Contextual Graph Attention for Answering Logical Queries over Incomplete Knowledge Graphs

Gengchen Mai, Krzysztof Janowicz, Bo Yan, Rui Zhu, Ling Cai, Ni Lao
STKO Lab, UCSB
{gengchen_mai,jano,boyan,ruizhu,lingcai}@geog.ucsb.edu
SayMosaic Inc.
ni.lao@mosaix.ai

ABSTRACT

Recently, several studies have explored methods for using KG embedding to answer logical queries. These approaches either treat embedding learning and query answering as two separated learning tasks, or fail to deal with the variability of contributions from different query paths. We proposed to leverage a graph attention mechanism [20] to handle the unequal contribution of different query paths. However, commonly used graph attention assumes that the center node embedding is provided, which is unavailable in this task since the center node is to be predicted. To solve this problem we propose a multi-head attention-based end-to-end logical query answering model, called Contextual Graph Attention model (CGA), which uses an initial neighborhood aggregation layer to generate the center embedding, and the whole model is trained jointly on the original KG structure as well as the sampled query-answer pairs. We also introduce two new datasets, *DB18* and *WikiGeo19*, which are rather large in size compared to the existing datasets and contain many more relation types, and use them to evaluate the performance of the proposed model. Our result shows that the proposed CGA with fewer learnable parameters consistently outperforms the baseline models on both datasets as well as Bio [5] dataset.

CCS CONCEPTS

• **Computing methodologies → Knowledge representation and reasoning**; **Neural networks**.

KEYWORDS

Knowledge Graph Embedding, Logical Query Answering, Multi-head Attention Model

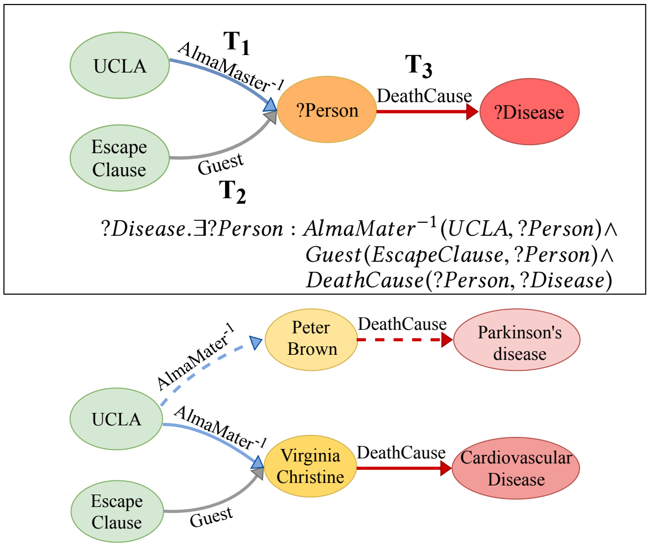
1 INTRODUCTION

Knowledge graphs represent statements in the form of graphs in which nodes represent entities and directed labeled edges indicate different types of relations between these entities [12]. In the past decade, the Semantic Web community has published and interlinked vast amounts of data on the Web using the machine-readable and reasonable Resource Description Framework (RDF) in order to create smart data [**?**]. By following open W3C standards or related proprietary technology stacks, several large-scale knowledge graphs have been constructed (e.g., DBpedia, Wikidata, NELL, Google's Knowledge Graph and Microsoft's Satori) to support applications such as information retrieval and question answering [3, 10].

Despite their size, knowledge graphs often suffer from incompleteness, sparsity, and noise as most KGs are constructed collaboratively and semi-automatically [23]. Recent work studied different ways of applying graph learning methods to large-scale knowledge graphs to support completion via so-called knowledge graph embedding techniques such as RESCAL [16], TransE [4], NTN [18], DistMult [24], TransR [11], and HOLE [15]. These approaches aim at embedding KG components including entities and relations into continuous vector spaces while preserving the inherent structure of the original KG [22]. Although these models show promising results in link prediction and entity classification tasks, they all treat each statement (often called *triple*) independently, thereby ignoring the correlation between them. In addition, since the model needs to rank all entities for a given triple in the link prediction task, their complexity is linear with respect to the total number of entities in the KG, which makes it impractical for more complicated query answering tasks.

Recent work [5, 13, 21] has explored ways to utilize knowledge graph embedding models for answering logical queries from incomplete KG. The task is to predict the correct answer to a query based on KG embedding models, even if this query cannot be answered directly because of one or multiple missing triples in the original graph. For example, Listing 1 shows an example SPARQL query over DBpedia which asks for the cause of death of a person whose alma mater was UCLA and who was a guest of Escape Clause. Executing this query via DBpedia SPARQL endpoint[1] yields one answer `dbr:Cardiovascular_disease` and the corresponding person is `dbr:Virginia_Christine`. However, if the triple (`dbr:Virginia_Christine dbo:deathCause dbr:Cardiovascular_disease`) is missing, this query would become an unanswerable one [13] as shown in Figure 1. The general idea of query answering via KG embedding is to predict the embedding of the root variable *?Disease* by utilizing the embeddings of known entities (e.g. `UCLA` and `EscapeClause`) and relations (`deathCause`, `almaMater` and `guest`) in the query. Ideally, a nearest neighbor search in the entity

[1] https://dbpedia.org/sparql



$$?Disease.\exists ?Person : AlmaMater^{-1}(UCLA, ?Person) \wedge Guest(EscapeClause, ?Person) \wedge DeathCause(?Person, ?Disease)$$

Figure 1: Top box: Conjunctive Graph Query (CGQ) and DAG of the query structure. Below: the matched underlining KG patterns represented by solid arrows.

embedding space using the predicted variable's embedding yields the approximated answer.

```
SELECT ?Disease
WHERE {
?Person dbo:deathCause ?Disease.
?Person dbo:almaMater dbr:
    ↪ University_of_California,
    ↪ _Los_Angeles .
dbr:Escape_Clause dbo:guest ?Person .
}
```

Listing 1: An example SPARQL query over DBpedia

Hamilton et al. [5] and Wang et al. [21] proposed different approaches for predicting variable embedding. However, an unavoidable step for both is to *integrate* predicted embeddings for the same variable (in this query *?Person*) from different paths (triple T_1 and T_2 in Fig. 1) by translating from the corresponding entity nodes via different relation embeddings. In Figure 1, triple T_1 and T_2 will produce different embeddings $\mathbf{p_1}$ and $\mathbf{p_2}$ for variable *?Person* and they need to be integrated to produce one single embedding $\mathbf{p}$ for *?Person*. An intuitive integration method is an element-wise mean operation over $\mathbf{p_1}$ and $\mathbf{p_2}$. This implies that we assume triple T_1 and T_2 have equal prediction abilities for the embedding of *?Person* which is not necessarily true. In fact, triple T_1 matches 450 triples in DBpedia while T_2 only matches 5. This indicates that $\mathbf{p_2}$ will be more similar to the real embedding of *?Person* because T_2 has more discriminative power.

Wang et al. [21] acknowledged this unequal contribution from different paths and obtained the final embedding $\mathbf{p}$ as a weighted average of $\mathbf{p_1}$ and $\mathbf{p_2}$ while the weight is proportional to the inverse of the number of triples matched by triple T_1 and T_2. However, this deterministic weighting approach lacks flexibility and will produce suboptimal results. Moreover, they separated the knowledge graph embedding training and query answering steps. As a result, the KG embedding model is not directly optimized on the query answering objective which further impacts the model's performance.

In contrast, Hamilton et al. [5] presented an end-to-end model for KG embedding model training and logical query answering. However, they utilized a simple permutation invariant neural network [25] to *integrate* $\mathbf{p_1}$ and $\mathbf{p_2}$ which treats each embedding equally. Furthermore, in order to train the end-to-end logical query answering model, they sampled logical query-answer pairs from the KG as training datasets while ignoring the original KG structure which has proven to be important for embedding model training based on previous research [9].

Based on these observations, we hypothesis that a graph attention network similar to the one proposed by Veličković et al. [20] can handle these unequal contribution cases. However, Veličković et al. [20] assume that the center node embedding (the variable embedding of *?Person* in Fig. 1), known as the *query embedding* [19], should be known beforehand for attention score computing which is unknown in this case. This prevents us from using the normal attention method. Therefore, we propose an end-to-end attention-based logical query answering model over knowledge graphs in which the situation of unequal contribution from different paths to an entity embedding is handled by a new attention mechanism [2, 19, 20] where **the center variable embedding is no longer a prerequisite**. Additionally, the model is jointly trained on both sampled logical query-answer pairs and the original KG structure information. **The contributions of our work are as follows:**

(1) We propose an end-to-end attention-based logic query answering model over knowledge graphs in which an attention mechanism is used to handle the unequal contribution of neighboring entity embeddings to the center entity embedding. To the best of our knowledge, this is the first attention method applicable to logic query answering.
(2) We show that the proposed model can be trained jointly on the original KG structure and the sampled logical QA pairs.
(3) We introduce two datasets - *DB18* and *WikiGeo19* - which have substantially more relation types (170+) compared to the Bio dataset [5].

The rest of this paper is structured as follows. We first introduce some basic notions in Section 2 and present our attention-based query answering model in Section 3. In Section 4, we discuss the datasets we used to evaluate our model and present the evaluation results. We conclude our work in Section 5.

2 BASIC CONCEPTS

Before introducing our end-to-end attention-based logical query answering model, we outline some basic notions relevant to Conjunctive Graph Query models.

2.1 Conjunctive Graph Queries (CGQ)

In this work, a knowledge graph (KG) is a directed and labeled multi-relational graph $\mathcal{G} = (\mathcal{V}, \mathcal{R})$ where $\mathcal{V}$ is a set of entities (nodes), $\mathcal{R}$ is the set of relations (predicates, edges); furthermore let $\mathcal{T}$ be a set of triples. A triple $T_i = (h_i, r_i, t_i)$ or $r_i(h_i, t_i)$ in this sense consists of a head entity h_i and a tail entity t_i connected by some relation r_i (predicate).[2]

Definition 2.1 (Conjunctive Graph Query (CGQ)). A query $q \in Q(\mathcal{G})$ that can be written as follows:

$$
\begin{aligned}
& q = V_?.\exists V_1, V_2, .., V_m : b_1 \wedge b_2 \wedge ... \wedge b_n \\
\textit{where} \quad & b_i = r(e_k, V_l), V_l \in \{V_?, V_1, V_2, .., V_m\}, e_k \in \mathcal{V}, r \in \mathcal{R} \\
\textit{or} \quad & b_i = r(V_k, V_l), V_k, V_l \in \{V_?, V_1, V_2, .., V_m\}, k \neq l, r \in \mathcal{R}
\end{aligned}
$$

Here $V_?$ denotes the target variable of the query which will be replaced with the answer entity, while $V_1, V_2, .., V_m$ are existentially quantified bound variables. b_i is a basic graph pattern in this CGQ. To ensure q is a valid CGQ, the dependence graph of q must be a *directed acyclic graph (DAG)* [5] in which the entities (anchor nodes) e_k in q are the source nodes and the target variable $V_?$ is the unique sink node.

Figure 1 shows an example CGQ which is equivalent to the SPARQL query in Listing 1, where *?Person* is an existentially quantified bound variable and *?Disease* is the target variable. Note that for graph pattern $r(s, o)$ where subject s is a variable and object o is an entity, we can convert it into the form $b_i = r(e_k, V_l)$ by using the inverse relation of the predicate r. In other words, we convert $r(s, o)$ to $r^{-1}(o, s)$. For example, In Figure 1, we use $AlmaMater^{-1}(UCLA, ?Person)$ to represent the graph pattern $AlmaMater(?Person, UCLA)$. The benefit of this inverse relation conversion is that we can construct CGQ where the dependence graph is a *directed acyclic graph (DAG)* as shown in Figure 1 .

Comparing Definition 2.1 with SPARQL, we can see several differences:

1. Predicates in CGQs are assumed to be fixed while predicates in a SPARQL 1.1 basic graph pattern can also be variables [13].
2. CGQs only consider the conjunction of graph patterns while SPARQL 1.1 also contains other operations (UNION, OPTION, FILTER, LIMIT, etc.).
3. CGQs require one variable as the answer denotation, which is in alignment with most question answering over knowledge graph literature [3, 10]. In contrast, SPARQL 1.1 allows multiple variables as the returning variables. The unique answer variable property make it easier to evaluate the performance of different deep learning models on CGQs.

2.2 Geometric Operators in Embedding Space

Here we describe two geometric operators - the projection operator and the intersection operator - in the entity embedding space, which were first introduced by Hamilton et al. [5].

Definition 2.2 (Geometric Projection Operator). Given an embedding $\mathbf{e}_i \in \mathbb{R}^d$ in the entity embedding space which can be either an embedding of a real entity e_i or a computed embedding for an existentially quantified bound variable V_i in a conjunctive query q, and a relation r, the projection operator $\mathcal{P}$ produces a new embedding $\mathbf{e}'_i = \mathcal{P}(e_i, r)$ where $\mathbf{e}'_i \in \mathbb{R}^d$. The projection operator is defined as follows:

$$\mathbf{e}'_i = \mathcal{P}(e_i, r) = \mathbf{R}_r \mathbf{e}_i \tag{1}$$

where $\mathbf{R}_r \in \mathbb{R}^{d \times d}$ is a trainable and relation-specific matrix for relation type r. The embedding $\mathbf{e}'_i = \mathcal{P}(e_i, r)$ denotes all entities that connect with entity e_i or variable V_i through relation r. If embedding $\mathbf{e}_i$ denotes entity e_i, then $\mathbf{e}'_i = \mathcal{P}(\mathbf{e}_i, r)$ denotes $\{e_k | r(e_i, e_k) \in \mathcal{G}\}$. If embedding $\mathbf{e}_i$ denotes variable V_i, then $\mathbf{e}'_i = \mathcal{P}(e_i, r)$ denotes $\{e_k | e_j \in V_i \wedge r(e_j, e_k) \in \mathcal{G}\}$.

In short, $\mathbf{e}'_i = \mathcal{P}(\mathbf{e}_i, r)$ denotes the embedding of the relation r specific neighboring set of entities. Different KG embedding models have different ways to represent the relation r. We can also use TransE's version ($\mathbf{e}'_i = \mathbf{e}_i + \mathbf{r}$) or a diagonal matrix version ($\mathbf{e}'_i = diag(\mathbf{r})\mathbf{e}_i$, where $diag(\mathbf{r})$ is a diagonal matrix parameterized by vector $\mathbf{r}$ in its diagonal axis). The bilinear version shown in Equation 1 has the best performance in logic query answering because it is more flexible in capturing different characteristics of relation r [5].

As for the intersection operator, we first present the original version from Graph Query Embedding (GQE) [5], which will act as baseline for our model.

Definition 2.3 (Geometric Intersection Operator). Assume we are given a set of n different input embeddings $\mathbf{e}'_1$, $\mathbf{e}'_2$, ..., $\mathbf{e}'_i$,..., $\mathbf{e}'_n$ as the outputs from n different geometric projection operations $\mathcal{P}$ by following n different relation r_j paths. We require all $\mathbf{e}'_i$ to have the same entity type. The geometric intersection operator outputs one embedding $\mathbf{e}''$ based on this set of embeddings which denotes the intersection of these different relation paths:

$$\mathbf{e}'' = \mathcal{I}_{GQE}(\{\mathbf{e}'_1, \mathbf{e}'_2, \ldots, \mathbf{e}'_i, \ldots, \mathbf{e}'_n\}) = \mathbf{W}_{\gamma 1}\Psi(ReLU(\mathbf{W}_{\gamma 2}\, \mathbf{e}'_i), \forall i \in \{1, 2, .., n\}) \tag{2}$$

where $\mathbf{W}_{\gamma 1}, \mathbf{W}_{\gamma 2} \in \mathbb{R}^{d \times d}$ are trainable entity type γ specific matrices. $\Psi()$ is a symmetric vector function (e.g., an element-wise mean or minimum of a set of vectors) which is permutation invariant on the order of its inputs [25]. As $\mathbf{e}'_1$, $\mathbf{e}'_2$, ..., $\mathbf{e}'_i$,..., $\mathbf{e}'_n$ represent the embeddings of the neighboring set of entities, $\mathbf{e}'' = \mathcal{I}_{GQE}(\{\mathbf{e}'_1, \mathbf{e}'_2, \ldots, \mathbf{e}'_i, \ldots, \mathbf{e}'_n\})$ is interpreted as the intersection of these sets.

2.3 Entity Embedding Initialization

Generally speaking, any (knowledge) graph embedding model can be used to initialize entity embeddings. In this work, we adopt the simple "bag-of-features" approach. We assume each entity e_i will have an entity type $\gamma = \Gamma(e_i)$, e.g. `Place`, `Agent`. The entity embedding lookup is shown below:

$$\mathbf{e}_i = \frac{\mathbf{Z}_\gamma \mathbf{x}_i}{\parallel \mathbf{Z}_\gamma \mathbf{x}_i \parallel_{L2}} \tag{3}$$

$\mathbf{Z}_\gamma \in \mathbb{R}^{d \times m_\gamma}$ is the type-specific embedding matrices for all entities with type $\gamma = \Gamma(e_i)$ which can be initialized using a normal embedding matrix normalization method. The $\mathbf{x}_i$ is a binary feature vector such as a one-hot vector which uniquely identifies entity e_i among all entities with the same entity type γ. The $\parallel \cdot \parallel_{L2}$ indicates the $L2$-norm. The reason why we use type-specific embedding matrices rather than one embedding matrix for all entities as [4, 8, 11, 15, 16, 18, 24] did is that recent node embedding work [5, 6] show that most of the information contained in the node embeddings is type-specific information. Using type-specific entity embedding

[2] Note that in many knowledge graphs, a triple can include a datatype property as the relation where the tail is a literal. In line with related work [14, 22] we do not consider this kind of triples here. We will use head (h), relation (r), and tail(t) when discussing embeddings and subject (s), predicate (p), object (o) when discussing Semantic Web knowledge graphs to stay in line with the literature from both fields.

matrices explicitly handles this information. Note that in many KGs such as DBpedia one entity may have multiple types. We handle this by computing the common super class of these types (see Sec. 4).

3 METHOD

Next, we discuss the difference between our model and GQE [5]. Our geometric operators (1) use an attention mechanism to account for the fact that different paths have different embedding prediction abilities with respect to the center entity embedding and (2) can be applied to two training phases – training on the original KG and training with sampled logic query-answer pairs.

3.1 Attention-based Geometric Projection Operator

Since the permutation invariant function $\Psi()$ directly operates on the set $\{ReLU(\mathbf{W}_{\gamma 2}\ \mathbf{e}'_i) | \forall i \in \{1, 2, .., n\}\}$, Equation 2 assumes that each $\mathbf{e}'_i$ (relation path) has an equal contribution to the final intersection embedding $\mathbf{e}''$. This is not necessarily the case in real settings as we have discussed in Section 1. Graph Attention Network (GAT) [20] has shown that using an attention mechanism on graph-structured data to capture the unequal contribution of the neighboring nodes to the center node yields better result than a simple element-wise mean or minimum approaches. By following the attention idea of GAT, we propose an attention-based geometric intersection operator.

Assume we are given the same input as Definition 2.3, a set of n different input embeddings $\mathbf{e}'_1$, $\mathbf{e}'_2$, ..., $\mathbf{e}'_i$,..., $\mathbf{e}'_n$. The geometric intersection operator contains two layers: a multi-head attention layer and a feed forward neural network layer.

3.1.1 *The multi-head attention layer.*

The initial intersection embedding $\mathbf{e}''_{init}$ is computed as:

$$\mathbf{e}''_{init} = \Psi(\mathbf{e}'_i, \forall i \in \{1, 2, .., n\}) \tag{4}$$

Then the attention coefficient for each $\mathbf{e}'_i$ in the k^{th} attention head is

$$\alpha_{ik} = \mathrm{A}_k(\mathbf{e}''_{init}, \mathbf{e}'_i) = \frac{exp(LeakyReLU(\mathbf{a}^T_{\gamma k}[\mathbf{e}''_{init}; \mathbf{e}'_i]))}{\sum_{j=1}^{n} exp(LeakyReLU(\mathbf{a}^T_{\gamma k}[\mathbf{e}''_{init}; \mathbf{e}'_j]))} \tag{5}$$

where $\cdot^T$ represents transposition, $[;]$ vector concatenation, and $\mathbf{a}_{\gamma k} \in \mathbb{R}^{d \times 2}$ is the type-specific trainable attention vector for k^{th} attention head. Following the advice on avoiding spurious weights [20], we use *LeakyReLu* here.

The attention weighted embedding $\mathbf{e}''_{attn}$ is computed as the weighted average of different input embeddings while weights are automatically learned by the multi-head attention mechanism. Here, $\sigma()$ is the sigmoid activation function and K is the number of attention heads.

$$\mathbf{e}''_{attn} = \sigma(\frac{1}{K}\sum_{k=1}^{K}\sum_{i=1}^{n}\alpha_{ik}\mathbf{e}'_i) \tag{6}$$

Furthermore, we add a residual connection [7] of $\mathbf{e}''_{attn}$, followed by layer normalization [1] (Add & Norm).

$$\mathbf{e}''_{ln1} = LayerNorm_1(\mathbf{e}''_{attn} + \mathbf{e}''_{init}) \tag{7}$$

3.1.2 *The second layer.*

It is a normal feed forward neural network layer followed by the "Add & Norm" as shown in Equation 8.

$$\mathbf{e}'' = \mathcal{I}_{CGA}(\{\mathbf{e}'_1, \mathbf{e}'_2, ..., \mathbf{e}'_i, ..., \mathbf{e}'_n\}) = LayerNorm_2(\mathbf{W}_\gamma \mathbf{e}''_{ln1} + \mathbf{B}_\gamma + \mathbf{e}''_{ln1}) \tag{8}$$

where $\mathbf{W}_\gamma \in \mathbb{R}^{d \times d}$ and $\mathbf{B}_\gamma \in \mathbb{R}^d$ are trainable entity type γ specific weight matrix and bias vector, respectively, in a feed forward neural network.

Figure 2 illustrates the model architecture of our attention-based geometric intersection operator. The light green boxes at the bottom indicate n embeddings $\mathbf{e}_1$, $\mathbf{e}_2$,...,$\mathbf{e}_i$,...,$\mathbf{e}_n$, which are projected by the geometric projection operators. The output embeddings $\mathbf{e}'_1$, $\mathbf{e}'_2$, ..., $\mathbf{e}'_i$,..., $\mathbf{e}'_n$ are the n input embeddings of our intersection operator. The initial intersection embedding $\mathbf{e}''_{init}$ is computed based on these input embeddings as shown in Equation 4. Next, $\mathbf{e}''_{init}$ and $\mathbf{e}'_1$, $\mathbf{e}'_2$, ..., $\mathbf{e}'_i$,..., $\mathbf{e}'_n$ are fed into the multi-head attention layer followed by the feed forward neural network layer. This two-layer architecture is inspired by Transformer [19].

The multi-head attention mechanism shown in Equation 4, 5, and 6 is similar to those used in Graph Attention Network (GAT) [20]. The major difference is the way we compute the initial intersection embedding $\mathbf{e}''_{init}$ in Equation 4. In the graph neural network context, the attention function can be interpreted as mapping *the center node embedding* and *a set of neighboring node embeddings* to an output embedding. In GAT, the model directly operates on the local graph structure by applying one or multiple convolution operations over the 1-degree neighboring nodes of the center node. In order to compute the attention score for each neighboring node embedding, each of the neighboring node embedding is compared with the embedding of the center node for attention score computation. Here, the center node embedding is known in advance.

However, in our case, since we want to train our model directly on the logical query-answer pairs (**query-answer pair training phase**), the final intersection embedding $\mathbf{e}''$ might denote the variable in a conjunctive graph query q whose embedding is unknown. For example, in Figure 1, we can obtain two embeddings $\mathbf{p}_1$ and $\mathbf{p}_2$ for variable *?Person* by following two different triple path T_1 and T_2. In this case, the input embeddings for our intersection operator are $\mathbf{p}_1$ and $\mathbf{p}_2$. The *center node embedding* here is the true embedding for variable *?Person* which is unknown. Equation 4 is used to compute an initial embedding for the center node, the variable *?Person*, in order to compute the attention score for each input embedding.

Note that these two intersection operators in Definition 2.3 and Section 3.1 can also be *directly applied to the local knowledge graph structure* as R-GCN [17] does (**original KG training phase**). The output embedding $\mathbf{e}''$ can be used as the new embedding for the center entity which is computed by a convolution operation over its 1-degree neighboring entity-relation pairs. In this KG training phase, although the center node embedding is known in advance, in order to make our model applicable to both of these two training phases, we still use the initial intersection embedding idea. Note that the initial intersection embedding computing step (see Equation 4) solves the problem of the previous attention mechanism where the center node embedding is a prerequisite for attention score computing. This makes our graph attention mechanism applicable to both logic query answering and KG embedding training. As far as we know, it is the first graph attention mechanism applied on both tasks.

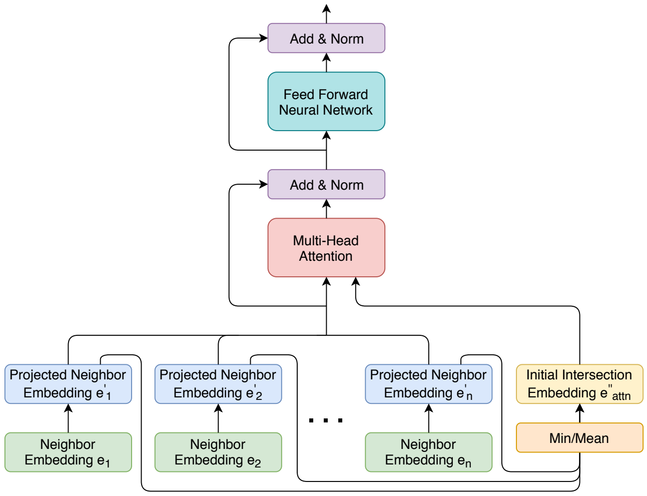

Add & Norm
Feed Forward Neural Network
Add & Norm
Multi-Head Attention
Projected Neighbor Embedding e'$_1$
Projected Neighbor Embedding e'$_2$
. . .
Projected Neighbor Embedding e'$_n$
Initial Intersection Embedding e''$_{attn}$
Neighbor Embedding e$_1$
Neighbor Embedding e$_2$
Neighbor Embedding e$_n$
Min/Mean

Figure 2: The attention-based geometric intersection operator - model architecture

3.2 Model Training

The projection operator and intersection operator constitute our attention-based logical query answering model. As for the model training, it has two training phases: the original KG training phase and the query-answer pair training phase.

3.2.1 Original KG Training Phase. In original KG training phase, we train those two geometric operators based on the local KG structure. Given a KG $\mathcal{G} = \langle \mathcal{V}, \mathcal{R} \rangle$, for every entity $e_i \in \mathcal{V}$, we use the geometric projection and intersection operator to compute a new embedding $\mathbf{e}_i''$ for entity e_i given its 1-degree neighborhood $N(e_i) = \{(r_{ui}, e_{ui}) | r_{ui}(e_{ui}, e_i) \in \mathcal{G}\} \cup \{(r_{oi}^{-1}, e_{oi}) | r_{oi}(e_i, e_{oi}) \in \mathcal{G}\}$ which is a sampled set of neighboring entity-relation pairs with size n. Here, $\mathcal{I}()$ indicates either $\mathcal{I}_{GQE}()$ (baseline model) or $\mathcal{I}_{CGA}()$ (proposed model).

$$\mathbf{e}_i'' = \mathbf{H}_{KG}(e_i) = \mathcal{I}(\{\mathcal{P}(e_{ci}, r_{ci}) | (r_{ci}, e_{ci}) \in N(e_i)\}) \tag{9}$$

Let $\mathbf{e}_i$ indicates the true entity embedding for e_i and $\mathbf{e}_i^-$ indicates the embedding of a negative sample $e_i^- \in Neg(e_i)$, where $Neg(e_i)$ is the negative sample set for e_i. The loss function for this KG training phase is a max-margin loss:

$$\mathcal{L}_{KG} = \sum_{e_i \in \mathcal{V}} \sum_{e_i^- \in Neg(e_i)} max(0, \Delta - \Phi(\mathbf{H}_{KG}(e_i), \mathbf{e}_i) + \Phi(\mathbf{H}_{KG}(e_i), \mathbf{e}_i^-)) \tag{10}$$

Here Δ is margin and $\Phi()$ denote the cosine similarity function:

$$\Phi(\mathbf{q}, \mathbf{a}) = \frac{\mathbf{q} \cdot \mathbf{a}}{\| \mathbf{q} \| \| \mathbf{a} \|} \tag{11}$$

3.2.2 Logical Query-Answer Pair Training Phase. In this training phase, we first sample Q different conjunctive graph query (logical query)-answer pairs $S = \{(q_i, a_i)\}$ from the original KG by sampling entities at each node in the conjunctive query structure according to the topological order (See Hamilton et al. [5]). Then for each conjunctive graph query q_i with one or multiple anchor nodes $\{e_{i1}, e_{i2}, .., e_{in}\}$, we compute the embedding for its target variable node $V_{i?}$, denote as $\mathbf{q}_i$, based on two proposed geometric operators (See Algorithm 1 in Hamilton et al. [5] for a detailed explanation). We denote the embedding for the correct answer entity as $\mathbf{a}_i$ and the embedding for the negative answer as $\mathbf{a}_i^-$ where $a_i^- \in Neg(q_i, a_i)$. The loss function for this query-answering pair train phase is:

$$\mathcal{L}_{QA} = \sum_{(q_i, a_i) \in S} \sum_{a_i^- \in Neg(q_i, a_i)} max(0, \Delta - \Phi(\mathbf{q}_i, \mathbf{a}_i) + \Phi(\mathbf{q}_i, \mathbf{a}_i^-)) \tag{12}$$

3.2.3 Negative Sampling. As for negative sampling method, we adopt two methods: 1) *negative sampling*: $Neg(e_i)$ is a fixed-size set of entities which have the same entity type as e_i except e_i itself; 2) *hard negative sampling*: $Neg(e_i)$ is a fixed-size set of entities which satisfy some of the entity-relation pairs in $N(e_i)$ but not all of them.

3.2.4 Full Model Training. The loss function for the whole model training is the combination of these two training phases:

$$\mathcal{L} = \mathcal{L}_{KG} + \mathcal{L}_{QA} \tag{13}$$

While Hamilton et al. [5] trains the model only using logical query-answer pair training phase and Equation 12 as the loss function. We generalize their approach by adding the KG training phase to better incorporate the KG structure into the training.

4 EXPERIMENT

We carried out empirical study following the experiment protocol of Hamilton et al. [5]. To properly test all models' ability to reason with larger knowledge graph of many relations, we constructed two datasets from publicly available *DBpedia* and *Wikidata*.

4.1 Datasets

Hamilton et al. [5] conducted logic query answering evaluation with Biological interaction and Reddits videogame datasets[3]. However, the reddit dataset is not made publicly available. The Bio interaction dataset has some issue of their logic query generation process[4]. Therefore, we regenerate the train/valid/test queries from the Bio KG. Furthermore, the Bio interaction dataset has only 46 relation types which is very simple compared to many widely used knowledge graphs such as *DBpedia* and *Wikidata*. Therefore we construct two more datasets (*DB18*and *WikiGeo19*) with larger graphs and more relations based on *DBpedia* and *Wikidata* [5].

Both datasets are constructed in a similar manner as [5]:

(1) First collect a set of *seed entities*;
(2) Use these seed entities to get their 1-degree and 2-degree object property triples;
(3) Delete the entities and their associated triples with node degree less than a threshold η;
(4) Split the triple set into training, validation, and testing set and make sure that every entity and relation in the validation and testing dataset will appear in training dataset. The training/validation/testing split ratio is 90%/1%/9%;
(5) Sample the training queries from the training KG[6].

[3] https://github.com/williamleif/graphqembed

[4] Hamilton et al. [5] sample the training queries from the whole KG rather than the training KG, which makes all the triples in the KG known to the model and makes the tasks simpler than realistic test situations.

[5] The code and both datasets are available at https://github.com/gengchenmai/Attention_GraphQA.

[6] We modify the query generation code provided by Hamilton et al. [5]

Table 1: Statistics for Bio, ***DB18*** and ***WikiGeo19*** (Section 4.1). "NUM/QT" indicates the number of QA pairs per query type.

	Bio			DB18			WikiGeo19		
	Training	Validation	Testing	Training	Validation	Testing	Training	Validation	Testing
# of Triples	3,258,473	20,114	181,028	122,243	1,358	12,224	170,409	1,893	17,041
# of Entities	162,622	-	-	21,953	-	-	18,782	-	-
# of Relations	46	-	-	175	-	-	192	-	-
# of Sampled 2-edge QA Pairs	1M	1k/QT	10k/QT	1M	1k/QT	10k/QT	1M	1k/QT	10k/QT
# of Sampled 3-edge QA Pairs	1M	1k/QT	10k/QT	1M	1k/QT	10k/QT	1M	1k/QT	10k/QT

For ***DB18*** the seed entities are all geographic entities directly linked to `dbr:California` via `dbo:isPartOf` with type (`rdf:type`) `dbo:City`. There are 462 seed entities in total. In Step 2, we filter out triples with no `dbo:` prefixed properties. The threshold η is set up to be 10. For ***WikiGeo19*** the seed entities are the largest cities in each state of the United States[7]. The threshold η is 20 which is a relatively small value compare to η=100 for the widely used FB15K and WN18 dataset. Statistic for these 3 datasets are shown in Table 1. Given that the widely used KG completion dataset FB15K and WN18 have 15K and 41K triples, *DB18* and *WikiGeo19* are rather large in size (120K and 170K triples). Note that for each triple $r(s, o)$ in training/validation/testing dataset, we also add its inverse relation $r^{-1}(o, s)$ to the corresponding dataset and *the geometric projection operator will learn two separated projection matrices* $\mathbf{R}_r$ $\mathbf{R}_{r^{-1}}$ *for each relation.* The training triples constitute the training KG. Note that both GQE and CGA require to know the unique type for each entity. However, entities in *DBpedia* and *Wikidata* have multiple types (`rdf:type`). As for *DB18*, we utilize the level-1 classes in *DBpedia* ontology and classify each entity to these level-1 classes based on the `rdfs:subClassOf` relationships. For *WikiGeo19*, we simply annotate each entity with class *Entity*.

4.2 Training Details

As we discussed in Section 3.2, we train our CGA model based on two training phases. In the original KG training phase, we adopt an minibatch training strategy. In order to speed up the model training process, we sample the neighborhood for each entity with different neighborhood sample size ($n = 4, 5, 6, 7$) in the training KG beforehand. We split these sampled node-neighborhood pairs by their neighborhood sample size n in order to do minibatch training.

As for the logical query-answer pair training phase, we adopt the same query-answer pair sampling strategy as Hamilton et al. [5]. We consider 7 different conjunctive graph query structures shown in Figure 3c. As for the 4 query structures with intersection pattern, we apply hard negative sampling (see Section 3.2.3) and indicate them as 4 separate query types. In total, we have 11 query types. All training (validation/testing) triples are utilized as 1-edge conjunctive graph queries for model training (evaluation). As for 2-edge and 3-edge queries, the number for sampled queries for training/validation/testing are shown in Table 1. Note that all training queries are sampled from the training KG. All validation and testing queries are sampled from the whole KG and we make sure these queries cannot be directly answered based on the training KG (unanswerable queries [13]). To ensure these queries are truly unanswerable, the matched triple patterns of these queries should contain at least one triple in the testing/validation triple set.

4.3 Baselines

We use 6 different models as baselines: two models with the billinear projection operator $\mathbf{e}_i' = \mathcal{P}(e_i, r)$ and the element-wise mean or min as the simple intersection operator: **Billinear[mean_simple]**, **Billinear[min_simple]**; two models with the TransE based projection operator and the GQE version of geometric intersection operator: **TransE[mean]**, **TransE[min]**; and two GQE models [5]: **GQE[mean]**, **GQE[min]**. Since $\Psi()$ can be element-wise mean or min, we differentiate them using **[mean]** and **[min]**. Note that all of these 6 baseline models only use the logical query-answer pair training phase (see Section 3.2.2) to train the model. As for model with billinear projection operator, based on multiple experiments, we find that the model with element-wise min consistently outperforms the model with element-wise mean. Hence for our model, we use element-wise min for $\Psi()$.

4.4 Results

We first test the effect of the origin KG training on the model performance without the attention mechanism called **GQE+KG[min]** here. Then we test the models with different numbers of attention heads with the added original KG training phase which are indicated as **CGA+KG+x[min]**, where **x** represents the number of attention heads (can be 1, 4, 8).

Table 2 shows the evaluation results of the baseline models as well as different variations of our models on the test queries. We use the ROC AUC score and average percentile rank (APR) as two evaluation metrics. All evaluation results are macro-averaged across queries with different DAG structures (Figure 3c).

(1) All 3 variations of CGA consistently outperform baseline models with fair margins which indicates the effectiveness of contextual attention. The advantage is more obvious in query types with hard negative queries.
(2) Comparing **GQE+KG[min]** with other baseline models we can see that adding the original KG training phase in the model training process improves the model performance. This shows that the structure information of the original KG is very critical for knowledge graph embedding model training even if the task is not link prediction.
(3) Adding the attention mechanism further improves the model performance. This indicates the importance of considering the unequal contribution of the neighboring nodes to the center node embedding prediction.

[7] https://www.infoplease.com/us/states/state-capitals-and-largest-cities

Table 2: Macro-average AUC and APR over test queries with different DAG structures are used to evaluate the performance. *All* and *H-Neg*. denote macro-averaged across all query types and query types with hard negative sampling (see Section 3.2.3).

Dataset	Bio				DB18				WikiGeo19			
Metric	AUC		APR		AUC		APR		AUC		APR	
	All	H-Neg	All	H-Neg	All	H-Neg	All	H-Neg	All	H-Neg	All	H-Neg
Billinear[mean_simple]	81.65	67.26	82.39	70.07	82.85	64.44	85.57	71.72	81.82	60.64	82.35	64.22
Billinear[min_simple]	82.52	69.06	83.65	72.7	82.96	64.66	86.22	73.19	82.08	61.25	82.84	64.99
TransE[mean]	80.64	73.75	81.37	76.09	82.76	65.74	85.45	72.11	80.56	65.21	81.98	68.12
TransE[min]	80.26	72.71	80.97	75.03	81.77	63.95	84.42	70.06	80.22	64.57	81.51	67.14
GQE[mean]	83.4	71.76	83.82	73.41	83.38	65.82	85.63	71.77	83.1	63.51	83.81	66.98
GQE[min]	83.12	70.88	83.59	73.38	83.47	66.25	86.09	73.19	83.26	63.8	84.3	67.95
GQE+KG[min]	83.69	72.23	84.07	74.3	84.23	68.06	86.32	73.49	83.66	64.48	84.73	68.51
CGA+KG+1[min]	84.57	74.87	85.18	77.11	84.31	67.72	87.06	74.94	83.91	64.83	85.03	69
CGA+KG+4[min]	**85.13**	**76.12**	85.46	**77.8**	84.46	67.88	87.05	74.66	83.96	64.96	85.36	69.64
CGA+KG+8[min]	85.04	76.05	**85.5**	77.76	**84.67**	**68.56**	**87.29**	**75.23**	**84.15**	**65.23**	**85.69**	**70.28**
Relative Δ over GQE	2.31	**7.29**	2.28	**5.97**	1.44	**3.49**	1.39	**2.79**	1.07	**2.24**	1.65	**3.43**

(4) Multi-head attention models outperforms single-head models which is consistent with the result from GAT [20].

(5) Theoretically, $\mathcal{I}_{GQE}()$ has $2Ld^2 = Ld(2d)$ learnable parameters while $\mathcal{I}_{CGA}()$ has $Ld^2 + 2KLd + Ld = Ld(d + 2K + 1)$ parameters where L is the total number of entity types in a KG. Since usually $K \ll d$, *our model has fewer parameters than GQE while achieves better performance.*

(6) CGA shows strong advantages over baseline models especially on query types with hard negative sampling (e.g., 7.3% relative AUC improvement over GQE on Bio dataset[8]).

All models shown in Table 2 are implemented in PyTorch based on the official code[9] of Hamilton et al. [5]. The hyper-parameters for the baseline models GQE are tuned using grid search and the best ones are selected. Then we follow the practice of Hamilton et al. [5] and used the same hyper-parameter settings for our CGA models: 128 for embedding dimension d, 0.001 for learning rate, 512 for batch size. We use Adam optimizer for model optimization.

The overall delta of CGA over GQE reported in Tab. 2 is similar in magnitude to the delta over baseline reported in Hamilton et al. [5]. This is because CGA will significantly outperform GQE in query types with intersection structures, e.g., the 9th query type in Fig. 3c, but perform on par in query types which do not contain intersection, e.g. the 1st query type in Fig. 3c. Macro-average computation over all query types makes the improvement less obvious. In order to compare the performance of different models on different query structures (different query types), we show the individual AUC and APR scores on each query type in three datasets for all models (See Figure 3a, 3b, 3c, 3d, 3e, and 3f). To highlight the difference, we subtract the minimum score from the other scores in each figure. We can see that our model consistently outperforms the baseline models in almost all query types on all datasets except for the sixth and tenth query type (see Figure 3) which correspond to the same query structure *3-inter_chain*. In both these two query types, **GQE+KG[min]** has the best performance. The advantage of our attention-based models is more obvious for query types with hard negative sampling strategy. For example, as for the 9th query type (Hard-3-inter) in Fig. 3d, **CGA+KG+8[min]** has **5.8%** and **6.5%** relative APR improvement (**5.9%** and **5.1%** relative AUC improvement) over GQE[min] on *DB18* and *WikiGeo19*. Note that this query type has the largest number of neighboring nodes (3 nodes) which shows that our attention mechanism becomes more effective when a query type contains more neighboring nodes in an intersection structure. This indicates that the attention mechanism as well as the original KG training phase are effective in discriminating the correct answer from *misleading* answers.

5 CONCLUSION

In this work we propose an end-to-end attention-based logical query answering model called contextual graph attention model (CGA) which can answer complex conjunctive graph queries based on two geometric operators: the projection operator and the intersection operator. We utilized multi-head attention mechanism in the geometric intersection operator to automatically learn different weights for different query paths. The original knowledge graph structure as well as the sampled query-answer pairs are used jointly for model training. We utilized three datasets (Bio, *DB18*, and *WikiGeo19*) to evaluate the performance of the proposed model against the baseline. The results show that our attention-based models (which are trained additionally on KG structure) outperform the baseline models (particularly on the hard negatives) despite using less parameters. The current model is utilized in a transductive setup. In the future, we want to explore ways to use our model in a inductive learning setup. Additionally, conjunctive graph queries are a subset of SPARQL queries which do not allow disjunction, negation, nor filters. They also require the predicates in all query patterns to be known. In the future, we plan to investigate models that can relax these restrictions.

REFERENCES

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

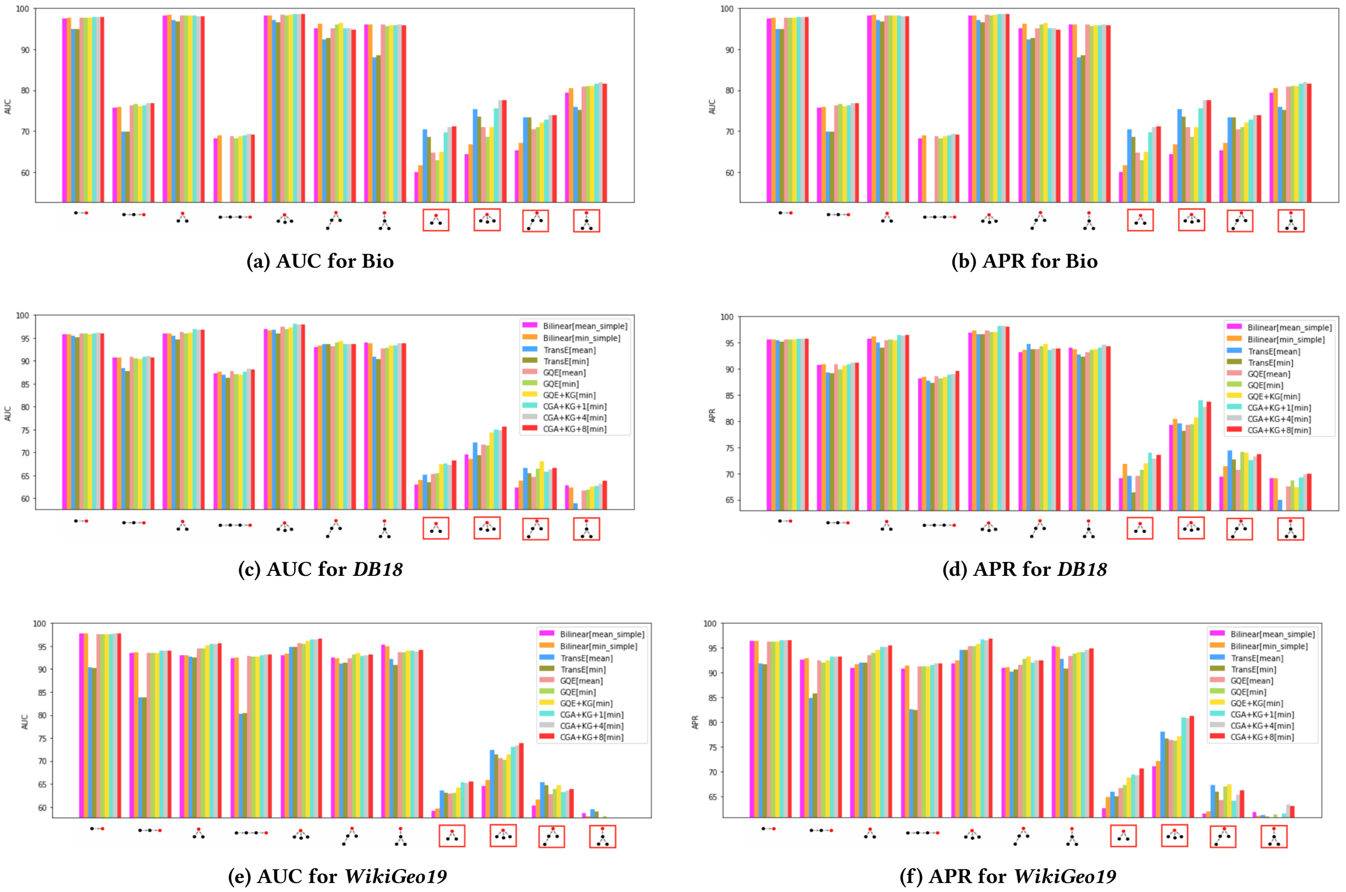
[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR 2015*.

[3] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *EMNLP*. 1533–1544.

[4] 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*. 2787–2795.

[8] Note that since we regenerate queries for Bio dataset, the GQE performance is lower than the reported performance in Hamilton et al. [5] which is understandable.

[9] https://github.com/williamleif/graphqembed



(a) AUC for Bio

(b) APR for Bio

(c) AUC for *DB18*

(d) APR for *DB18*

(e) AUC for *WikiGeo19*

(f) APR for *WikiGeo19*

Figure 3: Individual AUC and APR scores for different models per query type. Red boxes denote query types with hard negative sampling strategy

[5] Will Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. 2018. Embedding logical queries on knowledge graphs. In *Advances in Neural Information Processing Systems*. 2030–2041.

[6] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[8] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Knowledge graph embedding via dynamic mapping matrix. In *ACL*, Vol. 1. 687–696.

[9] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR 2017*.

[10] Chen Liang, Jonathan Berant, Quoc Le, Kenneth D Forbus, and Ni Lao. 2017. Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision. In *ACL*, Vol. 1. 23–33.

[11] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning Entity and Relation Embeddings for Knowledge Graph Completion.. In *AAAI*, Vol. 15. 2181–2187.

[12] Gengchen Mai, Krzysztof Janowicz, and Bo Yan. 2018. Support and Centrality: Learning Weights for Knowledge Graph Embedding Models. In *EKAW*. Springer, 212–227.

[13] Gengchen Mai, Bo Yan, Krzysztof Janowicz, and Rui Zhu. 2019. Relaxing Unanswerable Geographic Questions Using A Spatially Explicit Knowledge Graph Embedding Model. In *Proceedings of 22nd AGILE International Conference on Geographic Information Science*.

[14] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. 2016. A review of relational machine learning for knowledge graphs. *Proc. IEEE* 104, 1 (2016), 11–33.

[15] Maximilian Nickel, Lorenzo Rosasco, Tomaso A Poggio, et al. 2016. Holographic Embeddings of Knowledge Graphs.. In *AAAI*. 1955–1961.

[16] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2012. Factorizing yago: scalable machine learning for linked data. In *WWW*. ACM, 271–280.

[17] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*. Springer, 593–607.

[18] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. 2013. Reasoning with neural tensor networks for knowledge base completion. In *Advances in neural information processing systems*. 926–934.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*. 5998–6008.

[20] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR 2018*.

[21] Meng Wang, Ruijie Wang, Jun Liu, Yihe Chen, Lei Zhang, and Guilin Qi. 2018. Towards Empty Answers in SPARQL: Approximating Querying with RDF Embedding. In *International Semantic Web Conference*. Springer, 513–529.

[22] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. 2017. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering* 29, 12 (2017), 2724–2743.

[23] Jiacheng Xu, Kan Chen, Xipeng Qiu, and Xuanjing Huang. 2016. Knowledge graph representation with jointly structural and textual encoding. *arXiv preprint arXiv:1611.08661* (2016).

[24] 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*.

[25] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. 2017. Deep sets. In *Advances in neural information processing systems*. 3391–3401.