arXiv:1611.04642v5 [cs.AI] 22 Apr 2018

Link Prediction using Embedded Knowledge Graphs

Yelong Shen[‡*] **Po-Sen Huang**[‡*] **Ming-Wei Chang**[§†]

Jianfeng Gao[‡]
[‡] Microsoft Research; [§] Google Research
{yeshen, pshuang, jfgao}@microsoft.com; changmingwei@gmail.com

Abstract

Since large knowledge bases are typically incomplete, missing facts need to be inferred from observed facts in a task called knowledge base completion. The most successful approaches to this task have typically explored explicit paths through sequences of triples. These approaches have usually resorted to human-designed sampling procedures, since large knowledge graphs produce prohibitively large numbers of possible paths, most of which are uninformative. As an alternative approach, we propose performing a single, short sequence of interactive lookup operations on an embedded knowledge graph which has been trained through end-to-end backpropagation to be an optimized and compressed version of the initial knowledge base. Our proposed model, called Embedded Knowledge Graph Network (EKGN), achieves new state-of-the-art results on popular knowledge base completion benchmarks.

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. Such missing links have given rise to the open research problem of link prediction, or knowledge base completion (KBC) (Nickel et al., 2011).

Knowledge graph embedding-based methods have been popular for tackling the KBC task. In this framework, entities and relations (links) are mapped to continuous representations, then functions of those representations predict whether the two entities have a missing relationship. Equivalently, predicting whether a given edge (h, R, t) belongs in the graph can be formulated as predicting a candidate entity t given an entity h and a relation R, the input pair denoted as [h, R].

Early work Bordes et al. (2013); Socher et al. (2013); Yang et al. (2015) focused on exploring different objective functions to model *direct* relationships between two entities such as (Hawaii, PartOf, USA). Several recent approaches have demonstrated limitations of prior approaches relying upon vector-space models alone (Guu et al., 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 applying their answer to the next input Guu et al. (2015). Hence, recent work Gardner et al. (2014);

[*] Equal contribution.
[†] Work performed at Microsoft Research.

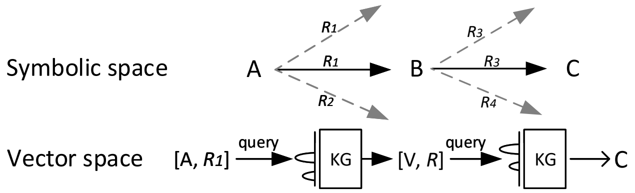


Figure 1: Contrasting examples of sampling an explicit path through sequences of triples in symbolic space vs. performing a short sequence of interactive lookup operations on an embedded knowledge graph in vector space, where A, B, and C are entities and R1, R2, R3, and R4 are relations. "E_{KG}" stands for embedded knowledge graph and [V, R] is an intermediate vector representation produced by one lookup from the entire embedded knowledge graph at once, and used to generate the query for the next lookup.

Neelakantan et al. (2015); Guu et al. (2015); Neelakantan et al. (2015); Lin et al. (2015a); Das et al. (2016); Wang and Cohen (2016); Toutanova et al. (2016); Jain (2016); Xiong et al. (2017) has proposed various approaches for injecting multi-step paths through sequences of triples during training, further improving performance in KBC tasks.

Although multi-step paths through sequences of triples achieve better performance than single steps, they also introduce technical challenges. Since the number of possible paths grows exponentially with the path length, it is prohibitive to consider all possible paths at training time for knowledge bases such as FB15k (Bordes et al., 2013). Existing approaches need to use human-designed sampling procedures (e.g., random walks) for sampling or pruning paths of observed triples in symbolic space (i.e., directly in the knowledge graph rather than the vector space of continuous representations.) As most paths are not informative for inferring missing relations, these approaches can be suboptimal.

In this paper, we aim to enrich neural knowledge completion models by embedding the knowledge graph through a training process. Instead of using human-designed sampling procedures in symbolic space and training a model separately, we propose learning to perform a sequence of interactive lookup operations on the embedded knowledge graph which is jointly trained through this same process. As a motivating example, consider an explicit path through sequences of triples in symbolic space ((Obama, BornIn, Hawaii) ∧ (Hawaii, PartOf, USA)) which provides a *connection* between two observed triples and a *path length* between the entities Obama and USA. An equivalent traversal in vector space requires the model both to represent the connections between related triples, and to know when to stop the process. Note that two triples can be "connected" by sharing either the same intermediate entity or two entities that are semantically related. Figure 1 provides an illustration of the differences between traversing a knowledge graph in symbolic space vs. vector space.

In this paper, we propose Embedded Knowledge Graph Networks (EKGNs) to realize the desired properties by means of an *embedded knowledge graph* and a *controller*. We design the embedded knowledge graph to learn a compact representation of the knowledge graph that captures the connections between related triples. We design the controller to learn to produce lookup sequences and to learn when to stop, as shown in Fig. 2. The controller, modeled by an RNN, uses a trainable termination module to decide when the EKGN should stop. On each step of a lookup sequence, the controller updates its next state based on its current state and a lookup vector obtained through attention over the embedded knowledge graph.

The embedded knowledge graph is an external weight matrix trained to be a compressed and optimized representation of triples in the training knowledge base. The trained knowledge graph is "embedded" in the sense that it represents the original knowledge base mapped through training into a vector space. An attention vector over the embedded knowledge graph provides context information that the controller uses to update its state and produce the next query in the lookup sequence. We provide interpretations and analysis of these components in the experimental section.

The main contributions of our paper are as follows:

- We propose the Embedded Knowledge Graph Network (EKGN), which is (to the best of our knowledge) the first model that considers multi-hop relations without relying on human-designed sampling procedures or explicit paths through triples.
- The proposed EKGN can store the important information from the training knowledge graph in a relatively compact representation (e.g. 64 embedded knowledge graph vectors for 483K training triples in FB15k).

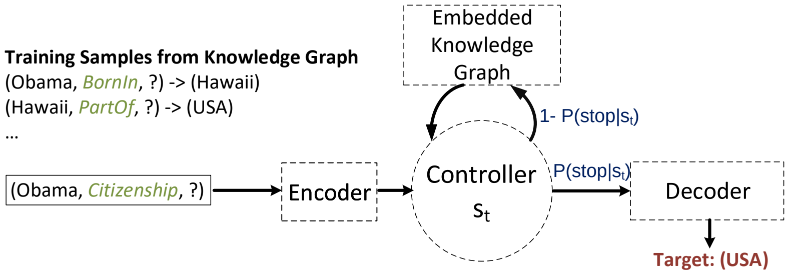


Figure 2: An overview of the EKGN for KBC tasks. The embedded knowledge graph is designed to store a compact representation of the training knowledge graph, effectively learning the connections between related triples. The controller is designed to adaptively produce lookup sequences in vector space and decide when to stop.

- Instead of sampling branching paths in the symbolic space of the original knowledge graph, EKGNs perform short sequences of interactive lookup operations in the vector space of an embedded knowledge graph.
- We evaluate EKGNs and demonstrate that they achieve new state-of-the-art results on the WN18, FB15k, and FB15k-237 benchmarks without using auxiliary information such as sampled paths or link features.
- Our analysis provides insight into the inference procedures employed by EKGNs.

2 Knowledge Base Completion Task

The goal of Knowledge Base Completion (KBC) is to infer a missing relationship between two entities, which can be formulated as predicting a head or tail entity given the other entity and the relation type. Early work on KBC focused on learning symbolic rules. Schoenmackers et al. (2010) learn inference rules from a sequence of triples. For instance, (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 triples 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 triple. In the evaluation, a triple with a missing entity $(\mathtt{h}, \mathrm{R}, \mathtt{?})$ or $(\mathtt{t}, \mathrm{R}^{-1}, \mathtt{?})$ is mapped into the vector space through embeddings, and the model outputs a prediction vector for the missing entity. The prediction is compared against all candidate entities to produce a list ranked by similarity to the prediction. Mean rank and precision of the target entity in the ranked list are used as evaluation metrics. In this paper, our proposed model uses the same task setup as in approaches of this type (Bordes et al., 2013; Socher et al., 2013; Yang et al., 2015).

3 Proposed Model

The overview of the proposed model is as follows. The *encoder* module is used to transform an input $[\mathtt{h}, \mathrm{R}]$ 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. The encoder and decoder modules are also used to convert representations between symbolic space and vector space for analysis. Effectively, the *embedded knowledge graph* learns to remember connections between relevant observed triples, and the *controller* learns to adaptively generate sequences of queries to perform lookup operations on the graph. The embedded knowledge graph is used to store a compact representation of the training knowledge graph as well as inferred connections between triples. The controller uses a termination module to determine whether to perform another lookup step, or to stop and produce the output prediction. If the decision is to continue, the controller learns to update its state representation using its previous state and a lookup vector obtained through attention over the embedded knowledge graph. Otherwise, if the decision is

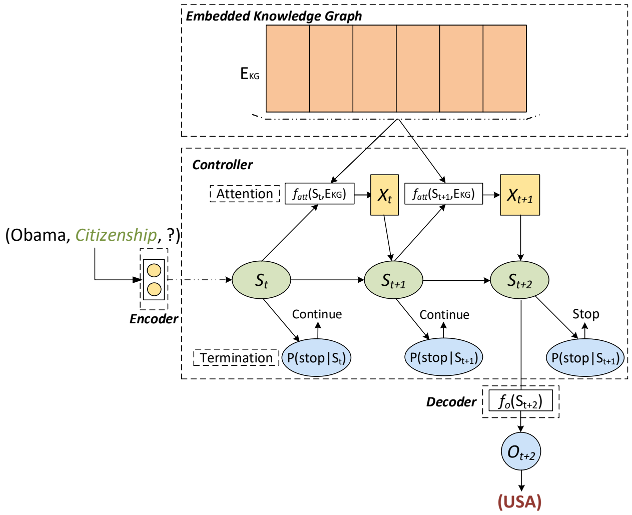


Figure 3: A running example of the EKGN architecture. After the controller's state vector is initialized from the input (Obama, CITIZENSHIP, ?), the model iteratively updates the state vector based on the current state vector and the attention vector over the embedded knowledge graph, and determines whether to stop based on the probability from the termination module.

to stop, the model produces an output prediction. Note that the number of lookup steps is free to vary 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 h and relation R embeddings from an embedding matrix, and then concatenates the two vectors to initialize the intermediate state representation s_1.

The decoder module outputs a prediction vector $f_o(s_t) = \mathtt{tanh}(W_o s_t + b_o)$, a nonlinear projection from the controller hidden state s_t, where W_o and b_o are a weight matrix and bias vector, respectively. W_o is a k-by-n matrix, where k is the dimension of the output entity embedding vector, and n is the dimension of the hidden vector s_t.

Embedded Knowledge Graph The embedded knowledge graph is designed to store a compact and optimized version of the information contained in the knowledge graph it is trained on, which means effectively storing the observed triples as well as semantic relations between them, in order to provide useful context information to the controller for updating its current state. The embedded knowledge graph is denoted as $E_{KG} = \{e_i\}_{i=1}^{|E_{KG}|}$, and consists of a list of vectors that are randomly initialized. The attention mechanism of Eq. (1) is used to access the E_{KG} during both training and inference. Note that no write operations are required since the embedded knowledge graph is updated w.r.t. the training objectives in Eq. (2) through back-propagation.

We now provide a motivating example for the intended functioning of the embedded knowledge graph. Suppose, in a KBC task, that the input is [Obama, NATIONALITY] and the model is required to output the missing entity (USA). Suppose also that the triple (Obama, NATIONALITY, USA) is not contained in the knowledge graph that was used for training. The system could still produce the correct output by relying on three items of relevant information: (1) the triple (Obama, BORNIN, Hawaii), (2) the triple (Hawaii, PARTOF, USA), and (3) the fact that BORNIN and NATIONALITY are semantically correlated relations (i.e., nationality depends on the country where one is born). Assume in this case that the controller, during training, can find items (1) and (2) in the embedded knowledge graph, but not item (3), and therefore fails to produce the right output. The training process can then modify the embedded knowledge graph E_{KG} to reflect information item (3) through back-propagation, updating the weights in such a way that the the two relations (BORNIN and NATIONALITY) are closer to each other in vector space. In this example, similar gradient updates will have previously embedded the triple items (1) and (2) into the E_{KG}.

Algorithm 1 EKGN Inference Process

```
Look up entity and relation embeddings, $\mathbf{h}$ and $\mathbf{r}$.
Set $s_1 = [\mathbf{h}, \mathbf{r}]$                                ▷ Encode
while True do
    if $u \sim P(\texttt{stop}|s_t) == 0$ and $t < T_{max}$ then
        $x_t = f_{\texttt{att}}(s_t, E_{KG})$                       ▷ Access $E_{KG}$
        $s_{t+1} = \text{RNN}(s_t, x_t), t \leftarrow t + 1$        ▷ Update RNN
    else
        Generate output $o_t = f_o(s_t)$                            ▷ Decode
        break                                                       ▷ Stop
    end if
end while
```

Limiting the size of E_{KG} (to 64 in our experiments) acts as a regularizer during training, encouraging EKGNs to store information in the E_{KG} in a general and reusable way. As a result, the E_{KG} becomes a compact representation of the training knowledge graph, optimized such that semantically related or similar triples, entities and relations are mapped to nearby locations in vector space, as we will illustrate using examples in Table 3.

Controller Given an incomplete input triple, the controller is designed to coordinate a search of the embedded knowledge graph to formulate the correct output prediction, thereby lowering the training loss. To achieve this, the controller iteratively modifies its internal state representation a small but variable number of times, incorporating context information retrieved from the embedded knowledge graph at each step. The controller decides when to stop this process, at which point the output prediction is generated. Until then, the controller continues performing updates until a maximum number of steps is reached (5 in our experiments). The controller is modeled by a recurrent neural network.

To decide when to halt this process, the controller uses a termination module to estimate $P(\texttt{stop}|s_t)$ by logistic regression: $\texttt{sigmoid}(W_c s_t + b_c)$, 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 used to generate the output. With probability $1 - P(\texttt{stop}|s_t)$ the controller will generate the next representation $s_{t+1} = \text{RNN}(s_t, x_t)$ then check the stop condition again.

The controller uses an attention mechanism to fetch information from relevant embedded knowledge graph vectors in E_{KG}. The attention vector x_t is generated based on the controller's current internal state s_t and the embedded knowledge graph E_{KG}. Specifically, the attention score $a_{t,i}$ applied to an embedded knowledge graph vector e_i given controller state s_t is computed as

$$a_{t,i} = \texttt{softmax}_i(\lambda\, cossim(W_1 e_i, W_2 s_t)) \tag{1}$$

where $cossim$ is the cosine similarity function, the weight matrices W_1 and W_2 are learned during training, and λ is chosen on the development set. ($\lambda = 10$ in our experiments.) The softmax operator normalizes the attention scores across all embedded knowledge graph vectors to produce an attention distribution over E_{KG} vectors. The resulting attention vector x_t is an interpolation of all E_{KG} vectors, and can be written as $x_t = f_{\texttt{att}}(s_t, E_{KG}) = \sum_i^{|E_{KG}|} a_{t,i} e_i$.

Overall Process The inference process is formally described in Algorithm 1. To illustrate, given [Obama, NATIONALITY] as input, the encoder module looks up the embedding vectors for Obama and NATIONALITY separately, then concatenates those embeddings into a single vector s_1, which is the controller's initial RNN state. Then the controller uses the probability $P(\texttt{stop}|s_t)$ to decide whether to stop and output the prediction vector o_t. Until the stop decision is made (or a predefined maximum step T_{max} is reached), the controller's state s_{t+1} is updated, based on its previous state s_t and on the vector x_t generated by performing attention over the embedded knowledge graph. Note that EKGN is generic, and can be applied to different applications after modifying the encoder and decoder appropriately. An example of a shortest-path synthesis task is shown in Appendix C.

3.1 Training Objectives

In this section we introduce the objective function used to train EKGN. The number of iterations in a lookup sequence is not given by the training data, but is learned and decided by the controller on the fly. Therefore, the decision to terminate processing can be treated as an action within the reinforcement

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., 2015b)	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 et al., 2016)	NO	94.9	-	73.9	-	-	-
ComplEx (Trouillon et al., 2016)	NO	94.7	-	84.0	-	-	-
TransG (Xiao et al., 2016)	NO	94.9	345	88.2	50	-	-
ConvE (Dettmers et al., 2017)	NO	95.5	504	87.3	64	45.8	330
ProjE (Shi and Weninger, 2017)	NO	-	-	88.4	**34**	-	-
RTransE (García-Durán et al., 2015a)	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	-	-	-
EKGN	NO	**95.3**	249	**92.7**	38	**46.4**	**211**

learning framework. Accordingly, the training objective is motivated by the REINFORCE algorithm Williams (1992).

The expected reward at step t can be obtained as follows. At step t, given the representation vector s_t, the model generates the output vector $o_t = f_o(s_t)$. 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 choose hyperparameters γ and $|D|$ based on the development set, setting γ to 5, and sampling 20 negative examples in D to speed up training. Defining the ground truth target entity embedding as 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, assigned to 1 on a correct prediction of the target entity, and 0 otherwise.

Rewards are then summed over all steps. The overall probability of the model terminating at time t is $\Pi_{i=1}^{t-1}(1 - v_i)v_t$, where $v_i = P(\mathtt{stop}|s_i, \theta)$. Therefore, the overall objective function can be written as

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

Given this training objective function, all parameters can be updated through back-propagation.

4 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). WN18 contains 151,442 triples with 40,943 entities and 18 relations, and FB15k consists of 592,213 triples with 14951 entities and 1,345 relations. FB15k-237 consists of 310,116 triples with 14,505 entities and 237 relations. These datasets contain multi-relations between head and tail entities.

Given a head (or tail) entity and a relation, a model produces a ranked list of the entities according to the score of the entity being the tail (or head) entity of this triple. To evaluate the ranking, we report **mean rank (MR)**, which is the mean rank of the correct entity across the test examples, and **hits@10**, which is the proportion of correct entities ranked in the top-10 predictions. Better prediction performance is indicated by lower MR or higher hits@10. We follow the evaluation protocol in Bordes et al. (2013) in reporting filtered results, where negative examples are removed from the dataset. In this way, we prevent certain negative examples from being considered valid and ranked above the target triple.

A single set of hyper-parameter settings is used for all datasets. All entity and relation embedding vectors contain 100 dimensions. The encoder module produces embeddings for input entities and relations. The decoder module produces embeddings for output entities, which are not shared with the encoder's embeddings for those same entities. The embedded knowledge graph consists of 64 real-valued vectors of 200 dimensions each, initialized randomly with unit L_2-norm. The recurrent controller is a single-layer 200-dimensional GRU. The maximum number of lookup steps, T_{max}, is set to 5. All trainable parameters are initialized randomly. The training algorithm is SGD with mini-batch size of 64, and learning rate 0.01. To prevent the model from learning a trivial solution by increasing entity embedding norms, we follow Bordes et al. (2013) in constraining the L_2-norm of the entity embeddings to be 1. The validation metric for `EKGN` is hits@10. Following Lin et al. (2015a), we add reverse relations into the training triple set to increase training data, i.e., for each triple (h, r, t), we build two training instances, (h, r, t) and (t, r^{-1}, h).

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 contains the models that make use of auxiliary information from multi-step relations. For example, the RTransE (García-Durán et al., 2015a) and PTransE (Lin et al., 2015a) models extend the TransE (Bordes et al., 2013) model by explicitly exploring multi-step relations in the knowledge base to regularize the trained embeddings. NLFeat (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 new 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%. We also report the results of `EKGNs` with different embedded knowledge graph sizes $|E_{KG}|$ and different T_{max} on FB15k in Appendix A.

In order to explore the inference procedure learned by `EKGNs`, we map the representation s_t back to human-interpretable entity and relation names in the KB. In Table 2, 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. These mappings seem to reflect an inference procedure that is quite unlike the paths through sequences of triples in symbolic space employed by prior work (Schoenmackers et al., 2010). One potential explanation is that `EKGNs` exercise greater flexibility by operating in the vector space of the embedded knowledge graph. Instead of being constrained to connect only triples that share exactly the same entities in symbolic space, `EKGNs` can update the representations themselves and connect other, semantically related triples in vector space instead. As shown in Table 2, the model reformulates the representation s_t at each step and gradually increases the ranking score of the correct tail entity, generating progressively higher termination probabilities during the inference process. In the last step of Table 2, 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. These findings suggest that the `EKGN` inference process iteratively reformulates the representation s_t in order to minimize the distance between the decoder's output and the target entity in vector space.

To investigate what the model learned to store when trained on FB15k, we randomly selected six vectors from the embedded knowledge graph, and computed the average attention scores of each relation type for each vector. Table 3 shows the (8) top-scoring relations for each of the six vectors. Most of a vector's top relations seem clustered around a common theme, described in the last row, illustrating how these embedded knowledge graph vectors are activated by certain semantic patterns within the knowledge graph. This suggests that the embedded knowledge graph can capture the semantic connections between triples. But the patterns are broken by a few noisy relations in each column, e.g., the "bridge-player-teammates/teammate" relation in the "film" vector column, and

Table 2: Interpretation of state s at each step, obtained by finding the closest (entity, relation) tuple, and the corresponding top-3 predictions. "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

Table 3: Embedded knowledge graph visualization for an EKGN trained on FB15k. Each column shows the top 8 relations (ranked by their average attention scores) of one randomly selected vector in the embedded knowledge graph.

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

"olympic-medal-honor/medalist" in the "disease" vector column. We provide more EKGN prediction examples at each step from FB15k in Appendix B.

5 Related Work

5.1 Link Prediction using Multi-step Relations

Recently, several studies Gardner et al. (2014); Neelakantan et al. (2015); Guu et al. (2015); Neelakantan et al. (2015); Lin et al. (2015a); Das et al. (2016); Wang and Cohen (2016); Toutanova et al.

(2016); Jain (2016); Xiong et al. (2017) have demonstrated the importance of models incorporating multi-step paths through sequences of triples during training. Learning from multi-step relations injects the structured relationships between triples into the model. However, this also poses the technical challenge of considering exponential numbers of multi-step relationships. Prior approaches addressed 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. (2015) and Das et al. (2016) use an RNN to model multi-step relationships over a set of random walk paths on the observed triples. Toutanova and Chen (2015) shows the effectiveness of using simple node and link features that encode structured information on FB15k and WN18. Xiong et al. (2017) and Das et al. (2017) use RL to traverse a knowledge graph in symbolic space, which can be viewed as an example of a human-designed sampling procedure over a knowledge graph.

In our work, EKGN outperforms prior results[3] and shows that multi-step relationships can be captured by interactive lookup operations on an embedded knowledge graph. EKGN adaptively learns to construct the embedded knowledge graph, and learns to look up relevant information from the embedded knowledge graph. Each EKGN lookup operation potentially draws upon the entire embedded knowledge graph, instead of sampling a single path.

5.2 Neural Frameworks

EKGNs share certain features with Memory Networks (MemNN) (Weston et al., 2014; Miller et al., 2016; Jain, 2016) and Neural Turing Machines (NTM) (Graves et al., 2014, 2016). The distinctions of EKGNs are found in its controller and in its usage of the embedded knowledge graph (which corresponds to external memory in MemNN and NTM). MemNN and NTM explicitly store embedded inputs (such as graph triples or supporting facts) in their external memories. In contrast, EKGN stores no information in the embedded knowledge graph directly or explicitly. Instead, the embedded knowledge graph is initialized with random vectors, then trained (jointly with the controller) to implicitly learn a compact and (task-specific) optimized representation of the structured relationships in the training knowledge graph.

We adapt the stopping mechanism from Shen et al. (2017). Compared to Shen et al. (2017), EKGNs adaptively accumulate and aggregate information in the embedded knowledge graph, whereas Shen et al. (2017) only constructs a paragraph embedding matrix by using the embedding of each word in an article. The paragraph embedding is not shared across samples.

6 Conclusion

In this paper, we propose Embedded Knowledge Graph Networks (EKGNs) that learn to build and query an embedded knowledge graph. Without using auxiliary information, the embedded knowledge graph learns to store large-scale structured relationships from the training knowledge graph in a compact way, optimized for the training task, while the controller jointly learns to update its representations, generate lookup sequences, and decide when to stop and output the prediction. We demonstrate and analyze the EKGN inference process in KBC tasks. Our model, without using any auxiliary knowledge base information, achieves new state-of-the-art results on the WN18, FB15k, and FB15k-237 benchmarks. In future work, we aim to further develop EKGNs for downstream applications such as knowledge base question answering.

Acknowledgments

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

References

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

[3] We cannot compare the performance directly with Xiong et al. (2017) as that would require training one model for each relation, which does not scale up to a knowledge graph with a large number of relations.

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

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

Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. 2017. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. *arXiv preprint arXiv:1711.05851*.

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

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2017. Convolutional 2d knowledge graph embeddings. *CoRR*, abs/1707.01476.

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

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

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

Matt Gardner, Partha Pratim Talukdar, Jayant Krishnamurthy, and Tom Mitchell. 2014. Incorporating vector space similarity in random walk inference over knowledge bases. In *EMNLP*.

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 *CIKM*, pages 623–632.

Sarthak Jain. 2016. Question answering over knowledge base using factual memory networks. In *NAACL - WordNet and Other Lexical Resources Workshop*.

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

Yankai Lin, Zhiyuan Liu, Huanbo Luan, Maosong Sun, Siwei Rao, and Song Liu. 2015a. Modeling relation paths for representation learning of knowledge bases. In *EMNLP*.

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015b. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, 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 *IJCNLP*, 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, Lorenzo Rosasco, and Tomaso Poggio. 2016. Holographic embeddings of knowledge graphs. In *AAAI*, pages 1955–1961.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A three-way model for collective learning on multi-relational data. In *ICML*, pages 809–816.

Stefan Schoenmackers, Oren Etzioni, Daniel S Weld, and Jesse Davis. 2010. Learning first-order Horn clauses from web text. In *EMNLP*, pages 1088–1098.

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

Baoxu Shi and Tim Weninger. 2017. Proje: Embedding projection for knowledge graph completion. In *AAAI*.

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

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

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*.

Kristina Toutanova and Danqi Chen. 2015. Observed versus latent features for knowledge base and text inference. In *CVSC*.

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*.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *ICML*, pages 2071–2080.

William Yang Wang and William W Cohen. 2016. Learning first-order logic embeddings via matrix factorization. In *IJCAI*.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *AAAI*.

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 *WWW*, 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.

Han Xiao, Minlie Huang, and Xiaoyan Zhu. 2016. TransG : A generative model for knowledge graph embedding. In *ACL*.

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

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

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 *ACL*.

A Analysis: Embedded Knowledge Graph Size and Inference Steps in KBC

To provide additional insight into the behavior of EKGNs, Table 4 reports the results on FB15k for EKGNs using different hyperparameter settings. Note that an EKGN with T_{max} = 1 is equivalent to having no embedded knowledge graph. The table shows that the two performance metrics behave differently over hyperparameter settings.

Larger embedded knowledge graphs (to some extent) and larger maximum numbers of inference steps (to a significant extent) consistently improve the MR score, but the best hit@10 is obtained by $|E_{KG}| = 64$ and T_{max} = 5. For both metrics, performance is much more sensitive to T_{max} than it is to $|E_{KG}|$. One possible reason is that the optimal $|E_{KG}|$ is more strongly determined by the complexity of a given task than is T_{max}.

Table 4: The performance of EKGNs on the FB15k test set, using different numbers of vectors in the embedded knowledge graph ($|E_{KG}|$), and different maximum numbers of inference steps (T_{max}). Hyperparameter settings were not selected based on these results; they were selected based on development set results.

$\|E_{KG}\|$	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**

B Inference Steps in KBC

For further analysis of the behavior of EKGNs over the sequence of lookup steps, Table 5 gives more examples of tail entity prediction. Interestingly, the rank of the correct tail entity improves consistently throughout the inference process.

Table 5: Two inference examples from the FB15k dataset. Given the head entity and relation, the EKGN predictions at different steps are shown with their corresponding termination probabilities.

Input: [Dean Koontz, /PEOPLE/PERSON/PROFESSION]
Target: Film Producer

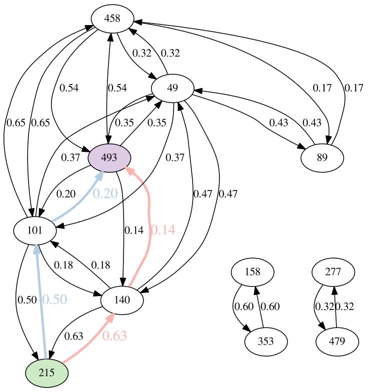
Step	Termination Prob.	Answer Rank	Top-3 Predicted Entities		
1	0.018	9	Author	TV. Director	Songwriter
2	0.052	7	Actor	Singer	Songwriter
3	0.095	4	Actor	Singer	Songwriter
4	0.132	4	Actor	Singer	Songwriter
5	0.702	3	Actor	Singer	**Film Producer**

Input: (War and Peace, /FILM/FILM/PRODUCED_BY)
Target: Carlo Ponti

Step	Termination Prob.	Answer Rank	Top-3 Predicted 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

C Analysis: Applying EKGNs to a Shortest Path Synthesis Task

To further investigate the inference capabilities of EKGNs, we constructed a synthetic *shortest path synthesis* task involving multi-step relations.



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: $(215) \rightarrow 101 \rightarrow (493)$). Only the nodes related to this example are shown, alongside the corresponding termination probability and prediction results. The model terminates on step 5.

As illustrated in Figure 4, the input for each training instance consists of a start node and an end node (e.g., 215 $\rightsquigarrow$ 493) within a (hidden) weighted directed graph. The output of each instance is the set of intervening nodes in the shortest path between the given start and end nodes, e.g., $(215) \rightarrow 101 \rightarrow (493)$. The edge weights of the graph determine the lengths of paths, but are not directly revealed during training. At test time, the model receives a start node and end node, and outputs a set of one or more intervening nodes. An output sequence is considered correct if it connects the start node to the end node in the underlying graph, and if the cost of the predicted path is the same as the optimal path.

The nodes of the underlying graph were defined by randomly sampling 500 points on a three-dimensional unit sphere. Each node was then connected to its k-nearest neighbors, using the Euclidean distances between nodes as the edge weights. ($k = 50$ by default.) The dataset was then constructed by repeatedly sampling a random pair of nodes, and finding the shortest path connecting the first node to the second. Paths were added to the dataset only if they contained at least one intervening node. To test multi-step inference rather than simple memorization, paths were added only if they did not contain existing paths as sub-paths, and were not contained as sub-paths within existing paths.

Note that the task is very difficult and *cannot* be solved perfectly by dynamic programming algorithms since the weights on the edges are not directly revealed. To recover some of the shortest paths at 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 predict 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 shorter using statistical information.

The dataset was split into 20,000 instances for training, 10,000 instances for validation, and 10,000 instances for testing. For this sequence generation task, a GRU decoder (with 128 cells) was used as the EKGN output module, along with a GRU controller (with 128 cells). Reward R was assigned to 1 for an instance if all the predicted symbols were correct and 0 otherwise. A 64-dimensional embedding vector was used for input symbols. The maximum inference step T_{max} was set to 5.

We compare the EKGN with two baseline approaches: dynamic programming without edge-weight information, and a standard sequence-to-sequence model (Sutskever et al., 2014) using a similar number of parameters as the EKGN. Without knowing the edge weights, dynamic programming recovers only 589 correct paths at test time. The sequence-to-sequence model recovers 904 correct paths. The EKGN outperforms both baselines, recovering 1,319 paths. Furthermore, 76.9% of the paths predicted by EKGN are *valid* paths (connecting the start and end nodes of the underlying graph). In contrast, only 69.1% of the paths predicted by the sequence-to-sequence model are valid.

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

[4] 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 that 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.)