Go for a Walk and Arrive at the Answer: Reasoning Over Paths in Knowledge Bases with Reinforcement Learning

Rajarshi Das[*,1], **Shehzaad Dhuliawala**[*,1], **Manzil Zaheer**[2,4], **Luke Vilnis**[1], **Ishan Durugkar**[3], **Akshay Krishnamurthy**[1], **Alex Smola**[4], **Andrew McCallum**[1]
[1]University of Massachusetts, Amherst, [2]Carnegie Mellon University
[3]University of Texas at Austin, [4]Amazon Web Services
`{rajarshi, sdhuliawala, luke, akshay, mccallum}@cs.umass.edu`
`manzil@cmu.edu, ishand@cs.utexas.edu, alex@smola.org`

Abstract

Knowledge bases (KB), both automatically and manually constructed, are often incomplete — many valid facts can be inferred from the KB by synthesizing existing information. A popular approach to KB completion is to infer new relations by combinatory reasoning over the information found along other paths connecting a pair of entities. Given the enormous size of KBs and the exponential number of paths, previous path-based models have considered only the problem of predicting a missing relation given two entities, or evaluating the truth of a proposed triple. Additionally, these methods have traditionally used random paths between fixed entity pairs or more recently learned to pick paths between them. We propose a new algorithm, MINERVA[1], which addresses the much more difficult and practical task of answering questions where the relation is known, but only one entity. Since random walks are impractical in a setting with combinatorially many destinations from a start node, we present a neural reinforcement learning approach which learns how to navigate the graph conditioned on the input query to find predictive paths. Empirically, this approach obtains state-of-the-art results on several datasets, significantly outperforming prior methods.

1 Introduction

Automated reasoning, the ability of computing systems to make new inferences from observed evidence, has been a long standing goal of artificial intelligence. We are interested in automated reasoning on large knowledge bases (KB) with rich and diverse semantics [38, 1, 4]. KBs are highly incomplete [22], and facts not directly stored in a KB can often be inferred from those that are, creating exciting opportunities and challenges for automated reasoning. For example, consider the small knowledge graph in figure 1. We can infer the (unobserved fact) home stadium of Colin Kaepernick from the following reasoning *path*: Colin Kaepernick → PlaysInTeam → 49ers → TeamHomeStadium → Levi's Stadium. Our goal is to automatically learn such reasoning paths in KBs. We frame the learning problem as one of query answering, that is to say, answering questions of the form (Colin Kaepernick, PlaysInLeague, ?).

From its early days, the focus of automated reasoning approaches has been to build systems which can learn crisp symbolic logical rules [21, 29]. Symbolic representations have also been integrated with machine learning especially in statistical relational learning [24, 13, 19, 20], but due to poor generalization performance, these approaches have largely been superceded by distributed vector representations. Learning embedding of entities and relations using tensor factorization or neural methods has been a popular approach (Nickel et al., 2011; Bordes et al., 2013; Socher et al., 2013; inter alia), but these methods cannot capture chains of reasoning expressed by KB paths. Neural multi-hop models [25, 15, 41] address the aforementioned problems to some extent by operating on KB paths in vector space. However, these models take as input a set of paths which are gathered by

[1]`https://github.com/shehzaadzd/MINERVA`

*The first two authors contributed equally.

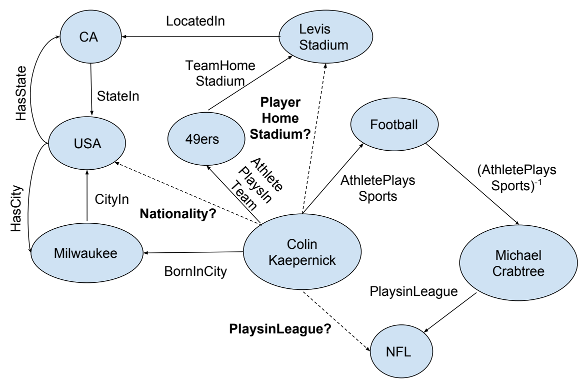

Figure 1: A small fragment of a knowledge base represented as a knowledge graph. Solid edges are observed and dashed edges are part of queries. Note how each query (e.g. Nationality, PlaysInLeague, PLayerHomeStadium) can be answered by traversing the graph via "logical" paths between entity 'Colin Kaepernick' and the corresponding answer.

performing random walks *independent* of the query relation. Additionally, models such as [25, 8] use the same set of initially collected paths to answer a diverse set of query types (e.g. MarriedTo, Nationality, WorksIn etc.).

This paper presents a method for efficiently searching the graph for answer-providing paths using reinforcement learning (RL) conditioned on the input question, eliminating any need for precomputed paths. Given a massive knowledge graph, we learn a policy, which, given the query $(entity_1, relation, ?)$, starts from $entity_1$ and learns to walk to the answer node by choosing to take a labeled relation edge at each step, *conditioning* on the query relation and entire path history. This formulates the query-answering task as a reinforcement learning (RL) problem where the goal is to take an optimal sequence of decisions (choices of relation edges) to maximize the expected reward (reaching the correct answer node). We call the RL agent MINERVA for "Meandering In Networks of Entities to Reach Verisimilar Answers."

Our RL-based formulation has many desirable properties. First, MINERVA has the built-in flexibility to take paths of variable length, which is important for answering harder questions that require complex chains of reasoning [36]. Secondly, MINERVA needs *no pretraining* and trains on the knowledge graph from scratch with reinforcement learning; no other supervision or fine-tuning is required representing a significant advance over prior applications of RL in NLP. Third, our path-based approach is computationally efficient, since by searching in a small neighborhood around the query entity it avoids ranking all entities in the KB as in prior work. Finally, the reasoning paths found by our agent automatically form an interpretable provenance for its predictions.

The main contributions of the paper are: (a) We present agent MINERVA, which learns to do query answering by walking on a knowledge graph conditioned on an input query, stopping when it reaches the answer node. The agent is trained using reinforcement learning, specifically policy gradients (§ 2). (b) We evaluate MINERVA on several benchmark datasets and compare favorably to Neural Theorem Provers (NTP) [33] and Neural LP [48], which do logical rule learning in KBs. (c) We also compare to DeepPath [46] which uses reinforcement learning to pick paths between entity pairs. The main difference is that the state of their RL agent includes the answer entity since it is designed for the simpler task of predicting if a fact is true or not. As such their method cannot be applied directly to our more challenging query answering task where the second entity is unknown and must be inferred. Nevertheless, MINERVA outperforms DeepPath on their benchmark NELL-995 dataset when compared in their experimental settings.

2 Task and Model

We formally define the task of query answering in a KB. Let $\mathcal{E}$ denote the set of entities and $\mathcal{R}$ be the set of binary relations. Then a KB is a collection of facts stored as triplets $(\mathrm{e}_1, \mathrm{r}, \mathrm{e}_2)$ where $\mathrm{e}_1, \mathrm{e}_2 \in \mathcal{E}$ and $\mathrm{r} \in \mathcal{R}$. Query answering seeks to answer questions of the form $(\mathrm{e}_1, \mathrm{r}, ?)$, e.g. Toronto, locatedIn, ?. We would also like to clearly point out the difference between query answering and the task of fact prediction. Fact prediction involves predicting if a fact is true or not, e.g. (Toronto, locatedIn, Canada)?. This task is easier than predicting the correct entity as the answer in query answering since the latter require finding the answer entity among many possible entities.

Next we describe how we reduce the problem of query answering in a KB to a finite horizon sequential decision making problem and solve it using reinforcement learning. We begin by representing the environment as a deterministic Markov decision process on a knowledge graph $\mathcal{G}$ derived from the KB (§2.1). Our RL agent is given an input query of the form $\left(\mathrm{e_{1q}}, \mathrm{r_q}, ?\right)$. Starting from vertex corresponding to e_{1q} in the knowledge graph $\mathcal{G}$, the agent learns to traverse the environment/graph to mine the answer and stop when it determines the answer (§ 2.2). The agent is trained using policy gradient more specifically by REINFORCE [45] with control variates (§ 2.3). Let us begin by describing the environment.

2.1 Environment - States, Actions, Transitions and Rewards

Our environment is a finite horizon deterministic and partially observed Markov decision process that lies on a knowledge graph derived from the KB. Recall that a KB is collection of facts stored as triplets $(\mathrm{e_1}, \mathrm{r}, \mathrm{e_2})$ where $\mathrm{e_1}, \mathrm{e_2} \in \mathcal{E}$ and $r \in \mathcal{R}$. From the KB, a knowledge graph $\mathcal{G}$ can be constructed where the entities s, t are represented as the nodes and relation r as labeled edge between them. Also, following previous approaches [2, 25, 46], we add the inverse relation of every edge, i.e. for an edge $(\mathrm{e_1}, \mathrm{r}, \mathrm{e_2}) \in E$, we add the edge $(\mathrm{e_2}, \mathrm{r}^{-1}, \mathrm{e_1})$ to the graph. On this graph we will now specify a deterministic partially observable Markov decision process, which we elaborate below.

States. The state space consists of all possible query-answers cartesian product with the set of entities. Intuitively, we want a state to encode the query $(\mathrm{e_{1q}}, \mathrm{r_q})$, the answer $(\mathrm{e_{2q}})$, and a location of exploration $\mathrm{e_t}$ (current node of the entity). Thus overall a state is represented by $(\mathrm{e_t}, \mathrm{e_{1q}}, \mathrm{r_q}, \mathrm{e_{2q}})$ and the state space consists of all valid combinations.

Observations. The complete state of the environment is not observable, but only its current location of exploration and query can be observed but not the answer, i.e. only $(\mathrm{e_t}, \mathrm{e_{1q}}, \mathrm{r_q})$ is observed.

Actions. The set of possible actions $\mathcal{A}_S$ from a state $S = (\mathrm{e_t}, \mathrm{e_{1q}}, \mathrm{r_q}, \mathrm{e_{2q}})$ consists of all outgoing edges of the vertex $\mathrm{e_t}$ in $\mathcal{G}$ and NO-OP. Basically, this means an agent at each state has option to select which outgoing edge it wishes to take having the knowledge of the label of the edge r and destination vertex t or just take no action and remain at the current vertex.

Transition. The environment evolves deterministically by just updating the state to the new vertex pointed by the edge selected by the agent through its action. The query and answer remains the same.

Rewards. We only have a terminal reward of +1 if the current location is the correct answer at the end and 0 otherwise. To elaborate, if $S_T = (\mathrm{e_t}, \mathrm{e_{1q}}, \mathrm{r_q}, \mathrm{e_{2q}})$ is the final state, then we receive a reward of +1 if $\mathrm{e_t} = \mathrm{e_{2q}}$ else 0.

2.2 Policy Network

To solve the finite horizon deterministic partially observable Markov decision process described above, we aim to design a randomized history-dependent policy $\pi = (\mathbf{d_1}, \mathbf{d_2}, ..., \mathbf{d_{T-1}})$, where $\mathbf{d_t} : H_t \to \mathcal{P}(\mathcal{A}_{S_t})$ and history $H_t = (H_{t-1}, A_{t-1}, O_t)$ is just the sequence of observations and actions taken. We restrict ourselves to the function class expressed by long short-term memory network (LSTM) [17] for learning the randomized history-dependent policy.

An agent based on LSTM encodes the history H_t as a continuous vector $\mathbf{h_t} \in \mathbb{R}^{2d}$. We also have embedding matrix $\mathbf{r} \in \mathbb{R}^{|\mathcal{R}| \times d}$ and $\mathbf{e} \in \mathbb{R}^{|\mathcal{E}| \times d}$ for the binary relations and entities respectively. The history embedding for $H_t = (H_{t-1}, A_{t-1}, O_t)$ is updated according to LSTM dynamics:

$$\mathbf{h_t} = \mathrm{LSTM}\left(\mathbf{h_{t-1}}, [\mathbf{a_{t-1}}; \mathbf{o_t}]\right) \tag{1}$$

where $\mathbf{a_{t-1}} \in \mathbb{R}^d$ and $\mathbf{o_t} \in \mathbb{R}^d$ denote the vector representation for action/relation at time $t-1$ and observation/entity at time t respectively and $[;]$ denote vector concatenation. To elucidate, $\mathbf{a_{t-1}} = \mathbf{r}_{A_{t-1}}$, i.e. the embedding of the relation corresponding to label of the edge the agent chose at time $t-1$ and $\mathbf{o_t} = \mathbf{e}_{\mathrm{e}_t}$ if $O_t = (\mathrm{e_t}, \mathrm{e_{1q}}, \mathrm{r_q})$ i.e. the embedding of the entity corresponding to vertex the agent is at time t.

Based on the history embedding $\mathbf{h_t}$, the policy network makes the decision to choose an action from all available actions $(\mathcal{A}_{S_t})$ *conditioned* on the query relation. Recall that each possible action represents an outgoing edge with information of the edge relation label l and destination vertex/entity d. So embedding for each $A \in \mathcal{A}_{S_t}$ is $[\mathbf{r_l}; \mathbf{e_d}]$, and stacking embeddings for all the outgoing edges we obtain the matrix $\mathbf{A_t}$. The network taking these as inputs is parameterized as a two-layer feed-forward network with ReLU nonlinearity which takes in the current history representation $\mathbf{h_t}$ and the embedding for the query relation $\mathbf{r_q}$ and outputs a probability distribution over the possible actions

Task	Metric	Model			
		ComplEx	**NTP**	**NTP-λ**	**MINERVA**
S1		99.37±0.4	90.83±15.4	**100.0±0.0**	**100.0±0.0**
S2	AUC-PR	87.95±2.8	87.4±11.7	**93.04±0.4**	91±0.01
S3		48.44±6.3	56.68±17.6	77.26±17.0	**93±0.01**

Table 1: Performance on COUNTRIES dataset. MINERVA significantly outperforms baselines in the challenging S3 task.

from which a discrete action is sampled. In other words,

$$\mathbf{d_t} = \text{softmax}\left(\mathbf{A_t}(\mathbf{W_2}\text{ReLU}\left(\mathbf{W_1}\left[\mathbf{h_t}; \mathbf{o_t}; \mathbf{r_q}\right]\right))\right)$$
$$A_t \sim \text{Categorical}\left(\mathbf{d_t}\right)$$

Note that the nodes in $\mathcal{G}$ do not have a fixed ordering or number of edges coming out from them. The size of matrix $\mathbf{A_t}$ is $|\mathcal{A}_{S_t}| \times 2d$, so the decision probabilities d_t lies on simplex of size $|\mathcal{A}_{S_t}|$. Also the procedure above is invariant to order in which edges are presented as desired and falls in purview of neural networks designed to be permutation invariant [49]. Finally, to summarise, the parameters of the LSTM, the weights $\mathbf{W_1}$, $\mathbf{W_2}$, the corresponding biases (not shown above for brevity), and the embedding matrices form the parameters θ of the policy network.

2.3 Training

For the policy network (π_θ) described above, we want to find parameters θ that maximizes the expected reward:

$$J(\theta) = \mathbb{E}_{(e_1,r,e_2)\sim D}\mathbb{E}_{A_1,..,A_{T-1}\sim\pi_\theta}[R(S_T)|S_1 = (e_1, e_1, r, e_2)]$$

where we assume there is a true underlying distribution $(\mathrm{e}_1, \mathrm{r}, \mathrm{e}_2) \sim D$. To solve this optimization problem, we employ REINFORCE [45] as follows:

- The first expectation is replaced with empirical average over the training dataset.
- For the second expectation, we approximate by running multiple rollouts for each training example. The number of rollouts is fixed and for all our experiments we set this number to 20.
- For variance reduction, a common strategy is to use an additive control variate baseline [16, 12, 11]. We use a moving average of the cumulative discounted reward as the baseline. We tune the weight of this moving average as a hyperparameter. Note that in our experiments we found that learnt baseline performed similarly, but we finally settled for cumulative discounted reward as the baseline owing to its simplicity.
- To encourage the policy to sample more diverse paths rather than sticking with a few, we add an entropy regularization term to our cost function after multiplying it by a constant (β). We treat β as a hyperparameter to control the exploration exploitation trade-off.

Experimental Details We choose the relation and embedding dimension size as 200. The action embedding is formed by concatenating the entity and relation embedding. We use a 3 layer LSTM with dimension size of 400. The hidden layer size of MLP (weights $\mathbf{W_1}$ and $\mathbf{W_2}$) is set to 400. We use Adam [18] with the default parameters in REINFORCE for the update.

3 Experiments

We test our model on the benchmark datasets of NTP [33]. We also compare those results with ComplEx [42] and NeuralLP [48]. We also test our model on a large knowledge graph NELL-995 in the same setting as DeepPath [46]. Lastly, we test our model on a synthetic grid world dataset.

COUNTRIES, KINSHIP, UMLS We first test our model on the COUNTRIES dataset which is explicitly designed to test the ability of models to learn logical rules. It contains countries, regions and subregions as entities. The dataset has 3 tasks (S1-3 in table 1) each requiring reasoning steps of increasing length and difficulty. We report the performance of the area under the precision-recall curve (AUC-PR) for the COUNTRIES dataset. We also compare MINERVA to NeuralLP [48] on the UMLS and KINSHIP datasets.

Task	DeepPath	MINERVA
athleteplaysinleague	0.960	**0.970**
worksfor	0.711	**0.825**
organizationhiredperson	0.742	**0.851**
athleteplayssport	0.957	**0.985**
teamplayssport	0.738	**0.846**
personborninlocation	**0.795**	0.793
athletehomestadium	0.890	**0.895**
organizationheadquarteredincity	0.790	**0.946**
athleteplaysforteam	0.750	**0.824**

Table 2: MAP scores for different query relations on the NELL dataset. Note that in this comparison, MINERVA refers to only a single learnt model for all query relations which is competitive with individual DeepPath models trained separately for each query relation.

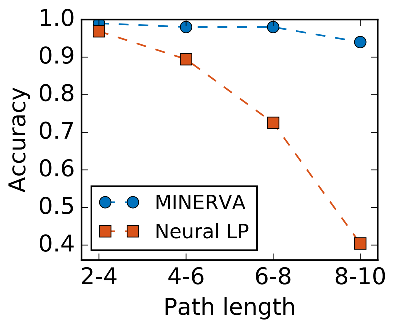

Figure 2: Grid world experiment: We significantly outperform NeuralLP for longer path lengths.

Model	UMLS	KINSHIP
NeuralLP	0.70	0.73
MINERVA	**0.91**	**0.93**

Table 3: HITS@10 on UMLS and KINSHIP

Our experimental settings and scores are directly comparable to [33] and are reported in table 1. Table 3 reports the results for UMLS and KINSHIP in which we substantially outperform NeuralLP.

NELL-995 We also compare MINERVA to DEEPPATH. For a fair comparison, we only rank the answer entities and the negative examples in the dataset released by them. But unlike them, we train one model which learns for all query relations. If our model is not able to find the correct entity or one of the negative entities, the query gets a score of negative infinity. As show in table 2, we outperform them in most and achieve comparable performance in the rest of the query relations.

GRID WORLD PATH FINDING Noted by previous work [33, 46, 8, 48], often the reasoning chains required to answer queries in KB is not too long (restricted to 3 or 4 hops). To test if our model can learn long reasoning paths, we test our model on a synthetic grid world dataset created by [48] where the task is to navigate to a particular cell (answer entity) starting from a random cell (start entity) and following a set of directions (query relation) (e.g. North, SouthWest). Figure 2 shows the accuracy on varying path lengths. Compared to NEURALLP, MINERVA is much more robust for queries which require longer path lengths showing a very little degrade in performance.

4 Related Work

Learning vector representations of entities and relations using tensor factorization [27, 28, 2, 32, 26, 47] or neural methods [37, 40, 43] has been a popular approach to reasoning with a knowledge base. However, these methods cannot capture more complex reasoning patterns such as those found by following inference paths in KBs. Multi-hop link prediction approaches [20, 25, 15, 41, 8] address the problems above, but the reasoning paths that they operate on are gathered by performing random walks independent of the type of query relation. Lao et al.,(2011) [20] further filters paths from the set of sampled paths based on the restriction that the path must end at one of the target entities in the training set and are within a maximum length. These constraints make them query dependent but they are heuristic in nature. Our approach eliminates any necessity to pre-compute paths and learns to efficiently search the graph conditioned on the input query relation.

Inductive Logic Programming (ILP) [24] aims to learn general purpose predicate rules from examples and background knowledge. Early work in ILP such as FOIL [30], PROGOL [23] are either rule-based or require negative examples which is often hard to find in KBs (by design, KBs store true facts). Statistical relational learning methods [13, 19, 35] along with probabilistic logic [31, 3, 44] combine machine learning and logic but these approaches operate on symbols rather than vectors and hence do not enjoy the generalization properties of embedding based approaches.

Neural Theorem Provers (NTP) [33] and Neural LP [48] are two recent methods in learning logical rules that can be trained end-to-end with gradient based learning. NTPs are constructed by Prolog's backward chaining inference method. It operates on vectors rather than symbols, thereby providing a success score for each proof path. However, since a score can be computed between any two vectors, the computation graph becomes quite large because of such *soft-matching* during substitution step of backward chaining. For tractability, it resides to heuristics such as only keeping the top-K scoring proof paths, but it loses any guarantee of computing exact gradients. Also the efficacy of NTPs has yet to be shown on large KBs. Neural LP introduces a differential rule learning system using operators defined in TensorLog [6] and has a LSTM based controller and a differentiable memory component [14, 39] and the rule scores are calculated via attention. Even though, differentiable memory allows the network to be trained end to end, it necessitates accessing the entire memory which can be computationally expensive. RL approaches which can make hard selection of memory [50] are computationally attractive. MINERVA uses a similar hard selection of relation edges to walk on the graph. More importantly, MINERVA outperforms both these methods on their respective benchmark datasets.

DeepPath [46] uses RL based approaches to find paths in KBs. However, the state of their MDP requires the target entity to be known in advance and hence their path finding strategy is dependent on knowing the answer entity. MINERVA does not need any knowledge of the target entity and instead learns to find the answer entity among all entities. DeepPath, additionally feeds its gathered paths to Path Ranking Algorithm [20], whereas MINERVA is a complete system trained to do query answering. DeepPath also uses fixed pretrained embeddings for its entity and relations. Lastly, on comparing MINERVA with DeepPath in their experimental setting on the NELL dataset, we match their performance or outperform them. MINERVA is also similar to methods for learning to search for structured prediction [7, 10, 9, 34, 5]. These methods are based on imitating a reference policy (oracle) which make near-optimal decision at every step. In our problem setting, it is unclear what a good reference policy would be. For example, a shortest path oracle between two entities would be bad, since the answer providing path should depend on the query relation.

5 Conclusion

As part of this ongoing work, we explored a new way of automated reasoning on large knowledge bases in which we use the knowledge graphs representation of the knowledge base and train an agent to walk to the answer node conditioned on the input query. We achieve state-of-the-art results on multiple benchmark knowledge base completion tasks and we also show that our model is robust and can learn long chains-of-reasoning. Moreover it needs no pretraining or initial supervision. Future research directions include applying more sophisticated RL techniques and working directly on textual queries and documents.

Acknowledgements

This work was supported in part by the Center for Data Science and the Center for Intelligent Information Retrieval, in part by DARPA under agreement number FA8750-13-2-0020, in part by Defense Advanced Research Agency (DARPA) contract number HR0011-15-2-0036, in part by the National Science Foundation (NSF) grant numbers DMR-1534431 and IIS-1514053 and in part by the Chan Zuckerberg Initiative under the project Scientific Knowledge Base Construction. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

References

[1] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *ICDM*, 2008.

[2] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, 2013.

[3] Matthias Broecheler, Lilyana Mihalkova, and Lise Getoor. Probabilistic similarity logic. In *UAI*, 2010.

[4] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka, Jr., and Tom M. Mitchell. Toward an Architecture for Never-ending Language Learning. In *AAAI*, 2010.

[5] Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume, and John Langford. Learning to search better than your teacher. In *ICML*, 2015.

[6] William W Cohen. Tensorlog: A differentiable deductive database. *arXiv preprint arXiv:1605.06523*, 2016.

[7] Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *ACL*, 2004.

[8] Rajarshi Das, Arvind Neelakantan, David Belanger, and Andrew McCallum. Chains of reasoning over entities, relations, and text using recurrent neural networks. In *EACL*, 2017.

[9] Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 2009.

[10] Hal Daumé III and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *ICML*, 2005.

[11] Michael Evans and Timothy Swartz. *Approximating integrals via Monte Carlo and deterministic methods*, volume 20. OUP Oxford, 2000.

[12] George Fishman. *Monte Carlo: concepts, algorithms, and applications*. Springer Science & Business Media, 2013.

[13] Lise Getoor and Ben Taskar. *Introduction to statistical relational learning*. MIT press, 2007.

[14] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv:1410.5401*, 2014.

[15] Kelvin Guu, John Miller, and Percy Liang. Traversing knowledge graphs in vector space. In *EMNLP*, 2015.

[16] John Hammersley. *Monte carlo methods*. Springer Science & Business Media, 2013.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[18] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] Stanley Kok and Pedro Domingos. Statistical predicate invention. In *ICML*, 2007.

[20] Ni Lao, Tom Mitchell, and William W Cohen. Random walk inference and learning in a large scale knowledge base.

[21] John McCarthy. *Programs with common sense*. RLE and MIT Computation Center, 1960.

[22] Bonan Min, Ralph Grishman, Li Wan, Chang Wang, and David Gondek. Distant supervision for relation extraction with an incomplete knowledge base. In *HLT-NAACL*, 2013.

[23] Stephen Muggleton. Inverse entailment and progol. *New generation computing*, 1995.

[24] Stephen Muggleton, Ramon Otero, and Alireza Tamaddoni-Nezhad. *Inductive logic programming*. Springer, 1992.

[25] Arvind Neelakantan, Benjamin Roth, and Andrew McCallum. Compositional vector space models for knowledge base completion. In *ACL*, 2015.

[26] Maximilian Nickel, Xueyan Jiang, and Volker Tresp. Reducing the rank in relational factorization models by including observable patterns. In *NIPS*, 2014.

[27] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, 2011.

[28] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. Factorizing yago: scalable machine learning for linked data. In *WWW*, 2012.

[29] Nils J Nilsson. Logic and artificial intelligence. *Artificial intelligence*, 1991.

[30] J Ross Quinlan. Learning logical definitions from relations. *Machine learning*, 1990.

[31] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 2006.

[32] Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M. Marlin. Relation extraction with matrix factorization and universal schemas. In *NAACL*, 2013.

[33] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *NIPS*, 2017.

[34] Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

[35] Stefan Schoenmackers, Oren Etzioni, Daniel Weld, and Jesse Davis. Learning first-order horn clauses from web text. In *EMNLP*, 2010.

[36] Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. In *KDD*, 2017.

[37] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *NIPS*, 2013.

[38] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A core of semantic knowledge. In *WWW*, 2007.

[39] Sainbayar Sukhbaatar, Jason Weston, and Rob Fergus. End-to-end memory networks. In *NIPS*, 2015.

[40] Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. Representing text for joint embedding of text and knowledge bases. In *EMNLP*, 2015.

[41] Kristina Toutanova, Victoria Lin, Wen-tau Yih, Hoifung Poon, and Chris Quirk. Compositional learning of embeddings for relation paths in knowledge base and text. In *ACL*, 2016.

[42] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *ICML*, 2016.

[43] Patrick Verga, David Belanger, Emma Strubell, Benjamin Roth, and Andrew McCallum. Multilingual relation extraction using compositional universal schema. In *NAACL*, 2016.

[44] William Yang Wang, Kathryn Mazaitis, and William W Cohen. Programming with personalized pagerank: a locally groundable first-order probabilistic logic. In *CIKM*, 2013.

[45] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.

[46] Wenhan Xiong, Thien Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *EMNLP*, 2017.

[47] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR*, 2015.

[48] Fan Yang, Zhilin Yang, and William W Cohen. Differentiable learning of logical rules for knowledge base reasoning. 2017.

[49] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets. 2017.

[50] Wojciech Zaremba and Ilya Sutskever. Reinforcement learning neural turing machines. *arXiv:1505.00521*, 2015.