Modeling Large-Scale Structured Relationships with Shared Memory for Knowledge Base Completion

Yelong Shen* and **Po-Sen Huang*** and **Ming-Wei Chang** and **Jianfeng Gao**
Microsoft Research, Redmond, WA, USA
{yeshen,pshuang,minchang,jfgao}@microsoft.com

Abstract

Recent studies on knowledge base completion, the task of recovering missing relationships based on recorded relations, demonstrate the importance of learning embeddings from multi-step relations. However, due to the size of knowledge bases, learning multi-step relations directly on top of observed triplets could be costly. Hence, a manually designed procedure is often used when training the models. In this paper, we propose Implicit ReasoNets (`IRNs`), which is designed to perform multi-step inference *implicitly* through a controller and shared memory. Without a human-designed inference procedure, `IRNs` use training data to *learn* to perform multi-step inference in an embedding neural space through the shared memory and controller. While the inference procedure does not explicitly operate on top of observed triplets, our proposed model outperforms all previous approaches on the popular FB15k benchmark by more than 5.7%.

1 Introduction

Knowledge bases such as WordNet (Fellbaum, 1998), Freebase (Bollacker et al., 2008), or Yago (Suchanek et al., 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 entities based on the observed entities and relations. Thus, the knowledge base completion (KBC) task has emerged an important open research problem (Nickel et al., 2011).

Neural-network based methods have been very popular for solving the KBC task. Following Bordes et al. (2013), one of the most popular approaches for KBC is to learn vector-space representations of entities and relations during training, and then apply linear or bi-linear operations to infer the missing relations at test time. However, several recent papers demonstrate limitations of prior approaches relying upon vector-space models alone (Guu et al., 2015; Toutanova et al., 2016; Lin et al., 2015a). By themselves, there is no straightforward way to capture the structured relationships between multiple triples adequately. For example, assume that we want to fill in the missing relation for the triple (`Obama`, NATIONALITY, ?), a multi-step search procedure might be needed to discover the evidence in the observed triples such as (`Obama`, BORNIN, `Hawaii`) and (`Hawaii`, PARTOF, `U.S.A`). To address this issue, Guu et al. (2015); Toutanova et al. (2016); Neelakantan et al. (2015); Das et al. (2016); Lin et al. (2015a) propose different approaches of injecting structured information based on the human-designed inference procedure (e.g., random walk) that directly operates on the observed triplets. Unfortunately, due to the size of knowledge bases, these newly proposed approaches suffer from some limitations, as most paths are not informative for inferring missing relations, and it is prohibitive to consider all possible paths during the training time.

In this paper, we propose Implicit ReasoNets (`IRNs`) that take a different approach from prior work on KBC by addressing the challenges of performing multi-step inference through the design of

*These authors contributed equally.

Proceedings of the 2nd Workshop on Representation Learning for NLP, pages 57–68,
Vancouver, Canada, August 3, 2017.

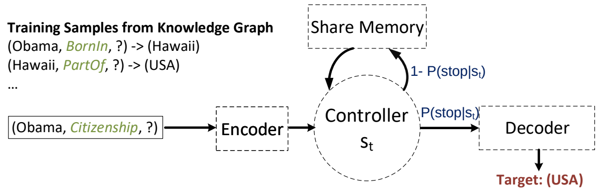


Figure 1: An overview of the IRN for KBC tasks.

controller and *shared memory*. We design a shared memory component to store KB information implicitly. That is, the model needs to determine what information it should store. Moreover, instead of explicitly manipulating the observed triples based on the human-designed inference procedure, the proposed model learns the multi-step inference procedure implicitly, i.e., without human intervention. Specifically, our model makes the prediction several times while forming different intermediate representations along the way. The controller determines how many steps the model should proceed given an input. At each step, a new representation is formed by taking the current representation and a context vector generated by accessing the shared memory. The detailed process is introduced in Section 3.3 and an overview of the model is shown in Figure 1.

The main contributions of our paper are as follows:

- We propose Implicit ReasoNets (IRNs), which use a shared memory guided by a controller to model multi-step structured relationships implicitly.
- We evaluate IRNs and demonstrate that our proposed model achieves the state-of-the-art results on the popular FB15k benchmark, surpassing prior approaches by more than 5.7%.
- Our analysis shows that the multi-step inference is crucial to the performance of our model.

2 Knowledge Base Completion Task

The goal of Knowledge Base Completion (KBC) tasks is to predict a head or a tail entity given the relation type and the other entity, i.e. predicting the head entity h given a triplet $(?, \mathtt{R}, \mathtt{t})$ with relation R and tail entity t, or predicting the tail entity t given a triplet $(\mathtt{h}, \mathtt{R}, ?)$ with head entity h and relation R, where ? denotes the missing 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.

Recent approaches (Bordes et al., 2013; Socher et al., 2013) achieve better generalization by operating on embedding representations, where the vector similarity can be regarded as semantic similarity.

In the evaluation, models compute the similarity between the output prediction and all entities. Mean rank and precision of the target entity are used as metrics for evaluation.

3 Proposed Model

Our proposed model uses the same setup as in the embedding type of approaches (Bordes et al., 2013; Socher et al., 2013), i.e., the model first takes a triplet with a missing entity, $(\mathtt{h}, \mathtt{R}, ?)$, as input, then maps the input into the neural space through embeddings, and finally outputs a prediction vector of the missing entity. Given that our model is a neural model, we use the *encoder* module to transform the input triplet $(\mathtt{h}, \mathtt{R}, ?)$ to a continuous representation. For generating the prediction results, the *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 tasks between symbolic space and neural space.

The main differences between our model and previous proposed models is that we make the prediction several times while forming *multiple* intermediate continuous representations along the way. Given an intermediate representation, the *controller* judges if the representation encodes enough information for us to produce the output prediction or not. If the controller agrees, we produce the current prediction as our final output. Otherwise, the controller generates a new continuous representation by taking current representation and a context vector generated by accessing the *shared memory*. Then the new presentation will be fed into

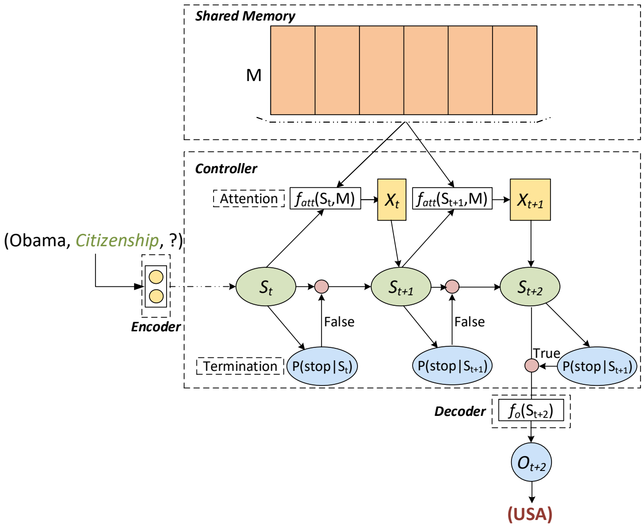


Figure 2: A running example of the IRN architecture. Given the input (Obama, CITIZENSHIP, ?), the model iteratively reformulates the input vector via the current input vector and the attention vector over the shared memory, and determines to stop when an answer is found.

the controller, and 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.

3.1 Inference

Encoder/Decoder Given an input $(\mathtt{h}, \mathtt{R}, ?)$, the encoder module retrieves the entity h and relation 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 $f_o(s_t) = \mathtt{tanh}(W_o s_t + b_o)$ 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.

Shared Memory The shared memory is denoted as $M = \{m_i\}_{i=1}^{|M|}$, which consists of a list of vectors. During training, the shared memory, which is shared across all training instances, is first randomly initialized and then is jointed learned with the controller on training data.

Controller The controller has two roles in our model. First, it needs to judge if the process should be stopped. If yes, the output will be generated. Otherwise, it needs to generate a new representation based on previous representation and the context vector generated from shared memory. The controller is a recurrent neural network and controls the process by keeping internal state sequences to track the current search process 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 model estimates $P(\mathtt{stop}|s_t)$ by a logistical regression module: $\mathtt{sigmoid}(W_c s_t + b_c)$, where the weight matrix W_c and bias vector b_c are learned during training. With probability $P(\mathtt{stop}|s_t)$, the process will be stopped, and the decoder will be called to generate the output.

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

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

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

Overall Process The inference process is formally described in Algorithm 1. Given 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 $P(\mathtt{stop}|s_t)$, model outputs the prediction vector o_i. With probability $1 - P(\mathtt{stop}|s_t)$, the state s_{i+1} is updated based on the previous state s_i and the vector x_t generated by performing attention over the shared memory.

We iterate the process till a predefined maximum step $T_{\max}$. At test time, the model outputs a prediction o_j where the step j has the maximum termination probability. Note that the overall framework is generic to different applications by tailoring the encoder/decoder to a target application. An example of shortest path synthesis task is shown in Appendix B.

Algorithm 1 Inference Process of IRNs

Lookup entity and relation embeddings, $\mathbf{h}$ and $\mathbf{r}$.
Set $s_1 = [\mathbf{h}, \mathbf{r}]$ ▷ Encoder
while True **do**
  $u \sim [0, 1]$
  if $u > P(\texttt{stop}|s_t)$ **then**
    $x_t = f_{\texttt{att}}(s_t, M)$ ▷ Access Memory
    $s_{t+1} = \text{RNN}(s_t, x_t), t \leftarrow t + 1$
  else
    Generate output $o_t = f_o(s_t)$ ▷ Decoder
    break ▷ Stop
  end if
end while

3.2 Training Objectives

In this section, we introduce the training objectives to train our model. While our process is stochastic, the model mainly needs to decide the number of steps for generating the intermediate representations for each example. Since the number of steps the model should take 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 score o_t as $f_o(s_t)$. We convert the output score 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 $d(o, y) = \|o - y\|_1$ 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 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:

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

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 terminated 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

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

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

3.3 Motivating Examples

We now describe the motivating examples to explain the design of shared memory that implicitly stores KB information and the design of the controller that implicitly learns the inference procedure.

Shared Memory Suppose, in a KBC task, the input is (Obama, NATIONALITY, ?) and the model is required to answer the missing entity (answer: U.S.A). Our model can learn to utilize and store information in the shared memory through the controller. When a new information from a new instance is received (e.g., (Obama, BORNIN, Hawaii)), the model first uses its controller to find relevant information (e.g., (Hawaii, PARTOF, U.S.A)). If the relevant information is not found, the model learns to store the information to memory vectors by gradient update in order to answer the missing entity correctly. Due to the limited size of the shared memory, the model cannot store all new information explicitly. Thus, the model needs to learn to utilize the shared memory efficiently to lower the training loss. If a related information from a new instance is received, the model learns to do inference by utilizing the controller to go over existing memory vectors iteratively. In this way, the model could learn to do inference and correlate training instances via memory cells without explicitly storing new information.

Controller The design of the controller allows the model to iteratively reformulate its representation through incorporating context information retrieved from the shared memory. Without explicitly providing human-designed inference procedure, during the iterative progress, the controller needs to explore the multi-step inference procedure 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.

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

Model	Additional Information	WN18		FB15k	
		Hits@10 (%)	MR	Hits@10 (%)	MR
SE (Bordes et al., 2011)	NO	80.5	985	39.8	162
Unstructured (Bordes et al., 2014)	NO	38.2	304	6.3	979
TransE (Bordes et al., 2013)	NO	89.2	251	47.1	125
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
NTN (Socher et al., 2013)	NO	66.1	-	41.4	-
DISTMULT (Yang et al., 2014)	NO	94.2	-	57.7	-
STransE (Nguyen et al., 2016)	NO	94.7 (93)	244 (**206**)	79.7	69
RTransE (García-Durán et al., 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 et al., 2016)	Path	94.8	-	74.7	-
`IRN`	NO	**95.3**	249	**92.7**	**38**

Table 2: The performance of `IRN`s 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**

4 Experimental Results

In this section, we evaluate the performance of our model on the benchmark FB15k and WN18 datasets for KBC (Bordes et al., 2013). These datasets contain multi-relations between head and tail entities. Given a head entity and a relation, the 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 N 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 both FB15k and WN18 datasets. Entity embeddings (which are not shared between 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 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 the 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 extra information. The second group of models make uses of additional information from multi-step relations. For example, RTransE (García-Durán et al., 2015) and PTransE

Table 3: Hits@10 (%) in the relation category on FB15k. (`M` stands for `Many`)

Model	Predicting head h				Predicting tail t			
	`1-1`	`1-M`	`M-1`	`M-M`	`1-1`	`1-M`	`M-1`	`M-M`
SE (Bordes et al., 2011)	35.6	62.6	17.2	37.5	34.9	14.6	68.3	41.3
Unstructured (Bordes et al., 2014)	34.5	2.5	6.1	6.6	34.3	4.2	1.9	6.6
TransE (Bordes et al., 2013)	43.7	65.7	18.2	47.2	43.7	19.7	66.7	50.0
TransH (Wang et al., 2014)	66.8	87.6	28.7	64.5	65.5	39.8	83.3	67.2
TransR (Lin et al., 2015b)	78.8	89.2	34.1	69.2	79.2	37.4	90.4	72.1
CTransR (Lin et al., 2015b)	81.5	89.0	34.7	71.2	80.8	38.6	90.1	73.8
KG2E (He et al., 2015)	**92.3**	94.6	66.0	69.6	**92.6**	67.9	94.4	73.4
TransD (Ji et al., 2015)	86.1	95.5	39.8	78.5	85.4	50.6	94.4	81.2
TATEC (García-Durán et al., 2015)	79.3	93.2	42.3	77.2	78.5	51.5	92.7	80.7
STransE (Nguyen et al., 2016)	82.8	94.2	50.4	80.1	82.4	56.9	93.4	83.1
PTransE (Lin et al., 2015a)	91.0	92.8	60.9	83.8	91.2	74.0	88.9	86.4
`IRN`	87.2	**96.1**	**84.8**	**92.9**	86.9	**90.5**	**95.3**	**94.1**

(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 significantly outperforms previous baselines, regardless of whether previous approaches use additional information or not. Specifically, on FB15k, the MR of our model surpasses all previous results by 12, and our hit@10 outperforms others by 5.7%. On WN18, the `IRN` obtains the highest hit@10 while maintaining similar MR results compared to previous work.*

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 shared 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.

We evaluate hits@10 results on FB15k with respect to the relation categories. Following the evaluation in Bordes et al. (2013), we categorize the relations according to the cardinalities of their associated head and tail entities in four types: `1-1`, `1-Many`, `Many-1`, and `Many-Many`. A given relation is `1-1` if a head entity can appear with at most one tail entity, `1-Many` if a head entity can appear with many tail entities, `Many-1` if multiple heads can appear with the same tail entity, and `Many-Many` if multiple head entities can appear with multiple tail entities. The detailed results are shown in Table 3. The `IRN` significantly improves the hits@10 results in the `Many-1` category on predicting the head entity (18.8%), the `1-Many` category on predicting the tail entity (16.5%), and the `Many-Many` category (over 8% in average).

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 4, we show a randomly sampled example with its top-3 closest triplets (`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 traditional inference chain that people designed in the symbolic space (Schoenmackers et al., 2010). The potential reason is that `IRNs` operate in the neural space. Instead of connecting triplets that share exactly the same entity as in the symbolic space, `IRNs` update the representations and connects other triplets in the semantic space instead. As we can observe in the examples of Table 4, 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 4, 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 tar-

*Nguyen et al. (2016) reported two results on WN18, where the first one is obtained by choosing to optimize hits@10 on the validation set, and second one is obtained by choosing to optimize MR on the validation set. We list both of them in Table 1.

Table 4: 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`

get entity. Hence, the whole inference process can be thought as the model iteratively reformulates the representations in order to minimize its distance to the target entity in neural space.

To understand what the model has learned in the shared memory in the KBC tasks, in Table 5, we visualize the shared 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 memory cells. These memory cells are activated by certain semantic patterns within the knowledge graph. It suggests that the shared memory can efficiently capture the relationships implicitly. 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 A. In addition to the KBC tasks, we construct a synthetic task, shortest path synthesis, to evaluate the inference capability over a shared memory as shown in the Appendix B.

5 Related Work

Link Prediction and Knowledge Base Completion Given that $\mathtt{R}$ is a relation, $\mathtt{h}$ is the head entity, and $\mathtt{t}$ is the tail entity, most of the embedding models for link prediction focus on finding the scoring function $f_r(\mathtt{h}, \mathtt{t})$ that represents the implausibility of a triple. (Bordes et al., 2011, 2014, 2013; Wang et al., 2014; Ji et al., 2015; Nguyen et al., 2016). In many studies, the scoring function $f_r(\mathtt{h}, \mathtt{t})$ is linear or bi-linear. For example, in TransE (Bordes et al., 2013), the function is implemented as $f_r(\mathtt{h}, \mathtt{t}) = \|\mathbf{h} + \mathbf{r} - \mathbf{t}\|$, where $\mathbf{h}$, $\mathbf{r}$ and $\mathbf{t}$ are the corresponding vector representations.

Recently, different studies (Guu et al., 2015; Lin et al., 2015a; Neelakantan et al., 2015; Das et al., 2016; Toutanova et al., 2016) demonstrate the importance for models to also learn from multi-step relations. 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 et al.

Table 5: Shared memory visualization in an IRN trained on FB15k, where we show the top 8 relations, ranked by the average attention scores, of some 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

(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 similar information can be captured by the model without explicitly designing inference procedures on the observed triplets. Our model can be regarded as a recursive function that iteratively update the representation in such a way that its distance to the target entity in the neural space is minimized, i.e., $\|f_{\texttt{IRN}}(\mathbf{h},\mathbf{r})-\mathbf{t}\|$.

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 et al., 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 et al., 2014, 2016).

Compared IRNs to Memory Networks (MemNN) (Weston et al., 2014; Sukhbaatar et al., 2015; Miller et al., 2016) and Neural Turing Machines (NTM) (Graves et al., 2014, 2016), the biggest difference between our model and the existing frameworks is the controller and the use of the shared 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 instance. 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 shared memory. Instead, we directly operate on the shared memory, which modeling the structured relationships 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.

6 Conclusion

In this paper, we propose Implicit ReasoNets (IRNs), which perform inference over a shared memory that stores large-scale structured relationships implicitly. The inference process is guided by a controller to access the memory that is shared across instances. We demonstrate and analyze the multi-step inference capability of IRNs in the knowledge base completion tasks. Our model, without using any explicit knowledge base information in the inference procedure, outperforms all prior approaches on the popular FB15k benchmark by more than 5.7%.

For future work, we aim to further extend IRNs in two ways. First, inspired from Ribeiro et al. (2016), we would like to develop techniques to exploit ways to generate human understandable reasoning interpretation from the shared 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 shared 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.

References

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of EMNLP*.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of SIGMOD-08*. pages 1247–1250.

Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. 2014. A semantic matching energy function for learning with multi-relational data. *Machine Learning* 94(2):233–259.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*. pages 2787–2795.

Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. 2011. Learning structured embeddings of knowledge bases. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*. pages 301–306.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* .

Rajarshi Das, Arvind Neelakantan, David Belanger, and Andrew McCallum. 2016. Chains of reasoning over entities, relations, and text using recurrent neural networks. *arXiv preprint arXiv:1607.01426* .

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

Alberto García-Durán, Antoine Bordes, and Nicolas Usunier. 2015. Composing relationships with translations. In *EMNLP*. pages 286–290.

Alberto García-Durán, Antoine Bordes, Nicolas Usunier, and Yves Grandvalet. 2015. Combining two and three-way embeddings models for link prediction in knowledge bases. *CoRR* abs/1506.00999.

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

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature* .

Kelvin Guu, John Miller, and Percy Liang. 2015. Traversing knowledge graphs in vector space. *arXiv preprint arXiv:1506.01094* .

Shizhu He, Kang Liu, Guoliang Ji, and Jun Zhao. 2015. Learning to represent knowledge graphs with gaussian embedding. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. pages 623–632.

Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Knowledge graph embedding via dynamic mapping matrix. In *ACL*. pages 687–696.

Yankai Lin, Zhiyuan Liu, Huanbo Luan, Maosong Sun, Siwei Rao, and Song Liu. 2015a. Modeling relation paths for representation learning of knowledge bases. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*.

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015b. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI'15, pages 2181–2187.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. In *EMNLP*.

Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *Proceedings of ACL-IJCNLP-09*. pages 1003–1011.

Arvind Neelakantan, Benjamin Roth, and Andrew McCallum. 2015. Compositional vector space models for knowledge base completion. *arXiv preprint arXiv:1504.06662* .

Dat Quoc Nguyen, Kairit Sirts, Lizhen Qu, and Mark Johnson. 2016. STransE: a novel embedding model of entities and relationships in knowledge bases. In *NAACL*. pages 460–466.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on Machine Learning*. pages 809–816.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. “Why Should I Trust You?”: Explaining the Predictions of Any Classifier. In *KDD*.

Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M. Marlin. 2013. Relation extraction with matrix factorization and universal schemas. In *HLT-NAACL*. pages 74–84.

Stefan Schoenmackers, Oren Etzioni, Daniel S Weld, and Jesse Davis. 2010. Learning first-order horn clauses from web text. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 1088–1098.

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

Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Y. Ng. 2013. Reasoning With Neural Tensor Networks For Knowledge Base Completion. In *Advances in Neural Information Processing Systems*.

Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. 2015. A neural network approach to context-sensitive generation of conversational responses. *arXiv preprint arXiv:1506.06714* .

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

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *Advances in Neural Information Processing Systems*. pages 2440–2448.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*. pages 3104–3112.

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

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

Kristina Toutanova, Xi Victoria Lin, Scott Wen tau Yih, Hoifung Poon, and Chris Quirk. 2016. Compositional learning of embeddings for relation paths in knowledge bases and text. In *ACL*.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. pages 1112–1119.

Zhuoyu Wei, Jun Zhao, and Kang Liu. 2016. Mining inference formulas by goal-directed random walks. In *EMNLP*.

Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. 2014. Knowledge base completion via search-based question answering. In *Proceedings of the 23rd International Conference on World Wide Web*. ACM, pages 515–526.

Jason Weston, Sumit Chopra, and Antoine Bordes. 2014. Memory networks. *arXiv preprint arXiv:1410.3916* .

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

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2014. Embedding entities and relations for learning and inference in knowledge bases. *CoRR* abs/1412.6575.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proc. of ACL*.

A Inference Steps in KBC

To analyze the behavior of `IRNs` in each inference step, we further pick some examples for the tail entity prediction in Table 6. Interestingly, we observed that the model can gradually increase the ranking score of the correct tail entity during the inference process.

B 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 over a shared memory. The motivations of applying our model to this task are as follows. First, we want to evaluate `IRNs` on another task requiring multi-step inference. 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 3, 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 previous 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 3.

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 et al., 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 3 shows the inference process of a test instance. Interestingly, to make the correct pre-

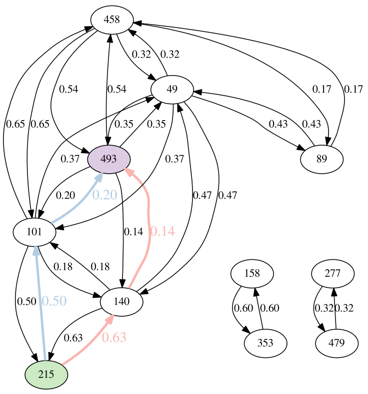
Table 6: An inference example of FB15k dataset. Given a head entity and a relation, the predictions of `IRN` in different steps associated with the corresponding termination probabilities.

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 → 158 → 89 → 458 → 493
2	∼0	N/A	215 → 479 → 277 → 353 → 493
3	∼0	N/A	215 → 49 → 493
4	∼0	0.77	215 → 140 → 493
5	0.999	0.70	215 → 101 → 493

Figure 3: An example of the shortest path synthesis dataset, given an input "215 ⇝ 493" (Answer: 215 → 101 → 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.

diction on this instance, the model has to perform a fairly complicated inference.† 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.

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