Traversing Knowledge Graph in Vector Space without Symbolic Space Guidance

Yelong Shen[†*] and **Po-Sen Huang**[†*] and **Ming-Wei Chang**[‡] and **Jianfeng Gao**[†]
[†]Microsoft Research, USA; [‡]Google Research, USA
{yeshen, pshuang, jfgao}@microsoft.com; changmingwei@gmail.com

Abstract

Recent studies on knowledge base completion, the task of recovering missing facts based on observed facts, demonstrate the importance of learning embeddings from multi-step relations. Due to the size of knowledge bases, previous works manually design relation paths of observed triplets in symbolic space (e.g. random walk) to learn multi-step relations during training. However, these approaches suffer some limitations as most paths are not informative, and it is prohibitively expensive to consider all possible paths. To address the limitations, we propose learning to traverse in vector space directly without the need of symbolic space guidance. To remember the connections between related observed triplets and be able to adaptively change relation paths in vector space, we propose Implicit ReasoNets (IRNs), that is composed of a global memory and a controller module to learn multi-step relation paths in vector space and infer missing facts jointly without any human-designed procedure. Without using any axillary information, our proposed model achieves state-of-the-art results on popular knowledge base completion benchmarks.

Introduction

Knowledge bases such as WordNet (Fellbaum 1998), Freebase (Bollacker et al. 2008), or Yago (Suchanek, Kasneci, and Weikum 2007) contain many real-world facts expressed as triples, e.g., (Bill Gates, FOUNDEROF, Microsoft). These knowledge bases are useful for many downstream applications such as question answering (Berant et al. 2013; Yih et al. 2015) and information extraction (Mintz et al. 2009). However, despite the formidable size of knowledge bases, many important facts are still missing. For example, West et al. (2014) showed that 21% of the 100K most frequent PERSON entities have no recorded nationality in a recent version of Freebase. We seek to infer unknown facts based on the observed triplets. Thus, the knowledge base completion (KBC) task has emerged as an important open research problem (Nickel, Tresp, and Kriegel 2011).

Knowledge graph embedding-based methods have been popular for tackling the KBC task. In this framework, entities and relations are represented in continuous representations. To infer whether two entities have a missing relationship, it can be predicted by functions of their corresponding representations. In other words, to predict whether a given edge (h, R, t) belongs in the graph, it can be formulated as predicting a candidate entity t given an input triplet (h, R, ?), which consists of an entity h, a relation R, and a missing entity ?, denoted as [h, R].

Early work (Bordes et al. 2013; Socher et al. 2013; Yang et al. 2015) focuses on exploring different objective functions to model *direct* relationships between two entities such as (Hawaii, PARTOF, USA). Several recent approaches demonstrate limitations of prior approaches relying upon vector-space models alone (Guu, Miller, and Liang 2015; Toutanova et al. 2016; Lin et al. 2015a). For example, when dealing with *multi-step* (compositional) relationships (e.g., (Obama, BORNIN, Hawaii) $\wedge$ (Hawaii, PARTOF, USA)), direct relationship-models suffer from cascading errors when recursively applied their answer to the next input (Guu, Miller, and Liang 2015). Hence, recent works (Gardner et al. 2014; Neelakantan, Roth, and McCallum 2015; Guu, Miller, and Liang 2015; Neelakantan, Roth, and McCallum 2015; Lin et al. 2015a; Das et al. 2016; Wang and Cohen 2016; Toutanova et al. 2016) propose different approaches of injecting multi-step relation paths from observed triplets during training, which further improve performance in KBC tasks.

Although using multi-step relation paths achieves better performance, it also introduces some technical challenges. Since the number of possible paths grows exponentially with the path length, it is prohibitive to consider all possible paths during the training time for knowledge bases such as FB15K (Bordes et al. 2013). Existing approaches need to use human-designed procedures (e.g., random walk) for sampling or pruning paths of observed triplets in the symbolic space. As most paths are not informative for inferring missing relations, these approaches might be suboptimal.

In this paper, we aim to rich neural knowledge completion models without the need of traversing the knowledge in the symbolic space. Instead of using human-designed relation paths in symbolic space and training a model separately, we propose learning relation paths in vector space jointly with model training without using any auxiliary information. For a motivating example, a relation path in symbolic space ((Obama, BORNIN, Hawaii) $\wedge$ (Hawaii, PARTOF, USA)) provides a *connection* between two observed triplets and the *path length* between the entities Obama and USA. Therefore, to traverse in vector space without relation paths from symbolic space, a model needs to satisfy both proper-

[*]Equal contribution.

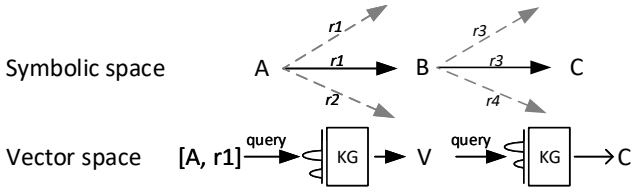


Figure 1: An illustrating example between knowledge base relation paths in symbolic space and vector space, where A, B, and C are entities and R1, R2, R3, and R4 are relations. "KG" stands for knowledge graph and V is an intermediate vector representation. In symbolic space, early works subsample paths by looking up one relation among the knowledge graph at each step in the relation paths. In vector space, we can generalize the look-up operations as a query over the knowledge graph and use the resulting vector as the next representation.

ties: remember the connections between related triplets and be able to adaptively change relation path length based on the input. Note that a "connection" between triplets can be either sharing the same intermediate entity in a knowledge graph or semantically related. Although it requires a model with a higher capacity to achieve the desired two properties, it relaxes the constraints that two triplets need to share the same intermediate entity and resolves the limitations of subsampling paths for traversing in symbolic space. Figure 1 shows a comparison between traversing knowledge graph in symbolic space and in vector space.

In this paper, we propose Implicit ReasoNets (IRNs), that capture the desired properties by the design of *global memory* and *controller*. We design the global memory to learn a compact representation of the knowledge graph that captures the connections between related triplets. We design the controller to adaptively change relation paths and determine path length in relation paths, as shown in Fig. 2. Specifically, to adaptively adjust relation path length, the controller, modeled by an RNN, uses a termination module to model whether IRNs should stop or how many steps IRNs should proceed given an input. To update vector representations in each step of a relation path, the controller updates its next state by using its current state and an attention vector over the global memory. The global memory is an external weight matrix that is shared across all triplets. Given limited size of the global memory, the global memory cannot store all observed triplets explicitly. The global memory is forced to learn to store new information and reuse/relate previously saved relevant information. Hence, an attention vector over the global memory provides some context information about the knowledge graph for the current state of the controller. The controller uses the context information to update the next step representation in a relation path. We name the model "Implicit" given the exact information stored in the global memory is implicit and the exact relation path of the controller is also implicit. We will provide some interpretations and analysis of these implicit components in the experimental section.

The main contributions of our paper are as follows:

- We propose Implicit ReasoNets (IRNs), that learn to traverse knowledge graph in vector space and infer missing triplets jointly using a global memory guided by a controller.
- We evaluate IRNs and demonstrate that our proposed model achieves the state-of-the-art results on the WN18, FB15k, and FB15k-237 benchmarks without using auxiliary information.
- Our analysis provides ways to understand the inference procedure by Implicit ReasoNets.

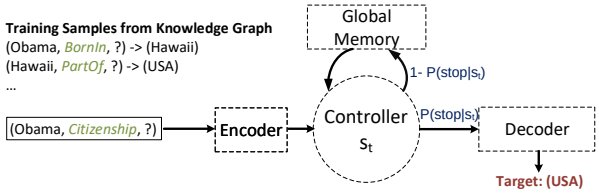


Figure 2: An overview of the IRN for KBC tasks. The global memory is designed to store a compact representation of the knowledge graph that connections between related triplets. The controller is designed to adaptively change relation paths in vector space and determine the length of relation paths.

Knowledge Base Completion Task

The goal of Knowledge Base Completion (KBC) tasks is to infer a missing relationship between two entities, which can be formulated as predicting a head or a tail entity given the relation type and the other entity. Early work on KBC focuses on learning symbolic rules. Schoenmackers et al. (2010) learns inference rules from a sequence of triplets, e.g., (X, COUNTRYOFHEADQUARTERS, Y) is implied by (X, ISBASEDIN, A) and (A, STATELOCATEDIN, B) and (B, COUNTRYLOCATEDIN, Y). However, enumerating all possible relations is intractable when the knowledge base is large, since the number of distinct sequences of triplets increases rapidly with the number of relation types. Also, the rules-based methods cannot be generalized to paraphrase alternations.

More recently, several approaches (Bordes et al. 2013; Socher et al. 2013; Yang et al. 2015) achieve better generalization by operating on embedding representations, where the vector similarity can be regarded as semantic similarity. During training, models learn a scoring function that optimizes the score of a target entity for a given triplet. In the evaluation, models are given a triplet with a missing entity, $(h, R, ?)$, as an input, then maps the input into the vector space through embeddings, and finally outputs a prediction vector of the missing entity. We then compute the similarity between the prediction vector and all candidate entities, which results in a ranked list of the entities. Mean rank and precision of the target entity are used as metrics for evaluation. In this paper, our proposed model uses the same setup as in the embedding type of approaches (Bordes et al. 2013; Socher et al. 2013; Yang et al. 2015).

Proposed Model

The overview of the proposed model is as follows. We use the *encoder* module to transform an input $[h, R]$ to a continuous representation. For generating the prediction results, the

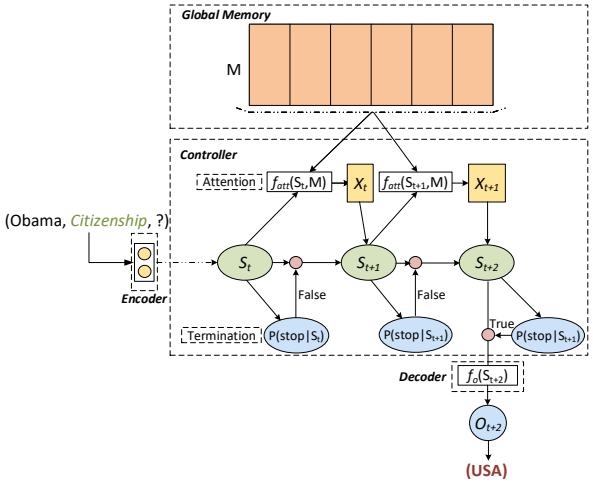


Figure 3: A running example of the IRN architecture. Given the input (Obama, CITIZENSHIP, ?), the model iteratively reformulates the input vector via the current input vector and the attention vector over the global memory, and determines to stop based on the termination module.

decoder module takes the generated continuous representation and outputs a predicted vector, which can be used to find the nearest entity embedding. Basically, we use encoder and decoder modules to convert the representations between symbolic space and vector space. Our model learns to remember the connections between relevant observed triplets implicitly in the *global memory* and to adaptively change relation paths depending on the complexity of the input guided by the *controller*. The global memory is shared across all triplets and is used to store a compact representation of the knowledge graph that captures connections between triplets. By using a termination module given current representation, the controller determines whether the model encodes enough information to produce the output prediction or not. If the model decides not to terminate, the controller learns to update its representation through its previous state and an attention vector over the global memory. Otherwise, the model uses its current representation to make a prediction as the output. The whole process is performed repeatedly until the controller stops the process. Note that the number of steps varies according to the complexity of each example. We will introduce each component of the model, the detailed algorithm, training objectives, and motivating examples in the following subsections.

Encoder/Decoder Given an input $[\mathtt{h}, \mathtt{R}]$, the encoder module retrieves the entity $\mathtt{h}$ and relation $\mathtt{R}$ embeddings from an embedding matrix, and then concatenates the two vectors as the intermediate representation s_1.

The decoder module outputs a prediction vector $f_o(s_t) = \tanh(W_o s_t + b_o)$ based on the intermediate representation s_t, which is a nonlinear projection from the controller hidden state and W_o and b_o are the weight matrix and bias vector, respectively. W_o is a k-by-n matrix, where k is the number of the possible entities, and n is the dimension of the hidden vector s_t.

Global Memory The design goal of the global memory is to store the knowledge graph information in a compact way. The global memory stores the connections between related observed triplets and provides context information to the controller for updating relation paths. The global memory is denoted as $M = \{m_i\}_{i=1}^{|M|}$, which consists of a list of vectors shared across all triplets and is randomly initialized. During training, the global memory is accessed through the attention mechanism in Eq. 1. Note that we do not need to define write operations since the global memory is updated w.r.t. the training objectives in Eq. 2 through back-propagation. During inference, we fix the learned global memory and access the global memory using the attention mechanism.

A motivating example for the design of the global memory is as follows. Suppose, in a KBC task, the input is [Obama, NATIONALITY] and the model is required to answer the missing entity (answer: USA). Our model can learn to utilize and store information in the global memory through the controller. In this example, in order to answer the query correctly, we need to collect three pieces of relevant information: (1) the triplet (Obama, BORNIN, Hawaii), (2) the triplet (Hawaii, PARTOF, USA), and (3) knowing that BORNIN and NATIONALITY are semantically correlated relations (i.e., nationality depends on country where he is born). Assume in this case, the controller can find (1) and (2) in the global memory, but not (3); thus fails to get the correct answer. Then, in training, the global memory M is updated to store the new information of (3), via gradient update. E.g., vectors are updated in a way such that the distance between the two relations (BORNIN and NATIONALITY) are closer in the vector space. Actually, the limited size of M ($|M|$=64 in our experiments) has an effect similar to a regularizer in model training, which forces IRNs to make the information stored in M as reusable and general as possible. As a result, M is a compact representation of the knowledge graph, where semantically related/similar triples, entities and relations are "condensed" to similar vectors in vector space, as we will illustrate using examples in Table 4. Similarly, when the model encounters (1) or (2) in the first time, the model learns to correlate the input with the global memory, which minimizes the distance between the attended vectors in the global memory and the input triplet. In this case, the new triplet information is stored in the global memory indirectly by gradient update. Hence, when the input [Obama, NATIONALITY] is received, the model can achieve (1), (2), and (3) by accessing the global memory. In this way, the model can learn to compactly store the knowledge graph and correlate observed triplets through the global memory.

Controller The design goal of the controller is to adaptively change relation paths in vector space and determinate the length of relation paths. The controller can iteratively reformulate its representation through incorporating context information retrieved from the global memory. Without explicitly providing human-designed relation paths, during the iterative progress, the controller needs to explore the multi-step relation paths on its own. Suppose a given input triplet is not able to be resolved in one step. The controller needs to utilize its reformulation capability to explore different representations and make a prediction correctly in order to lower

the training loss.

To achieve the properties, the controller has two roles in our model. First, it needs to judge if the process should stop. If yes, the output will be generated. Otherwise, it needs to generate a new representation based on previous representation and the a context vector from the global memory. The controller is modeled by a recurrent neural network and controls the process by keeping internal state sequences to track the current progress and history. The controller uses an attention mechanism to fetch information from relevant memory vectors in M, and decides if the model should output the prediction or continue to update the input vector in the next step.

To judge the process should be continued or not, the controller uses a termination module to estimate $P(\texttt{stop}|s_t)$ by a logistical regression module: $\texttt{sigmoid}(W_c s_t + b_c)$, where the weight matrix W_c and bias vector b_c are learned during training. With probability $P(\texttt{stop}|s_t)$, the process will be stopped, and the decoder will be called to generate the output.

With probability $1 - P(\texttt{stop}|s_t)$, the controller needs to generate the next representation $s_{t+1} = \text{RNN}(s_t, x_t)$. The attention vector x_t at t-th step is generated based on the current internal state s_t and the global memory M. Specifically, the attention score $a_{t,i}$ on a memory vector m_i given a state s_t is computed as

$$a_{t,i} \propto \lambda \cos(W_1 m_i, W_2 s_t) \tag{1}$$

where λ is chosen on the development set and is set to 10 in our experiments. The weight matrices W_1 and W_2 are learned during training. The attention vector x_t can be written as $x_t = f_{\texttt{att}}(s_t, M) = \sum_i^{|M|} a_{t,i} m_i$.

Overall Process During inference, the inference process is formally described in Algorithm 1. Given an input [Obama, NATIONALITY], the encoder module converts it to a vector s_1 by concatenating entity/relation embedding lookup. Second, at step t, with probability $P(\texttt{stop}|s_t)$, model outputs the prediction vector o_t. With probability $1 - P(\texttt{stop}|s_t)$, the state s_{t+1} is updated based on the previous state s_t and the vector x_t generated by performing attention over the global memory. We iterate the process till a predefined maximum step T_{max}. Note that the overall framework is generic to different applications by tailoring the encoder/decoder to a target application. An example of a shortest path synthesis task is shown in Appendix.

Training Objectives

In this section, we introduce the training objectives of our model. The controller of the model needs to determine the relation path length. Since the number of steps a model should proceed for each example is unknown in the training data, we optimize the expected reward directly, motivated by the REINFORCE algorithm (Williams 1992).

The expected reward at step t can be obtained as follows. At t-step, given the representation vector s_t, the model generates the output vector $o_t = f_o(s_t)$. We convert the output vector to a probability by the following steps. The probability of selecting a prediction $\hat{y} \in D$ is approximated as $p(\hat{y}|o_t) = \frac{\exp(-\gamma d(o_t, \hat{y}))}{\sum_{y_k \in D} \exp(-\gamma d(o_t, y_k))}$, where $d(o, y) = \|o - y\|_1$

Algorithm 1 Inference Process of IRNs

```
Lookup entity and relation embeddings, h and r. Set s_1 = [h, r]   ▷ Encoder
while True do
    u ~ [0, 1]
    if u > P(stop|s_t) and t < T_max then
        x_t = f_att(s_t, M)                                        ▷ Access Memory
        s_{t+1} = RNN(s_t, x_t), t ← t + 1
    else
        Generate output o_t = f_o(s_t)                             ▷ Decoder
        break                                                      ▷ Stop
    end if
end while
```

is the L_1 distance between the output o and the target entity y, and D is the set of all possible entities. In our experiments, we choose hyperparameters γ and $|D|$ based on the development set and set γ to 5 and sample 20 negative examples in D to speed up training. Assume that ground truth target entity embedding is y^*, the expected reward at time t is defined as:

$$\begin{aligned} J(s_t|\theta) &= \sum_{\hat{y}} R(\hat{y}) \frac{\exp(-\gamma d(o_t, \hat{y}))}{\sum_{\bar{y} \in D} \exp(-\gamma d(o, \bar{y}))} \\ &= \frac{\exp(-\gamma d(o_t, y^*))}{\sum_{\bar{y} \in D} \exp(-\gamma d(o, \bar{y}))}, \end{aligned}$$

where R is the reward function, and we assign the reward to be 1 when we make a correct prediction on the target entity, and 0 otherwise.

Next, we can calculate the reward by summing them over each step. The overall probability of model terminates at time t is $\Pi_{i=1}^{t-1}(1 - v_i)v_t$, where $v_i = P(\texttt{stop}|s_i, \theta)$. Therefore, the overall objective function can be written as

$$J(\theta) = \sum_{t=1}^{T_{\text{max}}} \Pi_{i=1}^{t-1}(1 - v_i) v_t J(s_t|\theta). \tag{2}$$

Then, the parameters can be updated through back-propagation.

Experimental Results

In this section, we evaluate the performance of our model on the benchmark WN18, FB15k, and FB15k-237 datasets for KBC (Bordes et al. 2013; Toutanova et al. 2015). These datasets contain multi-relations between head and tail entities. Given a head entity and a relation, a model produces a ranked list of the entities according to the score of the entity being the tail entity of this triple. To evaluate the ranking, we report **mean rank (MR)**, the mean of rank of the correct entity across the test examples, and **hits@10**, the proportion of correct entities ranked in the top-10 predictions. Lower MR or higher hits@10 indicates a better prediction performance. We follow the evaluation protocol in Bordes et al. (2013) to report filtered results, where negative examples are removed from the dataset. In this case, we avoid some negative examples being valid and ranked above the target triplet.

We use the same hyper-parameters of our model for all datasets. Entity embeddings (which are not shared between

Table 1: The knowledge base completion (link prediction) results on WN18, FB15k, and FB15k-237.

Model	Aux. Info.	WN18		FB15k		FB15k-237	
		Hits@10	MR	Hits@10	MR	Hits@10	MR
TransE (Bordes et al. 2013)	NO	89.2	251	47.1	125	-	-
NTN (Socher et al. 2013)	NO	66.1	-	41.4	-	-	-
TransH (Wang et al. 2014)	NO	86.7	303	64.4	87	-	-
TransR (Lin et al. 2015b)	NO	92.0	225	68.7	77	-	-
CTransR (Lin et al. 2015b)	NO	92.3	218	70.2	75	-	-
KG2E (He et al. 2015)	NO	93.2	348	74.0	59	-	-
TransD (Ji et al. 2015)	NO	92.2	212	77.3	91	-	-
TATEC (García-Durán et al. 2015)	NO	-	-	76.7	58	-	-
DISTMULT (Yang et al. 2015)	NO	94.2	-	57.7	-	41.9	254
STransE (Nguyen et al. 2016)	NO	93.4	**206**	79.7	69	-	-
HOLE (Nickel, Rosasco, and Poggio 2016)	NO	94.9	-	73.9	-	-	-
ComplEx (Trouillon et al. 2016)	NO	94.7	-	84.0	-	-	-
TransG (Xiao, Huang, and Zhu 2016)	NO	94.9	345	88.2	50	-	-
ConvE (Dettmers et al. 2017)	NO	95.5	504	87.3	64	45.8	330
RTransE (García-Durán, Bordes, and Usunier 2015)	Path	-	-	76.2	50	-	-
PTransE (Lin et al. 2015a)	Path	-	-	84.6	58	-	-
NLFeat (Toutanova et al. 2015)	Node + Link Features	94.3	-	87.0	-	-	-
Random Walk (Wei, Zhao, and Liu 2016)	Path	94.8	-	74.7	-	-	-
IRN	NO	**95.3**	249	**92.7**	**38**	**46.4**	**211**

Table 2: The performance of IRNs with different memory sizes and inference steps on FB15k, where $|M|$ and T_{max} represent the number of memory vectors and the maximum inference step, respectively.

$\|M\|$	T_{max}	FB15k	
		Hits@10 (%)	MR
64	1	80.7	55.7
64	2	87.4	49.2
64	5	**92.7**	38.0
64	8	88.8	**32.9**
32	5	90.1	38.7
64	5	**92.7**	38.0
128	5	92.2	36.1
512	5	90.0	35.3
4096	5	88.7	**34.7**

input and output modules) and relation embedding are both 100-dimensions. We use the encoder module and decoder module to encode input entities and relations, and output entities, respectively. There are 64 memory vectors with 200 dimensions each, initialized by random vectors with unit L_2-norm. We use a single-layer GRU with 200 cells as the search controller. We set the maximum inference step T_{max} of the IRN to 5. We randomly initialize all model parameters, and use SGD as the training algorithm with mini-batch size of 64. We set the learning rate to a constant number, 0.01. To prevent the model from learning a trivial solution by increasing entity embeddings norms, we follow Bordes et al. (2013) to enforce the L_2-norm of the entity embeddings as 1. We use hits@10 as the validation metric for the IRN. Following the work (Lin et al. 2015a), we add reverse relations into the training triplet set to increase training data.

Following Nguyen et al. (2016), we divide the results of previous work into two groups. The first group contains the models that directly optimize a scoring function for the triples in a knowledge base without using auxiliary information. The second group of models make uses of auxiliary information from multi-step relations. For example, RTransE (García-Durán, Bordes, and Usunier 2015) and PTransE (Lin et al. 2015a) models are extensions of the TransE (Bordes et al. 2013) model by explicitly exploring multi-step relations in the knowledge base to regularize the trained embeddings. The NLFeat model (Toutanova et al. 2015) is a log-linear model that makes use of simple node and link features.

Table 1 presents the experimental results. According to the table, our model achieves state-of-the-art results without using auxiliary information. Specifically, on FB15k, the MR of our model surpasses all previous results by 12, and our hit@10 outperforms others by 5.7%. To better understand the behavior of IRNs, we report the results of IRNs with different memory sizes and different T_{max} on FB15k in Table 2. We find the performance of IRNs increases significantly if the number of inference step increases. Note that an IRN with T_{max} = 1 is the case that an IRN without the global memory. Interestingly, given T_{max} = 5, IRNs are not sensitive to memory sizes. In particular, larger memory always improves the MR score, but the best hit@10 is obtained by $|M| = 64$ memory vectors. A possible reason is that the best memory size is determined by the complexity of the tasks.

In order to show the inference procedure determined by IRNs, we map the representation s_t back to human-interpretable entity and relation names in the KB. In Table 3, we show a randomly sampled example with its top-3 closest observed inputs [h, R] in terms of L_2-distance, and top-3 answer predictions along with the termination probability at each step. Throughout our observation, the inference procedure is quite different from the relation paths that people designed in the symbolic space (Schoenmackers et al. 2010). The potential reason is that IRNs operate in the vector space.

Table 3: Interpret the state s_t in each step via finding the closest (entity, relation) tuple, and the corresponding the top-3 predictions and termination probability. "Rank" stands for the rank of the target entity and "Term. Prob." stands for termination probability.

Input: [Milwaukee Bucks, /BASKETBALL_ROSTER_POSITION/POSITION]
Target: Forward-center

Step	Term. Prob.	Rank	Top 3 Entity, Relation/Prediction	
1	6.85e-6	5	[Entity, Relation]	1. [**Milwaukee Bucks**, /BASKETBALL_ROSTER_POSITION/POSITION] 2. [Milwaukee Bucks, /SPORTS_TEAM_ROSTER/POSITION] 3. [Arizona Wildcats men's basketball, /BASKETBALL_ROSTER_POSITION/POSITION]
			Prediction	1. Swingman 2. Punt returner 3. Return specialist
2	0.012	4	[Entity, Relation]	1. [(**Phoenix Suns**, /BASKETBALL_ROSTER_POSITION/POSITION] 2. [Minnesota Golden Gophers men's basketball, /BASKETBALL_ROSTER_POSITION/POSITION] 3. [Sacramento Kings, /BASKETBALL_ROSTER_POSITION/POSITION]
			Prediction	1. Swingman 2. Sports commentator 3. Wide receiver
3	0.987	1	[Entity, Relation]	1. [**Phoenix Suns**, /BASKETBALL_ROSTER_POSITION/POSITION] 2. [Minnesota Golden Gophers men's basketball, /BASKETBALL_ROSTER_POSITION/POSITION] 3. [Sacramento Kings, /BASKETBALL_ROSTER_POSITION/POSITION]
			Prediction	1. **Forward-center** 2. Swingman 3. Cabinet of the United States

Instead of connecting triplets that share exactly the same entity as in the symbolic space, IRNs update the representations and connect other triplets in the vector space instead. As we can observe in the examples of Table 3, the model reformulates the representation s_t at each step and gradually increases the ranking score of the correct tail entity with higher termination probability during the inference process. In the last step of Table 3, the closest tuple (Phoenix Suns, /BASKETBALL_ROSTER_POSITION/POSITION) is actually within the training set with a tail entity Forward-center, which is the same as the target entity. Hence, the whole inference process can be thought as the model iteratively reformulates the representation s_t in order to minimize its distance to the target entity in vector space.

To understand what the model has learned in the global memory in the KBC tasks, in Table 4, we visualize the global memory in an IRN trained from FB15k. We compute the average attention scores of each relation type on each memory cell. In the table, we show the top 8 relations, ranked by the average attention scores, of some randomly selected memory cells. These memory cells are activated by certain semantic patterns within the knowledge graph. It suggests that the global memory can capture the connections between triplets. We can still see a few noisy relations in each clustered memory cell, e.g., "bridge-player-teammates/teammate" relation in the "film" memory cell, and "olympic-medal-honor/medalist" in the "disease" memory cell.

We provide some more IRN prediction examples at each step from FB15k as shown in Appendix. In addition to the KBC tasks, we construct a synthetic task, shortest path synthesis, to evaluate the inference capability without any human-designed relation paths as shown in Appendix.

Related Work

Link Prediction and Knowledge Base Completion Given that R is a relation, h is the head entity, and t is the tail entity, most of the embedding models for link prediction focus on finding the scoring function $f_{\mathrm{R}}(\mathrm{h}, \mathrm{t})$ that represents the implausibility of a triple by capturing direct relationship of each triplet (Bordes et al. 2013; Wang et al. 2014; Ji et al. 2015; Nguyen et al. 2016).

Recently, different studies (Gardner et al. 2014; Neelakantan, Roth, and McCallum 2015; Guu, Miller, and Liang 2015; Neelakantan, Roth, and McCallum 2015; Lin et al. 2015a; Das et al. 2016; Wang and Cohen 2016; Toutanova et al. 2016) demonstrate the importance for models to incorporate multi-step relation paths during training. Learning from multi-step relations injects the structured relationships between triples into the model. However, this also poses a technical challenge of considering exponential numbers of multi-step relationships. Prior approaches address this issue by designing path-mining algorithms (Lin et al. 2015a) or considering all possible paths using a dynamic programming algorithm with the restriction of using linear or bi-linear models only (Toutanova et al. 2016). Neelakantan, Roth, and McCallum (2015) and Das et al. (2016) use an RNN to model the multi-step relationships over a set of random walk paths on the observed triplets. Toutanova and Chen (2015) shows the effectiveness of using simple node and link features that encode structured information on FB15k and WN18. In our work, the IRN outperforms prior results and shows that multi-step relation paths can be captured and jointly trained with the model without explicitly designing relation paths on the

Table 4: global memory visualization in an IRN trained on FB15k, where we show the top 8 relations, ranked by the average attention scores, of some randomly selected memory cells. The first row in each column represents the interpreted relation.

"family"	**"person"**	**"film", "award"**
lived-with/participant	person/gender	film-genre/films-in-this-genre
breakup/participant	person/nationality	film/cinematography
marriage/spouse	military-service/military-person	cinematographer/film
vacation-choice/vacationer	government-position-held/office-holder	award-honor/honored-for
support/supported-organization	leadership/role	netflix-title/netflix-genres
marriage/location-of-ceremony	person/ethnicity	director/film
canoodled/participant	person/parents	award-honor/honored-for
dated/participant	person/place-of-birth	bridge-player-teammates/teammate
"disease"	**"sports"**	**"tv program"**
disease-cause/diseases	sports-team-roster/team	tv-producer-term/program
crime-victim/crime-type	basketball-roster-position/player	tv-producer-term/producer-type
notable-person-with-medical-condition/condition	basketball-roster-position/player	tv-guest-role/episodes-appeared-in
cause-of-death/parent-cause-of-death	baseball-player/position-s	tv-program/languages
disease/notable-people-with-this-condition	appointment/appointed-by	tv-guest-role/actor
olympic-medal-honor/medalist	batting-statistics/team	tv-program/spin-offs
disease/includes-diseases	basketball-player-stats/team	award-honor/honored-for
disease/symptoms	person/profession	tv-program/country-of-origin

observed triplets.

Studies such as (Riedel et al. 2013) show that incorporating textual information can further improve the KBC tasks. It would be interesting to incorporate the information outside the knowledge bases in our model in the future.

Neural Frameworks Sequence-to-sequence models (Sutskever, Vinyals, and Le 2014; Cho et al. 2014) have shown to be successful in many applications such as machine translation and conversation modeling (Sordoni et al. 2015). While sequence-to-sequence models are powerful, recent work has shown the necessity of incorporating an external memory to perform inference in simple algorithmic tasks (Graves, Wayne, and Danihelka 2014; Graves et al. 2016).

Compared IRNs to Memory Networks (MemNN) (Weston, Chopra, and Bordes 2014; Miller et al. 2016) and Neural Turing Machines (NTM) (Graves, Wayne, and Danihelka 2014; Graves et al. 2016), the biggest difference between our model and the existing frameworks is the controller and the use of the global memory. We follow Shen et al. (2017) for using a controller module to dynamically perform a multi-step inference depending on the complexity of the input. MemNN and NTM explicitly store inputs (such as graph definition, supporting facts) in the memory. In contrast, in IRNs, we do not explicitly store all the observed inputs in the global memory. Instead, we directly operate on the global memory, which stores the structured relationships of a knowledge graph implicitly. During training, we randomly initialize the memory and update the memory jointly with the controller with respect to task-specific objectives via back-propagation, instead of explicitly defining memory write operations as in NTM.

Conclusion

In this paper, we propose Implicit ReasoNets (IRNs) that learns to traverse knowledge graph in vector space by the design of global memory and controller. Without using auxiliary information, the global memory stores large-scale structured relationships of knowledge graph implicitly, and the controller updates its representations in relation paths and adaptively changes relation path length depending on the complexity of the input. We demonstrate and analyze the multi-step inference capability of IRNs in the knowledge base completion tasks. Our model, without using any auxiliary knowledge base information, achieves state-of-the-art results on the WN18, FB15k, and FB15k-237 benchmarks.

For future work, we aim to further extend IRNs in two ways. First, inspired from Ribeiro, Singh, and Guestrin (2016), we would like to develop techniques to exploit ways to generate human understandable reasoning interpretation from the global memory. Second, we plan to apply IRNs to infer the relationships in unstructured data such as natural language. For example, given a natural language query such as "are rabbits animals?", the model can infer a natural language answer implicitly in the global memory without performing inference directly on top of huge amounts of observed sentences such as "all mammals are animals" and "rabbits are animals". We believe that the ability to perform inference implicitly is crucial for modeling large-scale structured relationships.

Acknowledgments

We thank Scott Wen-Tau Yih, Kristina Toutanova, Jian Tang, Greg Yang, Adith Swaminathan, Xiaodong He, and Zachary Lipton for their thoughtful feedback and discussions.

References

[2013] Berant, J.; Chou, A.; Frostig, R.; and Liang, P. 2013. Semantic parsing on Freebase from question-answer pairs. In *EMNLP*.

[2008] Bollacker, K.; Evans, C.; Paritosh, P.; Sturge, T.; and Taylor, J. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *SIGMOD*, 1247–1250.

[2013] Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston,

J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. In *NIPS*, 2787–2795.

[2014] Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[2016] Das, R.; Neelakantan, A.; Belanger, D.; and McCallum, A. 2016. Chains of reasoning over entities, relations, and text using recurrent neural networks. In *EACL*.

[2017] Dettmers, T.; Minervini, P.; Stenetorp, P.; and Riedel, S. 2017. Convolutional 2d knowledge graph embeddings. *CoRR* abs/1707.01476.

[1998] Fellbaum, C. 1998. *WordNet: An Electronic Lexical Database*. MIT Press.

[2015] García-Durán, A.; Bordes, A.; Usunier, N.; and Grandvalet, Y. 2015. Combining two and three-way embeddings models for link prediction in knowledge bases. *CoRR* abs/1506.00999.

[2015] García-Durán, A.; Bordes, A.; and Usunier, N. 2015. Composing relationships with translations. In *EMNLP*, 286–290.

[2014] Gardner, M.; Talukdar, P. P.; Krishnamurthy, J.; and Mitchell, T. 2014. Incorporating vector space similarity in random walk inference over knowledge bases. In *EMNLP*.

[2016] Graves, A.; Wayne, G.; Reynolds, M.; Harley, T.; Danihelka, I.; Grabska-Barwińska, A.; Colmenarejo, S. G.; Grefenstette, E.; Ramalho, T.; Agapiou, J.; et al. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature*.

[2014] Graves, A.; Wayne, G.; and Danihelka, I. 2014. Neural Turing machines. *arXiv preprint arXiv:1410.5401*.

[2015] Guu, K.; Miller, J.; and Liang, P. 2015. Traversing knowledge graphs in vector space. *arXiv preprint arXiv:1506.01094*.

[2015] He, S.; Liu, K.; Ji, G.; and Zhao, J. 2015. Learning to represent knowledge graphs with gaussian embedding. In *CIKM*, 623–632.

[2015] Ji, G.; He, S.; Xu, L.; Liu, K.; and Zhao, J. 2015. Knowledge graph embedding via dynamic mapping matrix. In *ACL*.

[2015a] Lin, Y.; Liu, Z.; Luan, H.; Sun, M.; Rao, S.; and Liu, S. 2015a. Modeling relation paths for representation learning of knowledge bases. In *EMNLP*.

[2015b] Lin, Y.; Liu, Z.; Sun, M.; Liu, Y.; and Zhu, X. 2015b. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, 2181–2187.

[2016] Miller, A.; Fisch, A.; Dodge, J.; Karimi, A.-H.; Bordes, A.; and Weston, J. 2016. Key-value memory networks for directly reading documents. In *EMNLP*.

[2009] Mintz, M.; Bills, S.; Snow, R.; and Jurafsky, D. 2009. Distant supervision for relation extraction without labeled data. In *IJCNLP*, 1003–1011.

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

[2016] Nguyen, D. Q.; Sirts, K.; Qu, L.; and Johnson, M. 2016. STransE: a novel embedding model of entities and relationships in knowledge bases. In *NAACL*, 460–466.

[2016] Nickel, M.; Rosasco, L.; and Poggio, T. 2016. Holographic embeddings of knowledge graphs. In *AAAI*, 1955–1961.

[2011] Nickel, M.; Tresp, V.; and Kriegel, H.-P. 2011. A three-way model for collective learning on multi-relational data. In *ICML*, 809–816.

[2016] Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *KDD*.

[2013] Riedel, S.; Yao, L.; McCallum, A.; and Marlin, B. M. 2013. Relation extraction with matrix factorization and universal schemas. In *HLT-NAACL*, 74–84.

[2010] Schoenmackers, S.; Etzioni, O.; Weld, D. S.; and Davis, J. 2010. Learning first-order Horn clauses from web text. In *EMNLP*, 1088–1098.

[2017] Shen, Y.; Huang, P.; Gao, J.; and Chen, W. 2017. ReasoNet: Learning to stop reading in machine comprehension. In *KDD*.

[2013] Socher, R.; Chen, D.; Manning, C. D.; and Ng, A. Y. 2013. Reasoning with neural tensor networks for knowledge base completion. In *NIPS*.

[2015] Sordoni, A.; Galley, M.; Auli, M.; Brockett, C.; Ji, Y.; Mitchell, M.; Nie, J.-Y.; Gao, J.; and Dolan, B. 2015. A neural network approach to context-sensitive generation of conversational responses. *arXiv preprint arXiv:1506.06714*.

[2007] Suchanek, F. M.; Kasneci, G.; and Weikum, G. 2007. Yago: A Core of Semantic Knowledge. In *WWW*.

[2014] Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *NIPS*.

[2015] Toutanova, K., and Chen, D. 2015. Observed versus latent features for knowledge base and text inference. In *CVSC*.

[2015] Toutanova, K.; Chen, D.; Pantel, P.; Poon, H.; Choudhury, P.; and Gamon, M. 2015. Representing text for joint embedding of text and knowledge bases. In *EMNLP*.

[2016] Toutanova, K.; Lin, X. V.; tau Yih, S. W.; Poon, H.; and Quirk., C. 2016. Compositional learning of embeddings for relation paths in knowledge bases and text. In *ACL*.

[2016] Trouillon, T.; Welbl, J.; Riedel, S.; Gaussier, É.; and Bouchard, G. 2016. Complex embeddings for simple link prediction. In *ICML*, 2071–2080.

[2016] Wang, W. Y., and Cohen, W. W. 2016. Learning first-order logic embeddings via matrix factorization. In *IJCAI*.

[2014] Wang, Z.; Zhang, J.; Feng, J.; and Chen, Z. 2014. Knowledge graph embedding by translating on hyperplanes. In *AAAI*.

[2016] Wei, Z.; Zhao, J.; and Liu, K. 2016. Mining inference formulas by goal-directed random walks. In *EMNLP*.

[2014] West, R.; Gabrilovich, E.; Murphy, K.; Sun, S.; Gupta, R.; and Lin, D. 2014. Knowledge base completion via search-based question answering. In *WWW*, 515–526.

[2014] Weston, J.; Chopra, S.; and Bordes, A. 2014. Memory networks. *arXiv preprint arXiv:1410.3916*.

[1992] Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8:229–256.

[2016] Xiao, H.; Huang, M.; and Zhu, X. 2016. TransG : A generative model for knowledge graph embedding. In *ACL*.

[2015] Yang, B.; Yih, W.; He, X.; Gao, J.; and Deng, L. 2015. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR*.

[2015] Yih, W.-t.; Chang, M.-W.; He, X.; and Gao, J. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *ACL*.

Inference Steps in KBC

To analyze the behavior of IRNs in each relation path step, we further pick some examples for the tail entity prediction in Table 5. Interestingly, we observe that the model can gradually increase the ranking score of the correct tail entity during the inference process.

Analysis: Applying IRNs to a Shortest Path Synthesis Task

To further understand the inference procedure of IRNs, we construct a synthetic task, shortest path synthesis, to evaluate the inference capability without using any human-designed relation paths. The motivations of applying our model to this task are as follows. First, we want to evaluate IRNs on another task requiring multi-step relation paths. Second, we select the *sequence generation* task so that we are able to analyze the inference capability of IRNs in details.

In the shortest path synthesis task, as illustrated in Figure 4, a training instance consists of a start node and an end node (e.g., $215 \rightsquigarrow 493$) of an underlying weighted directed graph that is unknown to models. The output of each instance is the shortest path between the given start and end nodes of the underlying graph (e.g., $215 \rightarrow 101 \rightarrow 493$). Specifically, models can only observe the start-end node pairs as input and their shortest path as output. The whole graph is unknown to the models and the edge weights are not revealed in the training data. At test time, a path sequence is considered correct if it connects the start node and the end node of the underlying graph, and the cost of the predicted path is the same as the optimal path.

We construct the underlying graph as follows: on a three-dimensional unit-sphere, we randomly generate a set of nodes. For each node, we connect its K-nearest neighbors and use the euclidean distance between two nodes to construct a graph. We randomly sample two nodes and compute its shortest path if it is connected between these two nodes. Given the fact that all the sub-paths within a shortest path are shortest paths, we incrementally create the dataset and remove the instances which are a sub-path of previously selected paths or are super-set of previously selected paths. In this case, all the shortest paths can not be answered through directly copying from another instance. In addition, all the weights in the graph are hidden and not shown in the training data, which increases the difficulty of the tasks. We set $k = 50$ as a default value.

Note that the task is very difficult and *cannot* be solved by dynamic programming algorithms since the weights on the edges are not revealed to the algorithms or the models. To recover some of the shortest paths at the test time, the model needs to infer the correct path from the observed instances. For example, assume that we observe two instances in the training data, "$A \rightsquigarrow D$: $A \rightarrow B \rightarrow G \rightarrow D$" and "$B \rightsquigarrow E$: $B \rightarrow C \rightarrow E$". In order to answer the shortest path between A and E, the model needs to infer that "$A \rightarrow B \rightarrow C \rightarrow E$" is a possible path between A and E. If there are multiple possible paths, the model has to decide which one is the shortest one using statistical information.

In the experiments, we construct a graph with 500 nodes and we randomly assign two nodes to form an edge. We split 20,000 instances for training, 10,000 instances for validation, and 10,000 instances for testing. We create the training and testing instances carefully so that the model needs to perform inference to recover the correct path. We describe the details of the graph and data construction parts in the appendix section. A sub-graph of the data is shown in Figure 4.

For the settings of the IRN, we switch the output module to a GRU decoder for a sequence generation task. We assign reward $r_T = 1$ if all the prediction symbols are correct and 0 otherwise. We use a 64-dimensional embedding vector for input symbols, a GRU controller with 128 cells, and a GRU decoder with 128 cells. We set the maximum inference step T_{max} to 5.

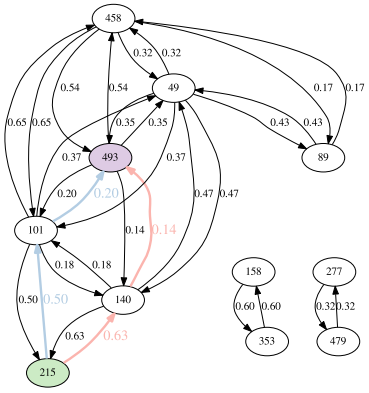
We compare the IRN with two baseline approaches: dynamic programming without edge-weight information and a standard sequence-to-sequence model (Sutskever, Vinyals, and Le 2014) using a similar parameter size to our model. Without knowing the edge weights, dynamic programming only recovers 589 correct paths at test time. The sequence-to-sequence model recovers 904 correct paths. The IRN outperforms both baselines, recovering 1,319 paths. Furthermore, 76.9% of the predicted paths from IRN are *valid* paths, where a path is valid if the path connects the start and end node nodes of the underlying graph. In contrast, only 69.1% of the predicted paths from the sequence-to-sequence model are valid.

To further understand the inference process of the IRN, Figure 4 shows the inference process of a test instance. Interestingly, to make the correct prediction on this instance, the model has to perform a fairly complicated inference.[1] We observe that the model cannot find a connected path in the first three steps. Finally, the model finds a valid path at the forth step and predict the correct shortest path sequence at the fifth step.

[1] In the example, to find the right path, the model needs to search over observed instances "$215 \rightsquigarrow 448$: $215 \rightarrow 101 \rightarrow 448$" and "$76 \rightsquigarrow 493$: $76 \rightarrow 308 \rightarrow 101 \rightarrow 493$", and to figure out the distance of "$140 \rightarrow 493$" is longer than "$101 \rightarrow 493$" (there are four shortest paths between $101 \rightarrow 493$ and three shortest paths between $140 \rightarrow 493$ in the training set).

Table 5: An inference example of FB15k dataset. Given a head entity and a relation, the predictions of IRN in different steps associated with the corresponding termination probabilities.

Input: [Dean Koontz, /PEOPLE/PERSON/PROFESSION]					
Target: Film Producer					
Step	Termination Prob.	Answer Rank	Predict top-3 entities		
1	0.018	9	Author	TV. Director	Songwriter
2	0.052	7	Actor	Singer	Songwriter
3	0.095	4	Actor	Singer	Songwriter
4	0.132	4	Actor	Singer	Songwriter
5	0.702	3	Actor	Singer	**Film Producer**
Input: (War and Peace, /FILM/FILM/PRODUCED_BY)					
Target: Carlo Ponti					
Step	Termination Prob.	Answer Rank	Predict top-3 entities		
1	0.001	13	Scott Rudin	Stephen Woolley	Hal B. Wallis
2	5.8E-13	7	Billy Wilder	William Wyler	Elia Kazan
3	0.997	1	**Carlo Ponti**	King Vidor	Hal B. Wallis



Step	Termination Probability	Distance	Predictions
1	0.001	N/A	$215 \rightarrow 158 \rightarrow 89 \rightarrow 458 \rightarrow 493$
2	$\sim$0	N/A	$215 \rightarrow 479 \rightarrow 277 \rightarrow 353 \rightarrow 493$
3	$\sim$0	N/A	$215 \rightarrow 49 \rightarrow 493$
4	$\sim$0	0.77	$215 \rightarrow 140 \rightarrow 493$
5	0.999	0.70	$215 \rightarrow 101 \rightarrow 493$

Figure 4: An example of the shortest path synthesis dataset, given an input "$215 \rightsquigarrow 493$" (Answer: $215 \rightarrow 101 \rightarrow 493$). Note that we only show the nodes that are related to this example here. The corresponding termination probability and prediction results are shown in the table. The model terminates at step 5.