Learning Knowledge Base Inference with Neural Theorem Provers

Tim Rocktäschel and **Sebastian Riedel**
University College London
London, UK
{t.rocktaschel,s.riedel}@cs.ucl.ac.uk

Abstract

In this paper we present a proof-of-concept implementation of *Neural Theorem Provers* (NTPs), end-to-end differentiable counterparts of discrete theorem provers that perform first-order inference on vector representations of symbols using function-free, possibly parameterized, rules. As such, NTPs follow a long tradition of neural-symbolic approaches to automated knowledge base inference, but differ in that they are differentiable with respect to representations of symbols in a knowledge base and can thus learn representations of predicates, constants, as well as rules of predefined structure. Furthermore, they still allow us to incorporate domain-knowledge provided as rules. The NTP presented here is realized via a differentiable version of the backward chaining algorithm. It operates on substitution representations and is able to learn complex logical dependencies from training facts of small knowledge bases.

1 Introduction

Current state-of-the-art methods for automated knowledge base (KB) construction learn distributed representations of fact triples (Nickel et al., 2012; Riedel et al., 2013; Socher et al., 2013; Chang et al., 2014; Neelakantan et al., 2015; Toutanova et al., 2015). An open question is how to enable first-order reasoning with commonsense knowledge (Nickel et al., 2015). We believe a promising direction towards this goal is the integration of deep neural networks with the capabilities of theorem provers. Neural networks can learn to generalize well when observing many input-output examples, but lack interpretability and straightforward ways of incorporating domain-specific knowledge. Theorem provers on the other hand provide effective ways to reason with logical knowledge. However, by operating on discrete symbols they do not make use of similarities between predicates or constants in training data (*e.g.*, LECTURERAT $\sim$ PROFESSORAT, ORANGE $\sim$ LEMON, etc).

Recent neural network architectures such as Neural Turing Machines (Graves et al., 2014, NTMs), Memory Networks (Weston et al., 2015b), Neural Stacks/Queues (Grefenstette et al., 2015; Joulin and Mikolov, 2015), Neural Programmer (Neelakantan et al., 2016), Neural Programmer-Interpreters (Reed and de Freitas, 2016) and Hierarchical Attentive Memory (Andrychowicz and Kurach, 2016) replace discrete functions and data structures by end-to-end differentiable counterparts. As such, they can learn complex behaviour from raw input-output examples via gradient-based optimization.

NTMs and their relatives are capable of learning programs and could in principle learn to emulate a theorem prover. However, they might not be the most efficient neural architecture for learning first-order reasoning from input-output examples. Akin to NTMs, which are end-to-end differentiable counterparts of Turing machines, we investigate *Neural Theorem Provers* (NTPs): end-to-end differentiable versions of automated theorem provers. A distinguishing property of NTPs is that they are differentiable with respect to symbol representations in a knowledge base. This enables us to learn representations of symbols in ground atoms (predicates and constants) and parameters of first-order rules of predefined structure using backpropagation. Furthermore, NTPs can seamlessly reason with provided domain-specific rules. As NTPs operate on distributed representations of symbols, a single handcrafted rule can be leveraged for many proofs of queries with similar symbol representations. Finally, NTPs allow for a high degree of interpretability by providing such proofs.

Our contributions are threefold: (i) we present the construction of an NTP based on differentiable backward chaining and unification, (ii) we show that when provided with rules this NTP can perform first-order inference in vector space like a discrete theorem prover would do on symbolic representations, and (iii) we demonstrate that NTPs can learn representations of symbols and first-order rules of predefined structure.

2 Related Work

Combining neural and symbolic approaches for relational learning and reasoning has let to many promising neural network architectures over the past decades (Garcez et al., 2012). Early proposals for neural-symbolic networks are limited to *propositional formulae* (*e.g.*, EBL-ANN (Shavlik and Towell, 1989), KBANN (Towell and Shavlik, 1994) and C-ILP (Garcez and Zaverucha, 1999)). Other neural-symbolic approaches focus on first-order inference, but do not allow one to learn vector representations of symbols from training facts of a KB (*e.g.*, SHRUTI (Shastri, 1992), Neural Prolog (Ding, 1995), CLIP++ (França et al., 2014) and Lifted Relational Neural Networks (Sourek et al., 2015)). Neural Reasoner (Peng et al., 2015) translates query representations in vector space without rule representations and can thus not incorporate domain-specific knowledge. Rocktäschel et al. (2014), Rocktäschel et al. (2015), Vendrov et al. (2016) and Hu et al. (2016) regularize distributed representations via domain-specific rules, but do not learn such rules from data and only support a restricted subset of first-order rules. The NTP proposed here builds upon differentiable backward chaining and is thus related to Unification Neural Networks (Komendantskaya, 2011; Hölldobler, 1990), but operates on vector representations of symbols instead of scalar values. Yin et al. (2015) and Andreas et al. (2016) map queries to multiple differentiable modules that can be used to retrieve answers from a KB. Clark et al. (2014) extract common-sense knowledge from textbooks in form of rules to improve KB inference by soft-matching and non-recursive forward inference. Lee et al. (2016) propose a Tensor Product Representation to answer Facebook bAbI (Weston et al., 2015a) questions. Gu et al. (2015) traverse KBs in vector space to answer queries. Socher et al. (2012) and Bowman et al. (2015) demonstrate that recursive neural networks can learn to evaluate propositional logic expressions.

3 Differentiable Backward Chaining

Backward chaining is a common method for automated theorem proving, and we refer the reader to Russell and Norvig (1995) for details. Given a goal/query (*e.g.* GRANDPARENTOF(X, Y)), backward chaining finds substitutions of free variables with constants of facts in a KB (*e.g.* $\{X/\text{ABE}, Y/\text{BART}\}$). This is achieved by recursively iterating through rules that translate a goal into sub-goals which it attempts to prove, thereby exploring possible proofs. For example, the KB could contain the following rule that can be applied to find answers for the above goal: $\forall X, Y, Z : \text{PARENTOF}(X,Y) \wedge \text{PARENTOF}(Y,Z) \Rightarrow \text{GRANPARENTOF}(X,Z)$. For the rest of the paper we assume all free variables are universally quantified. Furthermore, we call the conjunction of atoms before the implication symbol the left-hand side (or body) of the rule and the atom after the implication the right-hand side (or head) of the rule.

The proof exploration in backward-chaining is divided into two functions called $\underline{\text{OR}}$ and $\underline{\text{AND}}$. The former attempts to prove a goal by unifying it with every rule's right-hand side in a KB, yielding intermediate substitutions. For rules where this succeeds, the left-hand side and substitution is passed to the $\underline{\text{AND}}$ function. $\underline{\text{AND}}$ then attempts to prove every atom in the body sequentially by first applying substitutions and subsequently calling $\underline{\text{OR}}$. This is repeated recursively until unification fails, atoms are proven by unification with facts in the KB, or a certain proof-depth is exceeded.

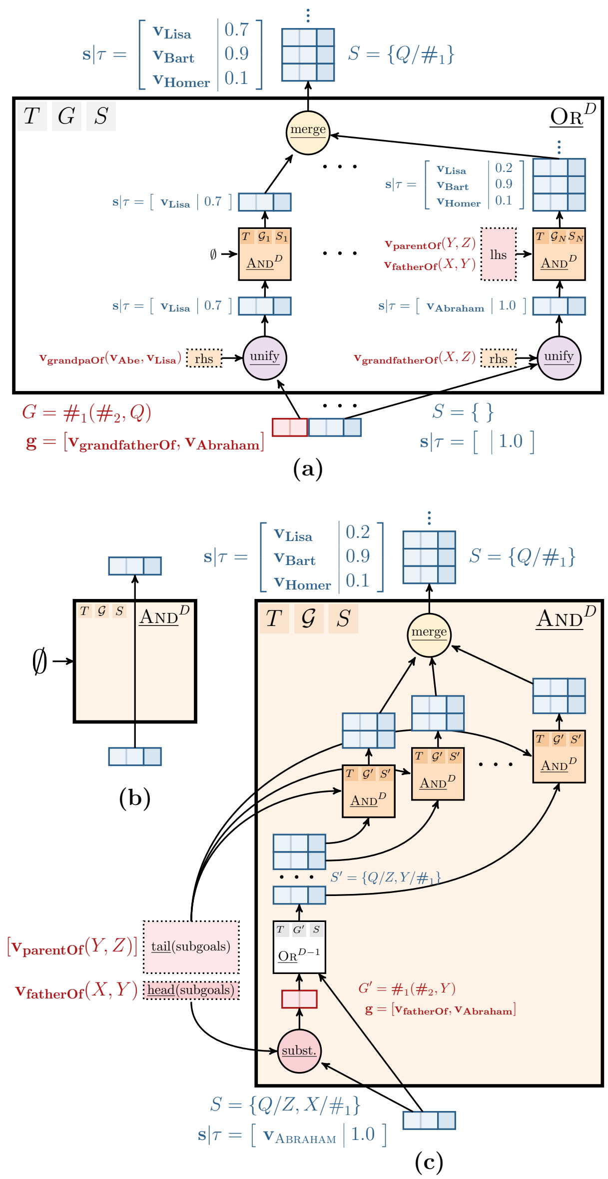

Figure 1: Overview of differentiable backward chaining.

Goal and Substitution Structures The key idea behind the proof-of-concept NTP presented here is to recursively construct a neural network by replacing operations on symbols in backward chaining with differentiable operations on distributed representations. To build such a network we separate goals and substitutions into *vector representations* of involved predicates and constants, and *structures* that define the connections of a neural network.

For instance, $G = \#_1(\#_2, X)$ is an example of a structure of an entire *class* of goals. This structure encodes that such goals encompass a vector representation of a predicate symbol $\#_1$ and the first argument of the predicate $\#_2$. For example, the goal $\text{GRANDPAOF}(\text{ABE}, X)$ can be specified by G and vector representations $\mathbf{g} = [\mathbf{v}_{\text{GRANDPAOF}}, \mathbf{v}_{\text{ABE}}]$. Furthermore, based on the structure G it is clear that proofs of that goal will be substitutions for X (*e.g.* $\mathbf{v}_{\text{BART}}$). Akin to goals, we divide substitutions into structures and representations, as well as a scalar score $\tau \in (0, 1)$ that measures the success of the substitution. For example, proofs of goals of the structure G as defined above will be substitutions with the structure $S = \{X/\#_1\}$ accompanied by substitution representations (*e.g.* $\mathbf{s} = [\mathbf{v}_{\text{BART}}]$).

With this divide we can now redefine operations in backward chaining as follows. Operations that concern variables and rules are mapping goal and substitution structures (G and S) to new structures that instantiate sub-networks. In contrast, operations on symbols of predicates and constants can be computed in vector space in a differentiable manner. The resulting recursively constructed NTP is end-to-end differentiable. An overview of the model architecture with an example is given in Figure 1 and discussed in detail below.

OR The entry point to the NTP is an OR network (Figure 1a) that for a given goal and substitution structure (G and S) instantiates a sub-network for each one of the N rules in a knowledge base T. The unification of the ith rule's right-hand side with a goal structure results in a new substitution structure S_i. When provided with a goal representation, a unification network is computing the unification success in vector space. For example, assume at some proof-depth D in the NTP we unify $\text{GRANDFATHEROF}(\text{ABRAHAM}, Q)$ with $\text{GRANDPAOF}(\text{ABE}, \text{LISA})$. This will result in a new substitution structure $S'_i = \{Q/\#_1\}$, representation $\mathbf{s} = [\mathbf{v}_{\text{LISA}}]$ and success τ^D that is passed further in the network. In contrast to discrete unification that checks for symbol equality, we calculate a soft unification from the previous unification success of the outer network τ^{D+1} and the similarity of predicate and constant representations as follows:

$$\tau_{\text{predicate}} = \text{sigmoid}(\mathbf{v}_{\text{GRANDFATHEROF}}^T \mathbf{v}_{\text{GRANDPA}}) \quad (1)$$

$$\tau_{\text{arg}_1} = \text{sigmoid}(\mathbf{v}_{\text{ABRAHAM}}^T \mathbf{v}_{\text{ABE}}) \quad (2)$$

$$\tau^D = \min(\tau^{D+1}, \tau_{\text{predicate}}, \tau_{\text{arg}_1}) \quad (3)$$

AND The new substitution structure calculated by unification instantiates an AND network (Figure 1c) at depth D that attempts to sequentially prove the left-hand side atoms of the rule given the current substitutions. If the rule's left-hand side structure is empty (*e.g.* when the right-hand side represents a fact in the KB) the AND network simply passes the substitutions and their success through (Figure 1b). Otherwise, it applies the substitution on the first atom of the left-hand side, resulting in a new goal structure and representation, and instantiates an OR network with that structure and the previous substitution.

For example, assume we have unified the right-hand side of $\text{FATHEROF}(X, Y) \wedge \text{PARENTOF}(Y, Z) \Rightarrow \text{GRANDFATHEROF}(X, Z)$ with the goal $\text{GRANDFATHEROF}(\text{ABRAHAM}, Q)$. The result is a unification success τ^D as calculated in Eq. 3, as well as a new substitution structure $S = \{Q/Z, X/\#_1\}$ where $\mathbf{s} = [\mathbf{v}_{\text{ABRAHAM}}]$ becomes the input to the AND network. This network will first apply the substitution to $\text{FATHEROF}(X, Y)$, resulting in a new goal structure $G' = \#_1(\#_2, Y)$. This structure now instantiates another NTP (*i.e.* an OR module) of depth $D - 1$, which attempts to prove the input goal representation $\mathbf{g} = [\mathbf{v}_{\text{FATHEROF}}, \mathbf{v}_{\text{ABRAHAM}}]$.

For every proof, *i.e.*, every possible substitution of the structure $S' = \{Q/Z, Y/\#_1\}$, a new AND module is instantiated that attempts to recursively prove the remainder of the left-hand side ($\text{PARENTOF}(Y, Z)$ in the example above). Finally, the successes of all identical substitutions (*i.e.* substitutions to the same variables or representations of constants) are merged by taking their max.

Note that given a KB, goal structure and depth, the network structure of the NTP is fully specified and many goals of the same structure can be used to perform training and inference with the NTP.

Trainable Rules NTPs are not only differentiable with respect to symbol representations in the KB, but also latent symbol representations in first-order rules of predefined structure. For instance, we could assume that for some predicates in a KB a transitive relationship holds. We can define a rule template $\#_1(X, Y) \wedge \#_1(Y, Z) \Rightarrow \#_2(X, Z)$ whose latent predicates $\mathbf{v}_{\#_1}, \mathbf{v}_{\#_2}$ are trainable parameters and optimized in the same way as symbol representations.

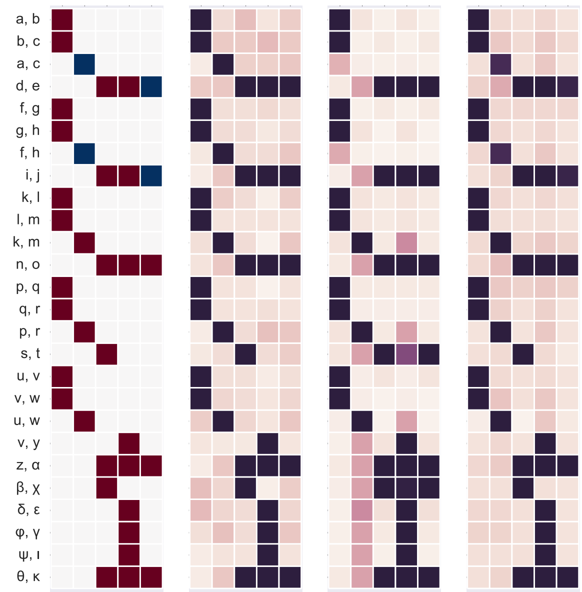

Figure 2: Predictions of different NTP modes on a toy KB where every column (within a subplot) represents a predicate and every row an entity-pair. Training facts (red) and test facts (blue) in the first subplot are consistent with two rules: $r_1(X, Y) \wedge r_1(Y, Z) \Rightarrow r_2(X, Z)$ and $r_3(X, Y) \wedge r_4(X, Y) \Rightarrow r_5(X, Y)$. The other three subplots show predictions between 0 (white) and 1 (black) of the different modes discussed in text.

4 Experiments and Results

We implemented an NTP with differentiable backward chaining in TensorFlow (Abadi et al., 2015). Symbol representations are initialized randomly and constrained to unit-length. During training we iterate over the set of known facts, and optimize negative log-likelihood of the proof success of every fact based on all other facts (and rules) using Adam (Kingma and Ba, 2015). Furthermore, for every training fact we sample an unobserved fact for the same predicate (but different entity-pair) and optimize its proof with a target success of zero.

Our NTP implementation is tested on toy KBs for different scenarios shown in the four different subplots in Figure 2. Every column (within a subplot) represents a predicate and every row an entity-pair. First, we run the NTP with given ground-truth rules without training symbol or rule representations, and test whether it can act as a discrete theorem prover. As expected, given rules the NTP can infer all test facts (2nd subplot in Figure 2). The third subplot shows predictions when we let the NTP try to reconstruct training facts only with the help of other facts by learning symbol representations (similar to other representation learning approaches for KB inference). Finally, a core benefit of the NTP is visible once we provide few reasonable rule templates[1] and optimize for rule representations that best explain observed facts (4th subplot). We found that this can work remarkably well, but also noticed that the quality of trained rules is varying with different random initialization of the rule's parameters. We need to investigate in future work how the robustness of rule learning in NTPs can be improved.

5 Conclusion and Future Work

We proposed neural theorem provers for knowledge base inference via differentiable backward chaining, which enables learning of symbol representations and parameters of rules of predefined structure.

Our current implementation has severe computational limitations and does not scale to larger KBs as it investigates all possible proof paths. However, there are many possibilities to improve upon the presented architecture. For instance, one can batch-unify all rules whose right-hand side have the same structure and employ existing architectures such as Memory Networks or hierarchical attention for this task. Furthermore, it is possible to partition and batch rules not only by their right-hand side but also left-hand side structure to instantiate a single AND module for every partition. To further speed-up the prover, we want to investigate processing batches of queries, as well as differentiable ways of maintaining only the N best instead of all possible substitution representations at every depth of the prover. In addition, we will work on more flexible versions of neural theorem provers, for instance, where unification, rule selection and application itself are trainable functions, or where facts in a KB and goals can be natural language sentences.

Acknowledgments

We thank Isabelle Augenstein, Dirk Weissenborn, Johannes Welbl and the reviewers for comments on drafts of this paper. This work was supported by Microsoft Research through its PhD Scholarship Programme and an Allen Distinguished Investigator Award.

[1]We use $\#_1(X,Y) \Rightarrow \#_2(X,Y)$, $\#_1(X,Y) \wedge \#_2(X,Y) \Rightarrow \#_3(X,Y)$ and $\#_1(X,Y) \wedge \#_1(Y,Z) \Rightarrow \#_2(X,Z)$.

References

[Abadi et al.2015] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2015. Tensorflow: Large-scale machine learning on heterogeneous systems.

[Andreas et al.2016] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Learning to compose neural networks for question answering. In *NAACL*.

[Andrychowicz and Kurach2016] Marcin Andrychowicz and Karol Kurach. 2016. Learning efficient algorithms with hierarchical attentive memory. *arXiv preprint arXiv:1602.03218*.

[Bowman et al.2015] Samuel R Bowman, Christopher Potts, and Christopher D Manning. 2015. Recursive neural networks can learn logical semantics. In *CVSC*.

[Chang et al.2014] Kai-Wei Chang, Wen-tau Yih, Bishan Yang, and Christopher Meek. 2014. Typed tensor decomposition of knowledge bases for relation extraction. In *EMNLP*.

[Clark et al.2014] Peter Clark, Niranjan Balasubramanian, Sumithra Bhakthavatsalam, Kevin Humphreys, Jesse Kinkead, Ashish Sabharwal, and Oyvind Tafjord. 2014. Automatic construction of inference-supporting knowledge bases. In *AKBC*.

[Ding1995] Liya Ding. 1995. Neural prolog-the concepts, construction and mechanism. In *3rd Int. Conference Fuzzy Logic, Neural Nets, and Soft Computing*.

[França et al.2014] Manoel VM França, Gerson Zaverucha, and Artur S dAvila Garcez. 2014. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine learning*, 94(1):81–104.

[Garcez and Zaverucha1999] Artur S d'Avila Garcez and Gerson Zaverucha. 1999. The connectionist inductive learning and logic programming system. *Applied Intelligence*, 11(1):59–77.

[Garcez et al.2012] Artur S d'Avila Garcez, Krysia Broda, and Dov M Gabbay. 2012. *Neural-symbolic learning systems: foundations and applications*. Springer.

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

[Grefenstette et al.2015] Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. 2015. Learning to transduce with unbounded memory. In *NIPS*.

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

[Hölldobler1990] S Hölldobler. 1990. A structured connectionist unification algorithm. In *AAAI*.

[Hu et al.2016] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. 2016. Harnessing deep neural networks with logic rules. *arXiv preprint arXiv:1603.06318*.

[Joulin and Mikolov2015] Armand Joulin and Tomas Mikolov. 2015. Inferring algorithmic patterns with stack-augmented recurrent nets. In *NIPS*.

[Kingma and Ba2015] Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *ICLR*.

[Komendantskaya2011] Ekaterina Komendantskaya. 2011. Unification neural networks: unification by error-correction learning. *Logic Journal of IGPL*, 19(6):821–847.

[Lee et al.2016] Moontae Lee, Xiaodong He, Wen-tau Yih, Jianfeng Gao, Li Deng, and Paul Smolensky. 2016. Reasoning in vector space: An exploratory study of question answering. In *ICLR*.

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

[Neelakantan et al.2016] Arvind Neelakantan, Quoc V Le, and Ilya Sutskever. 2016. Neural programmer: Inducing latent programs with gradient descent. In *ICLR*.

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

[Nickel et al.2015] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. 2015. A review of relational machine learning for knowledge graphs: From multi-relational link prediction to automated knowledge graph construction. *IEEE*.

[Peng et al.2015] Baolin Peng, Zhengdong Lu, Hang Li, and Kam-Fai Wong. 2015. Towards neural network-based reasoning. In *RAM*.

[Reed and de Freitas2016] Scott Reed and Nando de Freitas. 2016. Neural programmer-interpreters. In *ICLR*.

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

[Rocktäschel et al.2014] Tim Rocktäschel, Matko Bosnjak, Sameer Singh, and Sebastian Riedel. 2014. Low-dimensional embeddings of logic. In *SP14*.

[Rocktäschel et al.2015] Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. 2015. Injecting Logical Background Knowledge into Embeddings for Relation Extraction. In *NAACL*.

[Russell and Norvig1995] Stuart J Russell and Peter Norvig. 1995. *Artificial Intelligence: Modern Approach*. Prentice Hall, 3rd edition.

[Shastri1992] Lokendra Shastri. 1992. Neurally motivated constraints on the working memory capacity of a production system for parallel processing: Implications of a connectionist model based on temporal synchrony. In *Conference of the Cognitive Science Society*. Psychology Press.

[Shavlik and Towell1989] Jude W Shavlik and Geoffrey G Towell. 1989. An approach to combining explanation-based and neural learning algorithms. *Connection Science*, 1(3):231–253.

[Socher et al.2012] Richard Socher, Brody Huval, Christopher D Manning, and Andrew Y Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *EMNLP*.

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

[Sourek et al.2015] Gustav Sourek, Vojtech Aschenbrenner, Filip Zelezny, and Ondrej Kuzelka. 2015. Lifted relational neural networks. *arXiv preprint arXiv:1508.05128*.

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

[Towell and Shavlik1994] Geoffrey G Towell and Jude W Shavlik. 1994. Knowledge-based artificial neural networks. *Artificial intelligence*, 70(1):119–165.

[Vendrov et al.2016] Ivan Vendrov, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. 2016. Order-embeddings of images and language. In *ICLR*.

[Weston et al.2015a] Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. 2015a. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*.

[Weston et al.2015b] Jason Weston, Sumit Chopra, and Antoine Bordes. 2015b. Memory networks. In *ICLR*.

[Yin et al.2015] Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. 2015. Neural enquirer: Learning to query tables. *arXiv preprint arXiv:1512.00965*.