. This sub-symbolic soft unification employs the Radial Basis Function (RBF) formula $rbf(\theta_1, \theta_2) = exp(\frac{-||\theta_1-\theta_2||_2}{2\mu^2})$ for computing the similarity/distance between embedding vectors θ_1, θ_2.

Definition 1. *Given an embedding matrix θ and a substitution set d, the unification of i and j generates a tuple (d_u, s), where d_u is the updated substitution set, i, j can be predicates,*

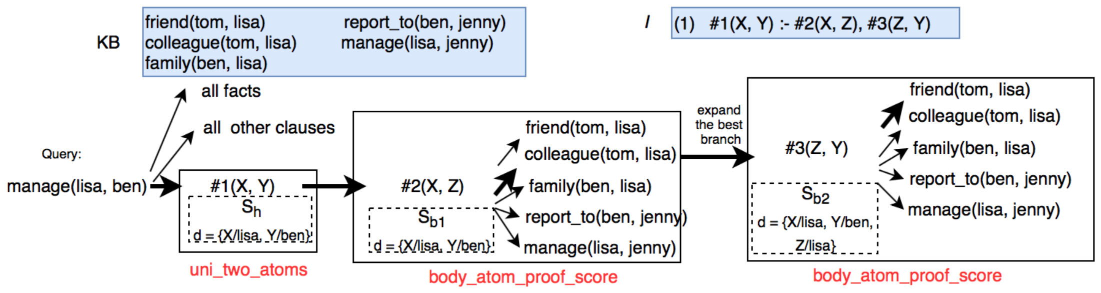

Figure 1: An example *NTP* computation tree using a knowledge base, part of which is shown in the top left corner. On receiving a query *NTP* finds the highest unification score between the query and each fact and rule in the knowledge base. The query is unified with facts using *uni_two_atoms* and with rules using *uni_two_atoms* and *body_atom_proof_score*. For illustration, we expand just one branch using a template rule. Firstly, the query unifies with the head of the given rule using *uni_two_atoms*. Then, the updated substitution set is passed for body atoms' proofs. The first body atom is proved using *body_atom_proof_score*, which unifies with all facts in the knowledge base pairwisely using *uni_two_atoms*. The best branch in *body_atom_proof_score* of $\#2(X, Z)$ would be the fact that outputs the highest proof score with $\#2(X, Z)$ using *uni_two_atoms*. Next, the k best branches ($k = 1$ here) are expanded to prove the second body atom using the same method. The proof score of the query using this rule is the minimal score among head and body unification.

constants or variables, and s is the unification score:

$$uni\_c(i, j, d) = \left\{ \begin{array}{ll} (d', 1), \textit{where } d' = d \cup \{i/j\}) & \textit{if is\_variable}(i) \textit{ and } i \notin d \\ (d,\ rbf(\theta_c, \theta_j)) & \textit{if is\_variable}(i) \textit{ and } d[i]{=}c \\ (d,\ rbf(\theta_i, \theta_j)) & \textit{otherwise} \end{array} \right\} \tag{1}$$

Unification of atoms is achieved by using *uni_c* together with the fact that atoms are encoded as tuples of embedding representations of the predicate symbol and its (constant) arguments. Specifically, the unification score of two atoms $a1$ and, $a2$ and an initial substitution d (which may be empty) is computed by the function $uni\_two\_atoms(a_1, a2, d)$, which returns the minimum score among each pairwise predicate-predicate and corresponding argument-argument unification scores (unified left to right) in which any updated substitutions are propagated through the argument unification to give a final updated substitution d_u. In Figure 1 the first application of *uni_two_atoms* is between the query *manage(lisa,ben)* and the atom *#1(X, Y)*, resulting in the substitution $\{X/lisa, Y/ben\}$ and the score between the predicate embeddings θ_{manage} and $\theta_{\#1}$.

The proof score of a query $q(\bar{c})$, denoted $ntp^{\kappa}_{\theta}(q, \bar{c})$, is computed as the maximum of the scores obtained by unifying $q(\bar{c})$ with each fact in $\mathcal{F}$ and with each template rule. The score of unifying $q(\bar{c})$ with a fact a is given by $uni\_two\_atoms(q(\bar{c}), a, \{\})$. To obtain the score of unifying $q(\bar{c})$ with a rule $r = h(\bar{V})\text{:-}b_1(\bar{V_1}), ..., b_n(\bar{V_n})$, a Prolog-like computation tree is constructed by first applying $uni\_two\_atoms(q(\bar{c}), h(\bar{V})\{\})$ to give a head-score and updated substitution d_u (see *uni_two_atoms* box in Figure 1). The score for the first body atom is then computed by $body\_atom\_proof\_score(b_1(\bar{V}), d_u)$ (see the first *body_atom_proof_score* box in Figure 1), which applies $uni\_two\_atoms(b_1(\bar{V}), a, d_u)$ to each fact $a \in \mathcal{F}$ generating a branch in the *NTP* tree for each fact and obtaining the score and updated substitution d'_u in each branch. The substitutions in the k_{max} branches with the best scores are propagated to the next body atom in the rule being matched. In Figure 1 we take $k_{max} = 1$ and the best scoring fact branch is with *colleague(tom,lisa)*. Similarly, $body\_atom\_proof\_score(b_i(\bar{X}), d'_u)$, $i = 2 \ldots n$ in sequence, is applied to following body atoms, where d'_u is the updated substitution from the previous body atom. The final score in each branch arising from matching with a rule is the minimum of the head-score and body atom scores in the branch. As just explained, unification of $q(\bar{c})$ with a fact introduces a branch in the computation tree, and unification with a template rule introduces several branches in the computation tree depending on the parameter k_{max}. However, of all these branches only the one with the maximum overall rule score receives gradients during backpropagation [22].

After constructing the computation tree, a number of iterations is performed over the tree to learn the embedding representations of predicates, constants and induced predicates, using the training dataset τ where ground facts from $\mathcal{K}$ have target score 1.0 and their corruptions have target score 0.0. At the beginning of the training process, the embedding matrix θ contains a randomly initialised embedding for each unique predicate and constant in the knowledge base and $\mathcal{I}$, and θ is the only training parameter. The training goal is to minimise the difference between $ntp^{\kappa}_{\theta}(q, \bar{c})$ and its target score y for each $q(\bar{c}) \in \tau$, by optimising θ. The loss function $L_{ntp^{\kappa}_{\theta}}$ is defined as the negative log-likelihood of ntp^{κ}_{θ}:

$$L_{ntp^{\kappa}_{\theta}} = \sum_{([q,\bar{c}],y)\in\tau} -y\, log(ntp^{\kappa}_{\theta}(q, \bar{c})) - (1 - y)\, log(1 - ntp^{\kappa}_{\theta}(q, \bar{c})) \tag{2}$$

To prevent a query $q(\bar{c})$ from unifying with itself in $\mathcal{F}$, resulting in no gradient updates, when $q(\bar{c})$ is queried during the training, $q(\bar{c})$ is temporarily masked in $\mathcal{F}$.

During training, a neural link predictor, *ComplEx*, is used to regularise *NTP* through a shared θ [28]. *ComplEx* is good at scoring binary atoms locally, whereas *NTP* can handle multi-hop reasonings. The joint training loss $L_{ntp\lambda^{\kappa}_{\theta}}$ is given by:

$$L_{ntp\lambda^{\kappa}_{\theta}} = L_{ntp^{\kappa}_{\theta}} + \sum_{(q(i,j),y)\in\tau} -y\, log(complex_{\theta}(q, i, j)) - (1 - y)\, log(1 - complex_{\theta}(q, i, j)) \tag{3}$$

where $q(i, j)$ is an atom in τ and y is its target score.

Training *NTP* with a full computation tree is computationally intensive and cannot be applied to any normal size knowledge base. Recall from $ntp^{\kappa}_{\theta}(q, \bar{c})$ that in each proof only one branch of the tree gets gradients in backpropagation, although many thousands of branches are computed in the forward pass. It is for this reason that *NTP* introduces a *K max* gradient approximation approach to reduce redundant computations – when a query unifies with a rule only the k_{max} branches of each body atom unification with the highest scores get propagated further and the other branches are discarded. For instance, assuming a knowledge base of 500 facts, a template rule $\#p(X)$:- $\#q(X), \#s(X)$ and $k_{max} = 10$, a query firstly unifies with the head atom $\#p(X)$ (1 branch). Then, it proves the first body atom $\#q(X)$ with all facts (1 * 500 branches). Since $k_{max} = 10$, only 10 out of 500 branches are expanded to prove $\#s(X)$, instead of expanding all 500 branches. This approximation reduces many redundant computations, but *NTP* still needs to unify each body atom with all possible facts in forward propagation to pick the k_{max} to expand, so it still suffers scalability issues. Our proposed approach addresses this problem by pruning the tree construction much further by means of *topics* as described in the next section.

3 Topical Neural Theorem Provers

We introduce now our approach, called *Topical NTP* (*TNTP*), which uses the notion of *topic* to prune the construction of the computational tree by focusing during each unification step only on predicates in the knowledge base that have "similar" embeddings, as opposed to the whole knowledge base.

Topic Generation. The prerequisite of topic generation in *TNTP* is a trained embedding matrix θ that captures semantic relationships in a knowledge base $\mathcal{K}$ consisting of facts $\mathcal{F}$. We describe the process of topic generation in Algorithm 1. To this end, embeddings are initialised randomly in *TNTP*. They are used to translate the symbolic knowledge base to a neural representation *neural_kb* (lines 2-3 of Algorithm 1). To get a proper θ (called $\theta_{pretrain}$), we pretrain *NTP* with a facts-only computation tree (i.e. without $\mathcal{I}$), denoted *FNTP* (line 4). *FNTP* is trained through gradient descent according to the loss function given by Equation 3. In this case, during *FNTP* training, each query q unifies with all facts except itself and outputs the highest prediction score, which is used to update θ_q. The purpose of this pretraining is to force embeddings of semantically similar predicates and constants to be closer in the vector space. For example, if the fact $treat(tom, diabetes)$ generates the highest proof score for the query $diagnose(tom, diabetes)$, it would be because $\theta_{diagnose}$ and θ_{treat} have become closer in vector space through gradient descent. The training makes a number of iterations (*PRETRAIN_ITER*) after which the trained predicate embeddings are extracted using the *predicate_ids* (lines 5-7). The embeddings are then used to obtain a set $\mathcal{G}$ of clusters, called *topics*, from the embedding θ_x of predicates x in $\mathcal{P}$ (we used the K-means clustering algorithm [15]). The size of $\mathcal{G}$, *G_CLUSTER*, can be predefined or selected based on the silhouette score [23]. It is known that high dimension embeddings inflate Euclidean distances involved in K-means training [11]; therefore, before clustering we use Principal component analysis (*PCA*) to transform θ_P to a d-dimensional θ_{d_P} (we used $d = 20$) [12]. The clusters $\{G_1, G_2, ..., G_g\}$ are then generated by applying K-means to θ_{d_P}. The topic t of each known predicate p is given by $topic(p) = i$ if $p \in G_i$ (lines 8-10).

```
def topic_generation(facts, train_data, predicate_ids):
    emb = initialise_embedding(facts)
    neural_kb = convert_knowledge_base_to_neural_form(facts, emb)
    fntp = construct_fact_only_computation_tree(neural_kb)
    for i in range(0, PRETRAIN_ITER):
        emb = train(fntp, emb, train_data)
    pred_emb = extract_trained_predicate_emb(emb, predicate_ids)
    pca_pred_emb = pca(pred_emb, D_DIMENSION)
    predicate_cluster_dic = compute_clusters_by_kmeans(pca_pred_emb, G_CLUSTER)
    return predicate_cluster_dic
```

Algorithm 1 A high-level algorithm of the *TNTP* pretraining phase. *NTP* without any template rules, called *FNTP*, is trained to get proper embeddings that reflect relationships of facts in $\mathcal{K}$. Topics are generated using these embeddings. Predefined constants are in capitals.

After generating topics, $\mathcal{F}$ is partitioned according to the topics of the atomic predicates, forming subsets $\mathcal{F}_i$, $i \in \{1, \ldots, g\}$, where $F_i = \{p(\bar{c}) \in \mathcal{F} : topic(p) = i\}$. It is this partition of $\mathcal{F}$ that is used during topical unification, which is described next and illustrated in Figure 2.

Topical Template Rules and Topic-based Unification. Recall from Section 2 that a template rule I typically has the form: (m_I) $\#p1(X,Y) : -\#p2(X,Y), \#p3(X,Y)$ where $\#p1$, $\#p2$ and $\#p3$ are placeholders for induced predicates and m_I indicates that m_I rules of this structure will be induced. We used $m_I = 20$ for each induced template rule in our experiments. Topical template rules are constructed after topic generation at the beginning of the *TNTP* induction phase. Each induced predicate $\#p$ is linked to a unique random embedding which is used to initialise the topic of $\#p$ by assigning it to a topic whose centroid is closest to the embedding of $\#p$ (lines 5-6 of Algorithm 2 listed at the end of this section), represented as

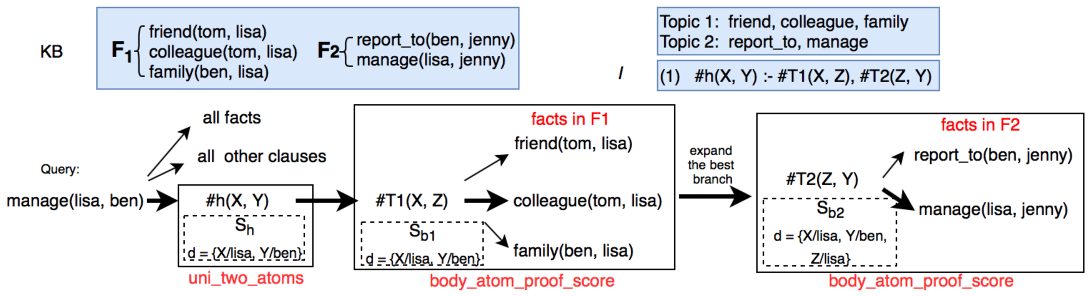

Figure 2: An example *topical computation tree* using a knowledge base, part of which is shown in the top left corner. Predicates are clustered into two topics (Topic 1 and Topic 2) and facts are partitioned according to their predicates (F1 and F2). In contrast to Figure 1, in each *body_atom_proof_score*, only facts of the given topics in the knowledge base are involved in unification.

$$topic(\#p) = \underset{i}{\operatorname{argmin}}\, distance(\#p, centroid(G_i)))$$

Note that the induced predicate of a head atom does not have a topic, because head atoms will be unified with given queries[1]. Assigning topics according to centroid distances is a random allocation with respect to cluster distributions in a vector space, which slightly favours topics that are more spread out. Maintaining a number of clones of each template structure allows for several topical induced rules with similar structural patterns to be induced.

Topic-based Unification. We have seen that in the *NTP* architecture when a body atom $q(\bar{c})$ is queried, all facts in $\mathcal{K}$ need to unify with $q(\bar{c})$. Instead, in *TNTP* the unification is restricted to the subset of facts in the topic of q, denoted as $\mathcal{F}_{topic(q)}$. The other top level mechanisms for proving a query are similar (see Figure 2). Topic-based unification forces unification to focus on a subset of the set $\mathcal{F}$ of facts in $\mathcal{K}$ that are most relevant, according to topic generation. This focus removes computation of irrelevant branches and makes each $\#p$ in a topic t converge to t faster. As a result, fewer training epochs are required. Since a topic acts as a filter in topic-based unification, the quality of the topics is important. However, if topics change significantly after some iterations of training, these topics can be updated to reflect the latest similarities of the learned predicates' representations. In our experiments, topic changes were observed to be minor, so we did not update topics.

Implementation Issues. In our experiments with *NTP* and *TNTP* we noticed that although the approach provides a framework for rule induction the rule induction is dominated by query-fact unification. Due to the computation tree construction, shorter rules (facts) have a stronger advantage than longer rules (rules in $\mathcal{I}$) and constants are unified more frequently than predicates, especially the induced predicates. When tracking the gradient flows, we found that most gradients go through facts and not template rules. Consequently, the induced predicates in template rules may not be trained well. To improve rule induction, we introduce two scaling factors: α and β, such that α scales induced predicate unification scores (see Definition 2) and β scales rule unification scores (see Definition 3). These scaling factors boost unification scores obtained via template rules and increase the chances for rules to receive gradients. With the

[1]For simplicity, if the topic of a body atom in an induced template rule is n, we will use $\#Tn$ to refer to the predicate of this atom.

boost of these scaling factors, embeddings of induced predicates get trained more frequently and better rules can be induced.

By way of exemplification, consider a query $q(a, b)$ to be proved using a template rule I: $\#h(X, Y)$:- $\#b1(X, Y), \#b2(X, Y)$, which after substitution becomes the instance $\#h(a, b)$:- $\#b1(a, b), \#b2(a, b)$. During each backpropagation, a and b could be updated three times, while h, $b1$ and $b2$ would each be updated once. In fact, induced predicates in $\#\mathcal{P}$ have even less chance to be updated than predicates in $\mathcal{P}$ as predicates in $\mathcal{P}$ may be updated also through the unifications with facts. The learned embedding of constants captures therefore better semantic information in the vector space giving a better unification score, whereas the learned embedding of predicates is more noisy and the unification score between two predicates is in general low. The scaling factors α is and β help to balance out the disparity between the training of predicates in $\mathcal{K}$ and the induced predicates and between predicates and constants.

Definition 2. *Given an embedding matrix θ, a substitution set d, an induced predicate scaling factor α, the unification score s between i and j and its updated substitution set d_u are output as a tuple (d_u, s):*

$$uni\_c(i, j, d) = \left\{ \begin{array}{ll} (d'\,1) \quad \textit{where } d' = d \cup \{i/j\} & \textit{if is\_variable(i) and } i \notin d \\ (d,\ rbf(\theta_c, \theta_j) & \textit{if is\_variable(i) and } d[i]{=}c \\ (d,\ \alpha \times rbf(\theta_i, \theta_j) & \textit{if is\_induced\_predicate(i)} \\ (d,\ rbf(\theta_i, \theta_j) & \textit{otherwise} \end{array} \right\} \tag{4}$$

Since the proof score of $q(\bar{c})$ using a rule IR is the minimal score among head and body atom scores, a fact tends to get a higher unification score than a rule. In order not to penalise rules, a rule scaling factor β is applied to amplify rules' unification scores by β.

Definition 3. *Given a knowledge base $\mathcal{K}$ with facts $\mathcal{F}$, template rules $\mathcal{I}$ and an embedding matrix θ, TNTP return a proof success score $tntp^{\mathcal{K}}_{\theta}(q, \bar{c})$ for each query $q(\bar{c})$ as follows:*

$$tntp^{\mathcal{K}}_{\theta}(q, \bar{c}) = \max(tanh \left\{ \begin{array}{ll} rule\_proof\_score(q(\bar{c}), r, \mathcal{F}) & foreach\ r \in \mathcal{F} \\ \beta * rule\_proof\_score(q(\bar{c}), r, \mathcal{F}) & foreach\ r \in \mathcal{I} \end{array} \right\}) \tag{5}$$

where rule_proof_score is the success score when unifying the query with a rule r (either a fact in $\mathcal{F}$ or a rule in $\mathcal{I}$) using TNTP. tanh scales TNTP predication scores between zero and one.

Algorithm 2 captures at a high-level the *TNTP* induction training mechanism.

```
def TNTP_induction(facts, train_data, template_rules, test_data, predicate_ids):
    predicate_cluster_dic = TOPIC_GENERATION(facts, train_data, predicate_ids)
    emb = initialise_embedding(facts)
    neural_kb = convert_knowledge_base_to_neural_form(facts, emb)
    predicate_cluster_dic = compute_induced_predicate_topic(
                                template_rules, predicate_cluster_dic)
    t_ntp = compute_topical_induction_computation_tree(
                neural_kb, template_rules, predicate_cluster_dic)
    for i in range(0, TRAIN_ITERATIONS):
        emb = train(t_ntp, emb, train_data)
    evaluate_ntp(t_ntp, test_data)
    induced_rules =
        decode_induced_rules(emb, template_rules, predicate_cluster_dic)
    return emb, induced_rules
```

Algorithm 2 Before computation tree construction, each predicate in $\mathcal{K}$ and $\mathcal{I}$ is linked to a topic, as defined by TOPIC_GENERATION (i.e. Algorithm 1). Next, a topical computation tree is constructed using a random-initialised embedding matrix *emb*. Topical unification is applied for each body atom. This *TNTP* is trained through gradient descent. Finally, trained *TNTP* is evaluated using test data and induced rules are generated by decoding the trained embeddings w.r.t. $\mathcal{K}$ (see Evaluation Metrics and Decoding Learned Rules in Section 4).

4 Experiments and Results.

We compare *TNTP* with the original *NTP* using three frequently used benchmark datasets: Alyawarra *Kinship*, *Nations* and Unified Medical Language System (*UMLS*) [7, 24, 17]. The characteristics of the datasets are listed in Table 1. We have evaluated the performance of *TNTP*

Dataset	Type	#constants	#predicates	#facts
Kinship [7]	genealogical	104	26	10686
Nations [24]	geographical	14	56	2565
UMLS [17]	biomedical	135	49	6529

Table 1: Characteristics of datasets used in experiments. Datasets are divided proportionately 8:1:1 into training, development and testing portions.

in several ways - quantitatively using the *MRR* and *HITS@m* metrics, and qualitatively by decoding the induced predicates in learned topical template rules and inspecting the (decoded) instances used in the rules and facts to score a query. We consider these in turn.

Evaluation Metrics. We have utilised similar evaluation metrics to those used in [22], that is the Mean Reciprocal Rank *MRR* and *HITS@m* (see [3]). For each test query q two sets of corrupted queries $\hat{q_1}$ (resp. $\hat{q_2}$) are constructed, corresponding to corrupting the first (resp. second) argument of q. The scores for the queries in $\hat{q_1}$ and $\hat{q_2}$ are obtained from the trained *TNTP* and the two sets of scores are ranked from highest score to lowest score to give $S_{\hat{q_1}}$ and $S_{\hat{q_2}}$. The indices in each list of the score of q ($index(S_{\hat{q_i}}, q)$, $i \in \{1,2\}$) are found[2], and the formulas in equations 6 are applied to calculate the *MRR* and *HITS@m* for each query q and for each $i \in \{1,2\}$.

$$MRR(q, S_{\hat{q_i}}) = \frac{1}{index(S_{\hat{q_i}}, q)} \qquad HITS@m(q, S_{\hat{q_i}}) = \left\{ \begin{array}{ll} 1 & \text{if } index(S_{\hat{q_i}}, q) \leq m \\ 0 & \text{otherwise} \end{array} \right\} \tag{6}$$

The *MRR* (resp. *HITS@m*) for each q is calculated as the average of $MRR(q, S_{\hat{q_1}})$ and $MRR(q, S_{\hat{q_2}})$ (resp. $HITS@m(q, S_{\hat{q_1}})$ and $HITS@m(q, S_{\hat{q_2}})$), and the *MRR* (resp. *HITS@m*) scores for the whole test set $\mathcal{T}$ are calculated by taking the average over all q. The intuition behind these metrics is the following. Corruptions of test queries are assumed to be false using the closed-world assumption [21], and hence would be expected to have low scores compared to a true test query. Therefore, if training has generalised well the test query score should be at or near the top of the ranking. However, since the knowledge base may be noisy and/or incomplete, such queries may actually be unknown positive facts and consequently be given high scores by the trained *TNTP*, possibly higher than the score of q, resulting in a lower rank in the *HITS* evaluation.

[2]The highest rank is used if there are duplicate proof scores.

Results		ComplEx[28]	NTP[22]	NTP2.0[19]	TNTP
Kinship	MRR	0.46	0.36	0.65	**0.90** ± **0.00**
	HITS@1	0.34	0.24	0.57	**0.87** ± **0.01**
	HITS@3	0.49	0.40	0.69	**0.92** ± **0.01**
	HITS@10	0.74	0.60	0.81	**0.95** ± **0.01**
Nations	MRR	0.60	0.62	**0.81**	0.80 ± 0.02
	HITS@1	0.46	0.45	0.73	**0.73** ± **0.02**
	HITS@3	0.67	0.72	0.83	**0.86** ± **0.01**
	HITS@10	0.97	0.99	0.99	**1.00** ± **0.00**
UMLS	MRR	0.58	0.60	0.76	**0.91** ± **0.01**
	HITS@1	0.47	0.51	0.68	**0.87** ± **0.01**
	HITS@3	0.63	0.64	0.81	**0.93** ± **0.01**
	HITS@10	0.80	0.81	0.88	**0.96** ± **0.01**

Table 2: Evaluation of a *TNTP* with 5 topics against *ComplEx*, *NTP* and *NTP2.0* using *Kinship*, *Nations* and *UMLS* datasets.

Results. The results are shown in Table 2. Our experiments demonstrate that *TNTP* significantly outperforms *ComplEx*, original *NTP* and a later *NTP* extension *NTP2.0*[3] on the *MRR* and *HITS@m* metrics. All experiments were run using the same set of template rules namely #1(X, Y) :- #2(X, Y); #1(X, Y) :- #2(Y, X); and #1(X, Y) :- #2(X, Z), #3(Z, Y), with 20 clones of each. *TNTP* mostly uses the same training parameters and template rules as [22]; that is 100-dimension embeddings, rbf parameter $\mu = \frac{1}{\sqrt{2}}$, ADAM gradient descent with 0.001 learning rate and 0.01 l_2 regularisation. However, *TNTP* training used 5 topics, $k_{max} = 1$ and 30 epochs instead of $k_{max} = 10$ and 100 epochs as used in [22]. The scaling factors (α, β) used in *Kinship*, *Nations* and *UMLS* are (20, 10), (10, 1) and (50, 10) respectively. Furthermore, the computational efficiency of *TNTP* is better than that of *NTP* by several orders of magnitude, more so when the knowledge base $\mathcal{K}$ has many predicates. In particular, in our experiments *TNTP* is approximately 77.5, 54.1 and 44.6 times faster than *NTP* for *Kinship*, *Nations* and *UMLS* datasets. We ran these timing experiments for both *NTP* and *TNTP* on the same processors and used the same parameters as used for the accuracy results in Table 2.

Decoding Learned Rules. After training, the embeddings in θ have been updated to reflect the semantic relationships among $\mathcal{P}$, $\#\mathcal{P}$ and $\mathcal{C}$ and the template rules and facts can be *decoded* into human interpretable rules using the vocabulary of $\mathcal{K}$. To do this the topical template rules in $\mathcal{I}$ are first ranked according to the frequency that a rule is used by positive (i.e. uncorrupted) queries in the last iteration of training. We set a frequency threshold for the frequency of a rule to be reached for it to be decoded. For each topical template rule, the N (here N=3) predicates in $\mathcal{P}$ that are closest to the induced head predicate $\#h$ (that is for these p the $\textit{rbf}(\#h, p)$ scores are the N highest) , and similarly for each induced body predicate $\#b$ the N closest predicates in $\textit{topic}(\#b)$ are found. Each such set of known predicates from $\mathcal{P}$ is called the *decoding tuple* for the induced predicate. In fact, any p_i in the decoding tuple of an induced predicate $\#p$ such that $\textit{rbf}(\#p, p_i)$, $i \in \{2, \ldots, N\}$ is lower than a preset threshold is removed. Finally, for each template rule the decoded rules are generated by combining possible predicates in the decoding tuple for each induced predicate. The score of each of the rules thus generated is the minimum of the computed *rbf* scores of predicates in the rule (See Table 3.)

In testing mode, *TNTP* tracks the rule and facts used to prove each query. We investigated this and found that for each test query, *TNTP* is able to select an induced rule which has a head atom that is closely related to the query, and body atoms that could be supported by facts. For example, the test query *interacts_with(antibiotic, biologically_active_substance)* is proved by the

[3]We did not evaluate on the WordNet18 dataset, because in [27] it was reported to suffer from severe test set leakage; also a warning was given at https://github.com/villmow/datasets_knowledge_embedding .

Topical Template Rule	Top 3 Induced Predicate Decoding	Decoded rule
#H(X, Y) :- #T2(X, Z), #T2(Z, Y).	#H [0.81, 0.21, 0.17] [affects, evaluation_of, indicates] #T2 [0.70, 0.37, 0.27] [affects, precedes, process_of] #T2 [0.82, 0.82, 0.23] [interacts_with, precedes, degree_of]	0.70: affects(X, Y) :- affects(X, Z), interacts_with(Z, Y). 0.70: affects(X, Y) :- affects(X, Z), precedes(Z, Y).
#H(X,Y) :- #T1(X,Y).	#H [0.87, 0.85, 0.20] [location_of, carries_out, adjacent_to] #T1 [0.84, 0.81, 0.28] [carries_out, location_of, interconnects]	0.84: location_of(X, Y) :- carries_out(X, Y). 0.84: carries_out(X, Y) :- carries_out(X, Y). 0.81: location_of(X, Y) :- location_of(X, Y). 0.81: carries_out(X, Y) :- location_of(X, Y).
#H(X,Y) :- #T0(X, Z), #T2(Z, Y).	#H [0.79, 0.51, 0.18] [causes, property_of, part_of] #T0 [0.79, 0.60, 0.21] [causes, property_of, prevents] #T2 [0.78, 0.74, 0.55] [complicates, co-occurs_with, interacts_with]	0.78: causes(X, Y) :- causes(X, Z), complicates(Z, Y). 0.74: causes(X, Y) :- causes(X, Z), co-occurs_with(Z, Y). 0.60: causes(X, Y) :- property_of(X, Z), complicates(Z, Y). 0.60: causes(X, Y) :- property_of(X, Z), co-occurs_with(Z, Y).

Table 3: This table illustrates decoding of template rules. Firstly, a template rule is trained in *TNTP* to update embeddings. If the rule is involved in training frequently enough, it would be decoded. Each induced predicate in the rule can be decoded to one of its top 3 nearest neighbours in the given topic (middle column). Various rules can be formed using different combinations of the top 3 predicates. The final decoded rules (last column) are the rules with rule scores above a decoding threshold (0.5 used in the table). For example, in the first row the predicates *evaluation_of* and *indicates* would always yield a decoded rule below the threshold so are not generated.

induced rule *interacts_with(X, Y) :- interacts_with(Y, X)* where the body atom is supported by a semantically similar fact *interacts_with(biologically_active_substance, receptor)*. Similarly, the test query *causes(vitamin, injury_or_poisoning)* is proved by using the induced rule *causes(X, Y) :- causes(X, Z), complicates(Z, Y)* and its two body atoms are, respectively, supported by *causes(neuroreactive_substance_or_biogenic_amine, mental_or_behavioral_dysfunction)* and *complicates(experimental_model_of_disease, injury_or_poisoning)*. This demonstrates that *TNTP* can use semantically similar predicates and constants interchangeably in proofs through soft unification, for example *mental_or_behavioral_dysfunction* and *experimental_model_of_disease* unify.

Discussion. The introduction of topical template rules reduces the chance of inducing duplicate rules and enables *TNTP* to generate a diverse set of induced rules covering many of the relationships that exist in a knowledge base. We favour using rules over facts because rules generalise better with unseen queries. This is achieved by means of a scaling factor in the scoring function that actively rewards answering a query, which leads to our better results. We observe this through the use of an index *INDUC* that measures the percentage of positive test queries that use induced rules to get prediction scores. In Table 2, the *INDUC* of *TNTP* results in *Kinship*, *Nations* and *UMLS* datasets are, respectively, 77.4%, 98.6%, 90.0%. A high *INDUC*, together with high *MRR* and *HITS* scores, indicate that induced rules are trustworthy and cover a majority relationships in the knowledge base. As mentioned above our decoding mechanism provides a human-readable representation to interpret induced rules. We checked that a majority of the highly-ranked induced rules are consistent with the knowledge base.

We evaluated the impact of increasing the numbers of template clones on proof accuracy using the *UMLS* dataset. The values of the *MRR* and *HITS@1* scores appear to be relatively stable, even for low numbers of clones, because a query can unify with similar facts if there are no suitable rules. However, the *INDUC* score is higher if there are more clones of template rules. This indicates *TNTP* learns rules whenever possible, and, as a result, induced rules and embeddings learned during training possess better generality when tested with unseen queries. We also noticed that the computational efficiency improvement is greater for large knowledge bases when using more topics, which is expected since the average size of each topic would be smaller, thus cutting down the number of potentially "matching" facts to consider.

5 Related Work

We mention here recent research on neural-symbolic integration that uses differentiable approaches to enable neural networks to learn or use symbolic rules. A Logic Tensor Network (LTN) [25] represents a knowledge base using embeddings, but in a different way to [22]. Each unique predicate and constant is linked to an embedding (called 'grounding' in the paper), which further composes atoms and rules using *Real Logic*, forming subsymbolic representations. It converts symbolic background knowledge to these representations and applies gradient descent to find the optimal values for these groundings. This work illustrates that logic can be applied in deep neural networks for knowledge completion and query predictions. However, there are no rule inductions in LTN. Other works that learn relations without rule inductions are [6, 26, 28].

Unlike the backward-chaining inference used in our work, [8] proposes a differentiable model of forward chaining inference for program induction. They also represent relations by functions, but they do not use embeddings. They define each predicate p by a rule weight matrix W_p which measures how likely a pair of rules defines a predicate correctly, but this approach is unscalable. The framework *AMIE+* [9] tackles the same problem of rule induction as *TNTP*, but uses a purely symbolic approach based on iterative rule refinement. Instead of using a closed world assumption to generate negative examples, *AMIE+* follows an open world assumption, more specifically a partial completeness assumption (PCA). Under PCA, a new atom $p(a, b)$ can be assumed false only if there is already an existing fact $p(a, c)$ where $c \neq b$.

The work most relevant to ours is [19], where they present a scalable improvement to *NTP*, *NTP2.0*. *NTP2.0* reduces the number of facts used in atom-level unification. Instead of using pretrained topics, they divide the knowledge base using the Hierarchical Navigable Small World algorithm [16] (an Approximate Nearest Neighbour Search algorithm) and recompute subsets of the knowledge base at intervals during training. However, this method is vulnerable to noise, because it is risky to use a small subset of facts when all embeddings are initialised randomly. Our experiments show that *TNTP* significantly outperforms *NTP* in the *MRR* and *HITS@m* metrics, whereas their work achieves similar performance as *NTP* under the same setting. In addition, our approach can split an induction task safely at the beginning, which enables us to perform distributive induction over large knowledge bases, whereas their induction has to be done monolithically.

6 Conclusion and Future Work

We have presented an interpretable, end-to-end differentiable, and scalable first-order inductive framework *TNTP*, that is able to induce rules in knowledge bases with good accuracy. The use of topic-based unification in *TNTP* improves the scalability with respect to *NTP* by enabling the construction of the proof tree to focus only on the most relevant atoms, according to semantic similarity of atoms in terms of their closeness in high dimensional space. Scaling factors in *TNTP* enable more gradients to go through induced rules, instead of known facts. Topics and scaling factors are orthogonal factors that together contribute to enable *TNTP* achieve state-of-the-art accuracy on the datasets *Kinship*, *Nations* and *UMLS* while using a fraction of computational time of *NTP*.

The use of topical template rules allows a big induction task to be split into smaller independent induction tasks that focus on different aspects of a knowledge base and which can be run in parallel, merging their induced rules later. This parallelisation empowers *TNTP* to induce rules on real world datasets and in the next step, we are going to fully evaluate parallel induction. We are currently testing a *TNTP* parallelism induction framework to enable full scalable support for such real-world datasets such as a subset of Freebase [2] and our preliminary experiments demonstrate that in each independent task *TNTP* can effectively induce rules given a subset of template rules, even if these templates cannot cover all relationships that exist in the knowledge base.

Finally, the ability to decode template rules using the vocabulary of the knowledge base allows *TNTP* to form human-readable rules, which are ranked according to the frequency that they are used by positive queries in the last iteration of training. A frequently used induced rule can be confidently included as background knowledge. Therefore, in the next step, we plan to conduct a full evaluation on knowledge bases that include general rules as well as facts in order to make use of commonsense or prior domain specific knowledge. Selecting a set of highly ranked induced rules as background knowledge should improve *TNTP* accuracy and make the knowledge base more complete. We are also going to investigate the use of sub-domains to cluster constants from a similar domain (analogous to topics for predicates) in order to improve computational efficiency further. Finally, we plan to extend the language of the template rules allowing predicates of arity one, two or three and negated body conditions in the rules.

References

[1] Tarek R. Besold, Artur S. d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro M. Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luís C. Lamb, Daniel Lowd, Priscila Machado Vieira Lima, Leo de Penning, Gadi Pinkas, Hoifung Poon, and Gerson Zaverucha. Neural-symbolic learning and reasoning: A survey and interpretation. *arXiv preprint: 1711.03902*, 2017.

[2] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 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, pages 1247–1250, New York, NY, USA, 2008. ACM.

[3] 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'11, pages 301–306. AAAI Press, 2011.

[4] Andres Campero, Aldo Pareja, Tim Klinger, Josh Tenenbaum, and Sebastian Riedel. Logical rule induction and theory learning using neural theorem proving. *arXiv preprint: 1809.02193*, 2018.

[5] W. F. Clocksin and C. S. Mellish. *Programming in Prolog.* Springer-Verlag, Berlin, Heidelberg, 1987.

[6] William W. Cohen, Fan Yang, and Kathryn Mazaitis. Tensorlog: Deep learning meets probabilistic dbs. *arXiv preprint: 1707.05390*, 2017.

[7] Woodrow W. Denham and Douglas R. White. Multiple measures of alyawarra kinship. *Field Methods*, 17(1):70–101, 2005.

[8] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *J. Artif. Int. Res.*, 61(1):1–64, January 2018.

[9] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Fast Rule Mining in Ontological Knowledge Bases with AMIE+. *The VLDB Journal*, 2015.

[10] Lisa Anne Hendricks, Subhashini Venugopalan, Marcus Rohrbach, Raymond J. Mooney, Kate Saenko, and Trevor Darrell. Deep compositional captioning: Describing novel object categories without paired training data. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 1–10, 2016.

[11] Alexander Hinneburg and Daniel A. Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. pages 506–517. Morgan Kaufmann, 1999.

[12] I.T. Jolliffe. *Principal Component Analysis.* Springer Verlag, 1986.

[13] Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs. In *Proceedings of the 14th European Conference on Logics in Artificial Intelligence - Volume 8761*, pages 311–325, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[14] Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs from noisy examples. *Advances in Cognitive Systems*, 6:1–20, 2018.

[15] J. Macqueen. Some methods for classification and analysis of multivariate observations. In *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[16] Yu Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 01 2013.

[17] Alexa McCray. An upper-level ontology for the biomedical domain. *Comparative and functional genomics*, 4:80–4, 01 2003.

[18] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, USA, 2013. Curran Associates Inc.

[19] Pasquale Minervini, Matko Bosnjak, Tim Rocktäschel, and Sebastian Riedel. Towards neural theorem proving at scale. *arXiv preprint:1807.08204*, 2018.

[20] 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, Jul 2015.

[21] Raymond Reiter. On closed world data bases. In *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, France, 1977*, pages 55–76, 1977.

[22] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3788–3800. Curran Associates, Inc., 2017.

[23] Peter Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.*, 20(1):53–65, November 1987.

[24] Rudolph J. Rummel. Dimensionality of nations project: Attributes of nations and behavior of nation dyads, 1950-1965. *MI: Inter-university Consortium for Political and Social Research*, pages 80–4, 1992.

[25] Luciano Serafini and Artur S. d'Avila Garcez. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. *arXiv preprint: 1606.04422*, 2016.

[26] Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Y. Ng. Reasoning with neural tensor networks for knowledge base completion. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, pages 926–934, USA, 2013. Curran Associates Inc.

[27] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66, Beijing, China, July 2015. Association for Computational Linguistics.

[28] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 2071–2080. JMLR.org, 2016.