Modeling Semantics with Gated Graph Neural Networks for Knowledge Base Question Answering

Daniil Sorokin and **Iryna Gurevych**
Ubiquitous Knowledge Processing Lab (UKP) and Research Training Group AIPHES
Department of Computer Science, Technische Universität Darmstadt
www.informatik.tu-darmstadt.de/ukp/

Abstract

The most approaches to Knowledge Base Question Answering are based on semantic parsing. In this paper, we address the problem of learning vector representations for complex semantic parses that consist of multiple entities and relations. Previous work largely focused on selecting the correct semantic relations for a question and disregarded the structure of the semantic parse: the connections between entities and the directions of the relations. We propose to use Gated Graph Neural Networks to encode the graph structure of the semantic parse. We show on two data sets that the graph networks outperform all baseline models that do not explicitly model the structure. The error analysis confirms that our approach can successfully process complex semantic parses.

1 Introduction

Knowledge base question answering (QA) is an important natural language processing problem. Given a natural language question, the task is to find a set of entities in a knowledge base (KB) that constitutes the answer. For example, for a question "What is Princess Leia's home planet?" the answer, "Alderaan", could be retrieved from a general-purpose KB, such as Wikidata[1]. A successful KB QA system would ultimately provide a universally accessible natural language interface to factual knowledge (Liang, 2016).

QA requires precise modeling of the question semantics through the entities and relations available in the KB in order to retrieve the correct answer. Figure 1 shows how the above example question could be modeled using Wikidata. The depicted graph structure consists of entities and relations from the KB and the special q-node. Any entity in the KB that can take the place of the q-node will be a part of the answer.

In this paper, we describe a semantic parsing approach to the problem of KB QA. That is, for each input question, we construct an explicit structural semantic parse (*semantic graph*), as in Figure 1. Semantic parses can be deterministically converted to a query to extract the answers from the KB. Similar graphical representations were used in previous work (Yih et al., 2015; Reddy et al., 2016; Bao et al., 2016).

However, the modern semantic parsing approaches usually focus either on the syntactic analysis of the input question (Reddy et al., 2016) or on detecting individual KB relations (Yu et al., 2017), whereas *the structure of the semantic parse* is ignored or only approximately modeled. Reddy et al. (2016) use the syntactic structure of the question to build all possible semantic parses and then apply a linear model with manually defined features to choose the correct parse. A subset of their features encodes basic information about the graph structure of the semantic parse (e.g. number of nodes). The state-of-the-art approach of Yih et al. (2015) and Bao et al. (2016) restricts all semantic parses to a single core relation and a small set of constraints that can be added to it. Their system uses manual features for the constraints and a similarity score between the core relation and the question to model the semantic parses.

The abovementioned systems were evaluated on the WebQuestions data set (Berant et al., 2013). Figure 2 plots results for the state-of-the-art systems by the number of relations that needs to be identified to get the correct answer to a question.[2] For example, the question in Figure 1 requires two relations

[1] https://www.wikidata.org/

[2] We include results for the most recent systems that have published the output on the WebQuestions data set. We compute the number of needed relations from the manual semantic parses provided by Yih et al. (2016).

Proceedings of the 27th International Conference on Computational Linguistics, pages 3306–3317
Santa Fe, New Mexico, USA, August 20-26, 2018.

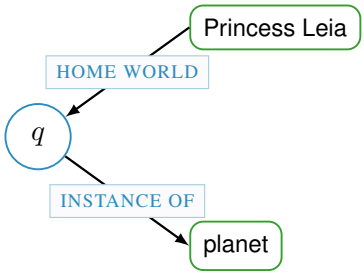

Figure 1: Semantic graph for an example question "What is Princess Leia's home planet?"

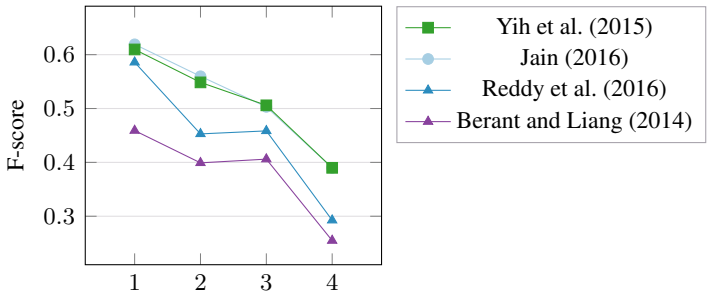

Figure 2: F-score QA results from previous work by the number of relations needed to find the correct answer

to find the answer. It can be clearly seen in Figure 2 that the lack of structure modeling in the modern approaches results in a worse performance on more complex questions that require more than one relation.

We claim that one needs to *explicitly model the semantic structure* to be able to find the correct semantic parse for complex questions. In this paper, we address this gap and investigate ways to encode the structure of a semantic parse and to improve the performance for more complex questions. In particular, we adapt Gated Graph Neural Networks (GGNNs), described in Li et al. (2016), to process and score semantic parses. To verify that GGNNs indeed offer an improvement, we construct a set of baselines based on the previous work that we train and evaluate in the same controlled QA environment. Throughout the experiments, we use the Wikidata open-domain KB (Vrandečić and Krötzsch, 2014) to construct semantic parses and retrieve the answers.[3]

Contributions To summarize, the main contributions of our work are:

(i) Our analysis shows that the current solutions for KB QA do not perform well on complex questions;

(ii) We apply Gated Graph Neural Networks on directed graphs with labeled edges and adapt them to handle a large set of possible entities and relations types from the KB. To the best of our knowledge, we are the first to use GGNNs for semantic parsing and KB QA;

(iii) Our Gated Graph Neural Network implementation for semantic parsing improves performance on complex questions in comparison to strong baselines. The results show a 27.4% improvement of the F-score against the best non-graph model.

Code and data sets Our system can be used with a pre-trained model to answer factual questions with Wikidata or trained anew on any data set that has question-answer pairs. The complete code, the scripts that produce the evaluation data and the installation instructions can be found here: `https://github.com/UKPLab/coling2018-graph-neural-networks-question-answering`.

2 Semantic parsing

2.1 Semantic graphs

We employ structural semantic representations in the form of graphs to encode the meaning of a question. Our *semantic graphs* consist of a question variable node (q), Wikidata entities (Taylor Swift), relation types from Wikidata (PERFORMER) and constraints (see Figure 3 for an example graph with a constraint). The q-node is always present and denotes the answer to the question. That is, all entities from the KB that can take its place so that all relations and constraints hold, constitute the answer to the question.

We write down a graph as a set of edges $\mathcal{E}$. Each edge links the q-node to one or two Wikidata entities (binary or ternary edge). The edge set $\mathcal{E}$ is defined via Wikidata relations and entities. Formally, Wikidata can be described as a very large graph $W = (E, R, I)$, where E is a set of entities, R is a set of binary

[3] At the moment, Wikidata contains more than 40 million entities and 350 million relation instances: `https://www.wikidata.org/wiki/Special:Statistics`

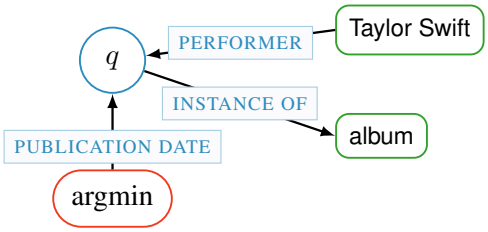

Figure 3: A semantic graph for an example question "What was the first Taylor Swift album?"

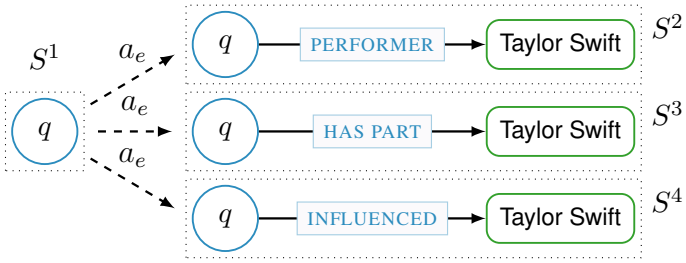

Figure 4: A single graph generation step: applying the add entity action a_e on an empty graph

relation types and I is a collection of relation instances encoded as $r(e_1, e_2),\ \ r \in R,\ \ e_1, e_2 \in E$ (e. g. CAPITAL(Hawaii, Honolulu)). Ternary relation instances in Wikidata are stored as a main relation triple and an attached modifer which itself is also a binary relation: $r_2\big(r_1(e_1, e_2), e_3\big),\ \ r_1, r_2 \in R,\ \ e_1, e_2, e_3 \in E$ (e.g. CHARACTER ROLE$\big($CAST MEMBER(Star Wars, Carrie Fisher), Princess Leia$\big)$. Then, the edge set $\mathcal{E}$ of a semantic graph consists of binary and ternary edges that correspond to Wikidata relation instances with q in place of one of the relation arguments. For temporal relations, the second argument can be either an argmax or an argmin constraint. This means that only entities that have the maximum or the minimum value of that relation are considered for the answer. This definition represents a subset of first-order logic semantic parses restricted to conjunctions of predicates. The graphs are also isomorphic to SPARQL queries that can be used to evaluate a graph against the KB.

Our semantic graphs were inspired by the query graphs of Yih et al. (2015) and their extension in Bao et al. (2016), the key difference being that we do not differentiate between the core relation and modifiers, but rather allow graphs that have multiple independent relations. We also allow relations to attach to other nodes rather than the q-node, enabling longer relation paths between a known entity and the q-node. Thus, the semantic graphs defined here are a superset of the query graphs in Yih et al. (2015) and allow us to model more complex meanings. Consequently, they also correspond to the formalized representations used by Reddy et al. (2014) and the simple λ-DCS (Berant et al., 2013), since those were the foundation for the query graphs (Yih et al., 2015).

2.2 Semantic graph construction

Yih et al. (2015) defined a step-by-step staged graph generation that does not need full syntactic parses and was also adopted in subsequent publications (Bao et al., 2016; Peng et al., 2017). We use the same procedure as Yih et al. (2015) to construct semantic graphs with a set of modifications that allow us to build more expressive and complex graphs.

We define a set of states and a set of actions that can be applied at each state to extend the current semantic graph. A state is a tuple of a graph and a set of free entities: $\mathcal{S} = (\mathcal{E}, \mathcal{F})$, where the graph is $\mathcal{E}$, as defined above, and $\mathcal{F} = \{e | e \in E\}$. The set of free entities $\mathcal{F}$ is the entities that were identified in the question but were not yet added to the graph.[4] The first state is a tuple of an empty graph with no edges and a set of all entities identified in the question $\mathcal{S}^1 = (\{\}, \mathcal{F})$.

Let $\mathcal{A} = \{a_e, a_c, a_m\}$ be the set of actions that can be taken to extend the graph at the current state. The action a_e (*add entity*) pops a free entity e from the set $\mathcal{F}$ at the current state $\mathcal{S}^t$. Then it queries the KB and retrieves the set of available relation types R_e for the entity e. For each relation type $r \in R_e$, it creates a new copy of the graph and adds a new directed edge between the q-node and e with the relation type r: $a_e(\mathcal{E}, \mathcal{F}) = \{\mathcal{E} \cup r(q, e), \mathcal{F} - e \mid e \in \mathcal{F}, r \in R_e\}$. Contrary to Yih et al. (2015) and Bao et al. (2016), we allow two-step paths between the q-node and the entity e: $r(q, d) \wedge r(d, e)$.[5]

The action a_c (*add constraint*) pops a free entity e and follows the same procedure as a_e, but instead of adding a new edge, it adds a modifier to the last added edge of the graph, thus creating a ternary edge: $a_c(\mathcal{E}, \mathcal{F}) = \{\mathcal{E} \cup r_2(r_1(q, e_1), e_2), \mathcal{F} - e_2 \mid e_2 \in \mathcal{F}, r_2 \in R_{e_2}, r_1(q, e_1) \in \mathcal{E}\}$. Finally, the action a_m

[4] We use an off-the-shelf entity linker to identify and disambiguate entity mentions in the question (see Section 4.4).

[5] Freebase relation instances always have an intermediate node and a two-step path corresponds to a single step in Wikidata.

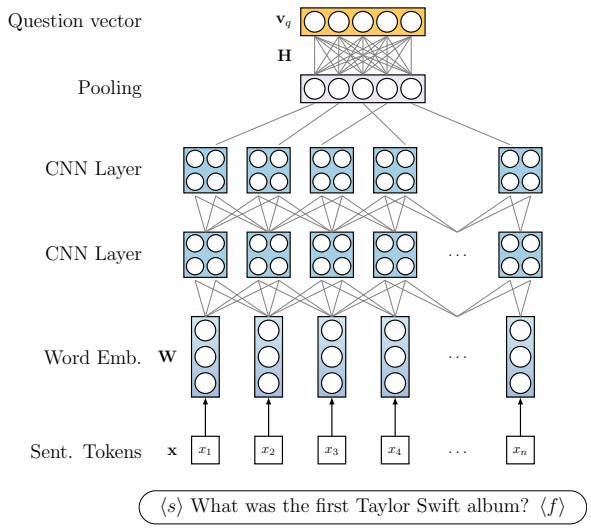

Figure 5: Deep Convolutional Neural Networks (DCNN) architecture, here used to process an example question

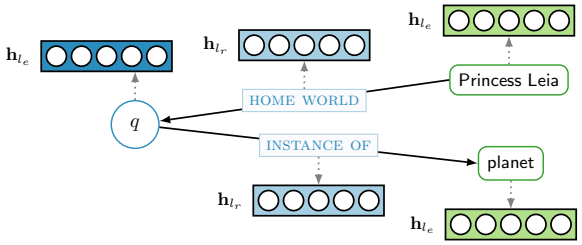

Figure 6: Encoding a graph into initial hidden states

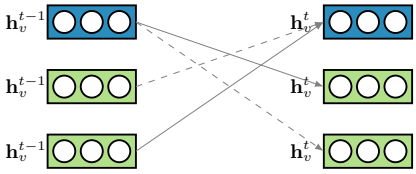

Figure 7: Unrolled recurrence for one timestep, solid lines show updates along the direction of the relation in the graph and the dashed lines in the opposite direction

(*add argmax/argmin*) adds a new edge with either argmax or argmin sorting constraint to the semantic graph: $a_m(\mathcal{E}, \mathcal{F}) = \{\mathcal{E} \cup r(q, \text{argmax}), \mathcal{F}; \mathcal{E} \cup r(q, \text{argmin}), \mathcal{F} \mid r \in R_d\}$, where R_d is a set of KB relation types that allow dates as values. Our semantic graph construction process allows to effectively search the space of possible graphs for a given question through an iterative application of the defined actions $\mathcal{A}$ on the last state $\mathcal{S}^t$ (see, for example, Figure 4).

3 Representation learning

We follow the state-of-the-art approaches for QA (Yih et al., 2015; Dong et al., 2015; Bao et al., 2016) and learn representations for the question and every possible semantic graph. Then we use a simple reward function γ to judge if a semantic graph is a correct semantic parse of the input question. Below we describe the architectures that we use to learn the representations for questions and graphs.

3.1 Deep Convolutional Networks

We use Deep Convolutional Neural Networks (DCNN) to learn a representation for a question. DCNNs have been proven effective for constructing sentence-level representations on a variety of NLP tasks, including named entity recognition (Strubell et al., 2017) and KB QA (Yih et al., 2015; Dong et al., 2015).

The DCNN architecture is depicted in Figure 5, where it is used to map an input question to a fixed-size vector representation. The input question is first tokenized and the special start and end tokens are added to the sequence: $\mathbf{x} = \{\langle s \rangle, x_1, x_2 \ldots x_n, \langle f \rangle\}$. Next, we map the tokens in $\mathbf{x}$ to d_w-dimensional pre-trained word embeddings, using a matrix $\mathbf{W}_{glove} \in \mathbb{R}^{|V_w| \times d_w}$, where $|V_w|$ is the size of the vocabulary. We use 50-dimensional GloVe embeddings that were trained on a 6 billion corpus (Pennington et al., 2014). The sequence of word embeddings is further processed by an array of CNN layers. We apply a pooling operation after the last CNN layer and transform the output with a fully connected layer $\mathbf{H}$ and a ReLU non-linearity. We take the resulting vector $\mathbf{v}_q$ as the representation of the question.

3.2 Gated Graph Neural Networks

Gate Graph Neural Networks (GGNN) process graphs by iteratively updating representations of graph nodes based on the neighboring nodes and relations (Li et al., 2016). We adopt GGNNs for semantic parsing to learn a vector representation of a semantic graph. Li et al. (2016) give a formulation of GGNNs for graphs with labeled nodes and typed directed edges. We extend their formulation to include labeled edges. To the best of our knowledge, we are the first to apply GGNN to semantic parsing and KB QA.

For a graph $\mathcal{E}$, we extract a set of all entities $\mathcal{V}$ in the graph and a set of all relation types $\mathcal{R}$ of its edges. We use labels from Wikidata to compute vectors for entities and relation types (Figure 6). This enables us to directly incorporate the information on millions of entities and hundreds of relation types from the KB. For an entity $e \in \mathcal{V}$ or a relation type $r \in \mathcal{R}$ we retrieve the label and tokenize it: $\mathbf{l} = \{l_1, l_2 \dots l_n\}$. Then we map each token in $\mathbf{l}_e$, $\mathbf{l}_r$ to a word embedding using the matrix $\mathbf{W}_{glove}$, sum them and process with a fully connected layer to get a single label vector: $\mathbf{h}_l = \tanh\big(\mathbf{W}_\mathbf{l}\big[\sum_{n=1}^{|\mathbf{l}|} w_n\big] + \mathbf{b}_\mathbf{l}\big)$. We initialize the hidden states for the graph nodes with the label vectors of the entities: $\mathbf{h}_v^{(1)} = \mathbf{h}_{l_e}$. We further transform the relation label vectors to get directional embeddings for relations types: $\mathbf{h}'_r = \mathbf{W}_{\rightarrow}\mathbf{h}_{l_r}$, $\mathbf{h}''_r = \mathbf{W}_{\leftarrow}\mathbf{h}_{l_r}$. Using the same word embeddings as an input to construct the question and relation representations has been shown successful in previous work (Jain, 2016).

Propagation Model GGNNs are a type of recurrent neural networks. The recurrence is unrolled for a fixed number of steps T and the gating mechanism works akin to Gated Recurrent Units (Cho et al., 2014). The propagation model for GGNN is defined as follows:

$$\begin{aligned}\mathbf{a}_v^{(t)} &= \mathbf{A}_{v:}^\top\big[\mathbf{h}_1^{(t-1)\top} \dots \mathbf{h}_{|\mathcal{V}|}^{(t-1)\top}\big] \\ &\quad + \mathbf{A'}_{r:}^\top\big[\mathbf{h}_1'^\top \dots \mathbf{h}_{|\mathcal{R}|}'^\top, \mathbf{h}_1''^\top \dots \mathbf{h}_{|\mathcal{R}|}''^\top\big] \end{aligned} \quad (1)$$

$$\mathbf{z}_v^t = \sigma\big(\mathbf{W}^z\mathbf{a}_v^{(t)} + \mathbf{U}^z\mathbf{h}_v^{(t-1)} + \mathbf{b}^z\big) \quad (2)$$

$$\mathbf{r}_v^t = \sigma\big(\mathbf{W}^r\mathbf{a}_v^{(t)} + \mathbf{U}^r\mathbf{h}_v^{(t-1)} + \mathbf{b}^r\big) \quad (3)$$

$$\widetilde{\mathbf{h}_v^{(t)}} = \tanh\big(\mathbf{W}\mathbf{a}_v^{(t)} + \mathbf{U}\big(\mathbf{r}_v^t \odot \mathbf{h}_v^{(t-1)}\big) + \mathbf{b}\big) \quad (4)$$

$$\mathbf{h}_v^{(t)} = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{t-1} + \mathbf{z}_v^t \odot \widetilde{\mathbf{h}_v^{(t)}}, \quad (5)$$

where σ is the logistic sigmoid function and $\odot$ is the element-wise multiplication. The matrix $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times 2|\mathcal{V}|}$ stores the structure of the graph: a row of the matrix $\mathbf{A}_{v:}$ records the edges between the node v and the other nodes in the graph (we differentiate between incoming and outgoing edges). The second matrix $\mathbf{A'} \in \mathbb{R}^{|\mathcal{V}|\times 2|\mathcal{R}|}$ stores the relation types of the incoming and outgoing edges.

The main difference from the model defined in Li et al. (2016) is that we compute activations $\mathbf{a}_v^{(t)}$ based on both the node hidden vectors $\mathbf{h}_v^{(t-1)}$ and relation hidden vectors $\mathbf{h}'_r$ (Eq 1). The $\mathbf{z}_v^t$ in Eq 2 and $\mathbf{r}_v^t$ in Eq 3 are update and reset gates, that incorporate information from nodes, relation types and from the previous step to update nodes' hidden states at each iteration. We do not make updates to the hidden vectors of the relations and use them only to pass the information to the nodes' hidden states.

Graph-level Output Vector We unroll the recurrence for $T = 5$ steps in the experiments (Figure 7). To produce a graph-level output vector, we take the hidden vector for the q-node at the last time step $t = T$ and transform it with a fully-connected layer and the ReLU non-linearity: $\mathbf{v}_g = \mathrm{ReLU}\big(\mathbf{W}\mathbf{h}_q^{(t)} + \mathbf{b}\big)$.

4 Experiments

4.1 Data

We use Wikidata to show experimentally that GGNNs are better at learning representations of semantic graphs than previous approaches. Hence, we choose two data sets that can be processed with Wikidata to compare the GGNNs to other models: WebQSP-WD and QALD-7.

WebQSP-WD We derive WebQSP-WD from WebQSP (Yih et al., 2016), which is a corrected version of the popular WebQuestions data set (Berant et al., 2013). WebQSP contains natural language questions, Freebase IDs of the correct answers and SPARQL queries to retrieve them that also use Freebase. Freebase was a common choice of a KB among the previous work, but was discontinued and is no longer up-to-date, including unavailability of APIs and new dumps. The questions in the data set were collected with the Google Suggest API and are thus more 'natural' than manually constructed questions. The answers were retrieved from Freebase with the help of crowd-sourcing. The data set contains both simple questions that can be answered with a single relation as well as complex questions that require multiple relations and constraints. It is a common benchmark for semantic parsers and information retrieval systems and was used in the most recent studies on KB QA. We automatically map Freebase IDs in the WebQSP train and test sets to Wikidata IDs and filter out questions which answers do not have the mapping. We designate

this version of the dataset WebQSP-WD. It is important to note that this does not ensure that a question is answerable with Wikidata as there still might be no relation paths in the KB that connect the entities in the question with the answer.

QALD-7 As the second data set, we use QALD-7 that was developed for Task 4 of the QALD-7 Shared Task, "English question answering over Wikidata" (Usbeck et al., 2017). The QALD-7 data set contains a small number of manually constructed complex questions that were specifically created to test system's ability to process questions with multiple entities and constraints. The data set uses Wikidata IDs for all annotations. We do not train on QALD-7 data set and use it solely for an out-of-domain evaluation. The data set statistics can be found in Table 1.

4.2 Models

We define five models for our experiments, including three baselines and two graph models. We use cosine similarity between the question representation and the semantic graph representation as a reward function that judges whether the semantic graph is the correct parse of the question: $\gamma = \cos(\mathbf{v_s}, \mathbf{v_g})$.

1. STAGG (re-implementation of Yih et al. (2015)) — We implement a model that uses a combination of a neural network and manual features suggested in Yih et al. (2015) and in the follow up work of Bao et al. (2016). The approach of Yih et al. (2015) performed best among the previous work on complex questions (see Figure 2). First, DCNN is used to produce a representation for the input question, as described in Section 3.1. Then, we replace the entity tokens in the input question with a special entity symbol $\langle e \rangle$ and apply DCNN on it again to get a second representation for the question. For each semantic graph, we take the label of the relation in the first edge (*core edge*), tokenize it and likewise apply DCNN on it to get a representation. We produce two representations for the core relation: one that includes the label of the attached entity and one that includes the entity symbol. The manual features include binary indicators for modifiers and constraints in the semantic graphs, as well as features for certain keywords in the question (see Yih et al. (2015) for a detailed description).

   The original system and the models of Yih et al. (2015) and of Bao et al. (2016) are not available and therefore we use our own implementation of their approach. There is a number of small difference with the original model: we use a deep CNN instead of a single CNN layer and train the DCNN together with the manual features, whereas Yih et al. (2015) pre-trained the CNN model on a separate corpus and used its output in the feature model.

2. Single Edge model — We use the DCNN to encode the question and the label of the first edge of a semantic graph. The rest of the information is ignored.

3. Pooled Edges model — We use the DCNN to encode the question and the label of each edge in the semantic graph. To get a fixed-vector representation of the graph, we apply a pooling operation over the representation of the individual edges. This model encodes all information about the semantic graphs, but disregards their structure.

4. Graph Neural Network (GNN) — To judge the effect of the gated graph neural architecture, we also include a model variant that does not use the gating mechanism and directly computes the hidden state as a combination of the activations (Eq 1) and the previous state.

5. Gated Graph Neural Network (GGNN) — We use the GGNN to process semantic parses, as described in Section 3.2. This model encodes all information from semantic graphs, including their structure, into a vector representation. To encode the question we use the same DCNN model (see Section 3.1).

The defined baselines use either manual features to capture the structure of the semantic graph (STAGG), a simple pooling mechanism (Pooled Edges) or disregard the structure completely (Single Edge). The two graph models (GNN and GGNN) make the full use of the graph structure of a semantic parse and encode it with graph neural networks. With this set-up, we are able to demonstrate what effect different levels of inclusion of the graph structure into the model have on the final performance for KB QA. We do

not include the published models of Berant et al. (2013) and Reddy et al. (2014) in the comparison since they were trained on Freebase and are not compatible with Wikidata. We use the more recent STAGG approach to position the graph models against the previous work and the feature-based models.

4.3 Training the model

To train the model, we need positive pairs of questions and semantic graphs. Since WebQSP does not contain semantic parses for Wikidata, we use weak supervision as suggested in Berant et al. (2013). Specifically, we follow Yih et al. (2015) and run our semantic graph construction procedure to create training instances (see Section 2.2). We use the state-of-the-art S-MART entity linking system for noisy data (Yang and Chang, 2015) to extract a set of entities $\mathcal{F}$ from each question.[6] Instead of scoring the semantic graphs with the model, we evaluate each graph against Wikidata and compare the extracted answers to the manual answers in the data set. The semantic graphs that result in a correct answer are stored as positive training instances and the rest of the graphs generated during the same process are used as negative instances. Due to the differences between Freebase and Wikidata, some question can not be answered exactly using Wikidata. We generate positive semantic graphs for 1945 questions out of 2880 (see Table 1) and put 628 as a development set aside.

Practical considerations At each training epoch, we take all positive semantic graphs and up to 100 negative graphs per question. We optimize the maximum margin loss function: $\mathcal{L} = \sum_{g \in C} \max\big(0, (m - \gamma(\mathbf{v}_q, \mathbf{v}_g^+) + \gamma(\mathbf{v}_q, \mathbf{v}_g^-))\big)$, where C is a set of semantic parses for the given question. From the loss function, we compute updates for the GGNN and the DCNN parts of the model.

All models are trained using the Adam optimizer (Kingma and Ba, 2014) with a batch size of 64. We use an early stopping criterion on the development data to determine the number of training epochs. The learning rate is fixed to 0.001 and the other optimization parameters are set as recommended in Kingma and Ba (2014): $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e-08$. We apply Dropout (Srivastava et al., 2014) at every fully-connected layer as well as on the embeddings layers. We determine the hyper-parameters, such as the size of the hidden layer, with the random search on the development set (see Table 2). On the 1945 training questions, the GGNN model has usually finished training in under two hours on a single GPU.

	WebQSP-WD (train)	WebQSP-WD (test)	QALD-7
questions	2880	1033	80
complex questions	419	293	42
avg. # of relations per question	1.35	1.39	2.13

Table 1: Dataset statistics

Parameter	Tested values	Selected
Hidden layer size	$[32, 64, 128, 256, 512]$	256
CNN filter size	$[32, 64, 128, 256, 512]$	256
Dropout ratio	$\mathcal{U}(0.0, 0.5)$	0.2
# of CNN layers	$[1, 2, 3]$	2
Pooling operation	$[\text{avg}, \text{sum}, \max]$	max

Table 2: Optimized hyper-parameter values

4.4 Inference

We take the steps described in Section 2.2 to construct possible semantic graphs for a question at inference time. For WebQSP-WD, we use the entities produced by S-MART (Yang and Chang, 2015) to start the graph construction. On QALD-7, we use the annotated entities provided by the Shared Task organizers.

Each question and semantic graph are encoded into fixed-size vector representations and the reward function γ is used to score the graphs. The highest scoring graph is used to retrieve the answers from Wikidata. Given the iterative nature of our semantic graph construction procedure, we adopt beam search to speed up the computation. We score the graphs after each step and select the top 10 to proceed.

[6] S-MART is not openly available, but the output on the WebQuestions dataset was made available by Yih et al. (2015): https://github.com/scottyih/STAGG

	P	R	F
STAGG	0.1911	0.2267	0.1828
Single Edge	0.2240	0.2713	0.2148
Pooled Edges	0.2094	0.2553	0.2032
GNN	0.2419	0.2890	0.2326
GGNN	**0.2686**	**0.3179**	**0.2588**

Table 3: Results on the WebQSP-WD test set

	P	R	F
STAGG	0.1934	0.2463	0.1861
Single Edge	0.2173	0.1963	0.1951
Pooled Edges	0.1904	0.1800	0.1605
GNN	0.1652	0.2072	0.1703
GGNN	**0.2176**	**0.2751**	**0.2131**

Table 4: Results on the QALD-7 data set

5 Results

We follow the previous work (Berant et al., 2013) and use precision, recall and F-score to evaluate the models. The measures are computed for each individual question and then macro-averaged. This ensures a fair evaluation, since a system might provide a partially correct answer that is nevertheless better than a complete miss. We compare the graph models to the baselines including the previous state-of-the-art STAGG architecture.[7]

5.1 WebQSP-WD

We compare the results on the WebQSP-WD data set in Table 3. As can be seen, the graph models outperform all other models across precision, recall and F-score, with GGNN showing the best overall result. We confirm thereby that the architecture that encodes the structure of a semantic parse has an advantage over other approaches. To validate the results, we have re-trained the model with different random seeds and observed little variance in the results (F-score, $\sigma = 0.0205$).

The STAGG architecture delivers the worst results in our experiments, the main reason being supposedly that the model had to rely on manually defined features that are less flexible. The Single Edge model outperforms the more complex Pooled Edges model by a noticeable margin. The Single Edge baseline prefers simple graphs that consist of a single edge which is a good strategy to achieve higher recall values.

Since our main goal is to produce better encoding of semantic graphs, we break down the performance of the evaluated models by the number of relations that are needed to find the correct answer.[8] In Figure 8, we see that for the STAGG and Single Edge baselines the performance on more complex questions drops compared to the results on simpler questions. The Pooled Edges model maintains a better performance across questions of different complexity, which shows the benefits of encoding all graph edges. Looking at Figure 8 we also get a further insight into the difference between the Single Edge and the Pooled Edges models. These two models achieve almost identical results on the simple questions, which is to be expected since these models are equivalent when the number of edges is 1. On other questions, the Single Edge baseline performs mostly under the Pooled Edges model, but significantly outperforms it on the questions that need 4 edges to get the correct answer.

We see that the GGNN model offers the best results both on simple and complex questions, as it effectively encodes the structure of semantic graphs. The performance of the model drops for the most complex questions (# of edges ≥ 4). That does not happen for the GNN model variant without the gating mechanism. We conjecture that this happens because GGNN has more parameters than GNN and therefore needs more data to learn. By looking at the model errors on the most complex questions, we could see that GGNN tends to incorrectly predict one of the relations in the graph, which results in a wrong answer. GNN, on the other hand, more often predicts less relations that are needed and therefore gets a non-zero score for a partially correct answer. For example for the question "Who is the prime minister of Spain 2011?", GNN predicts a graph of two relations INSTANCE OF(human) and HEAD OF GOVERNMENT which returns a list of all Spanish prime ministers. The complete correct graph would also include temporal constraints.

[7] Since we use Wikidata and WebQSP-WD, the values reported in this work are not directly comparable to those for Freebase.

[8] We use again the manually constructed queries provided by Yih et al. (2016) to estimate it.

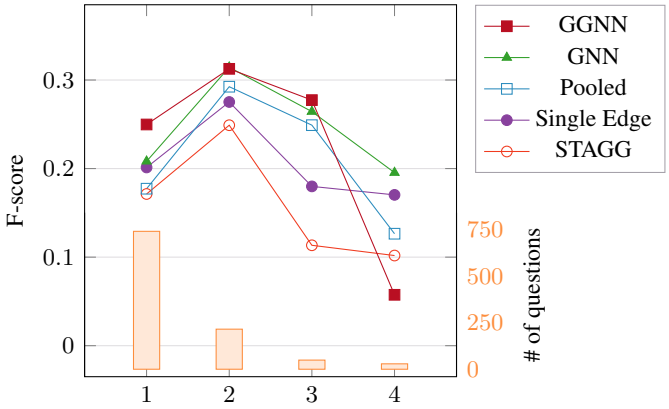

Figure 8: Evaluation results by # of relations needed to find the correct answer (bars show # of questions)

	Pooled	GGNN
F-score > 0.5	22.27%	28.37%
F-score > 0.0	29.82%	37.08%
Entity linker errors	6%	8%
No path to answer	14%	18%
Data set inconsistency	8%	14%
Wrong prediction	72%	60%
hit@10	35.62%	44.05%

Table 5: Manual error analysis results on WebQSP-WD

Notably, GGNN also has the best performance on the most simple semantic parses that only have one edge. In these cases, two nodes in the graph interact with each other through the single edge in both directions. The GGNN is better at capturing this interaction than other models.

5.2 QALD-7

Next, we examine the out-of-domain results on the QALD-7 data set for the five models (see Table 4). The difference in performance is less prominent on this data set, but we can observe the same trends. The Single Edge model outperforms both the STAGG and Pooled Edges baselines. The GGNN delivers the best performance, although overall the best result is worse than on the WebQSP-WD data set. QALD-7 is much smaller, but also more complex on average (cf. the average number of edges needed to find the correct answer in Table 1). Overall, we can conclude that the explicit modeling of the structure of semantic graphs with GGNN results in a better performance both in-domain on WebQSP-WD and out-of-domain on QALD-7 that was only used for evaluation.

5.3 Error analysis

To better understand the difference between the approaches that encode graph structure and the approaches that disregard it, we closely look at the output of GGNN compared to the Pooled Edges model. The first two rows in Table 5 show how often the respective model has returned an answer with F-score higher than 0.5 which is a mostly correct answer and how often it returned an answer that was not just completely wrong (F-score > 0.0). We see that GGNN delivers an acceptable answer almost 25% more often than Pooled Edges model, but there is still a lot of questions that are not answered correctly.

We have manually analyzed 100 sample answers from the two models where the resulting F-score was lower than 0.5 (see rows 3 through 6 in Table 5). Since GGNN makes less mistakes in general, the error propagation from the entity linking takes a slightly larger portion of the final error count. In 18% of the cases, it was impossible to find the answer in Wikidata since there were no path between entities in the question and the answer. For example, for the question "Where did Harper Lee attend high school?", the correct answer "Monroe County High School" is a valid entity in Wikidata, but it is not connected to "Harper Lee" via the EDUCATED AT relation. 14% of the time, the data set contained an inconsistent answer and even though the model has predicted the correct semantic graph, the answers did not match. For example, a correct answer for a question about someone's place of birth is usually a city or a town, yet for a smaller set of questions a city borough (e.g. Manhattan) or a country are listed in the data set.

Overall, in 32% of the cases the error was caused by the gap between the KB and the data set. This lets us put the current results into a perspective with the previously reported numbers for the Freebase KB. If we approximately adjust our results for this kind of errors, we achieve between 0.469 and 0.51 F-score. Reddy et al. (2016) report results for various approaches ranging from 0.404 to 0.503 F-score on the original WebQuestions data set that is a superset of WebQSP-WD (see Section 4.1).

The majority of errors for both models are caused by wrong predictions (row 6 in Table 5). GGNN selects significantly less wrong semantic graphs and more often successfully handles graphs with multiple edges. For example, for a question "What language do people speak in Brazil?", the GGNN model correctly predicts the graph with two edges HOME COUNTRY and NATIVE LANGUAGE to get a list of all languages that are spoken in Brazil. Whereas the other models either select the relation OFFICIAL LANGUAGE that returns only the official language of the country or choose a wrong interpretation altogether. We also look at the hit@10 measure that shows how often the correct semantic graph was in the top 10 scored graphs by the model (row 7 in Table 5). Notably, in 44% of the cases the correct semantic graph was still among the top scored graphs for the GGNN model.

6 Related work

We have focused on the problem of the increasing error rates for complex questions and the encoding of the semantic graph structure. In this paper, we describe a semantic parsing system for KB QA and follow the approach of Yih et al. (2015) who do not rely on syntactic parsing to construct semantic parses. Our semantic graphs do not cover some aspects of the first-order logic, such as negation. Reddy et al. (2016) define a semantic parser that builds first-order logic representations from syntactic dependencies. They further specify how it can be extended with negation in Fancellu et al. (2017).

We only train on the WebQSP-WD data set and we note that more data might be necessary to effectively train the gated graph architecture. Reddy et al. (2014) suggest an unsupervised learning method to learn a model from a large web corpus, while Su et al. (2016) use patterns and crowdsourcing to create new data with specific properties. These techniques can be used to further improve the performance of our model.

An alternative solution to semantic parsing is to build an information extraction pipeline that views the question as a query and the KB as a source of relevant information (Yao et al., 2014). Dong et al. (2015) and Jain (2016) construct a vector representation for the question and use it to directly score candidate answers from the KB. However, such approaches are hard to analyze for errors or to modify with explicit constraints. For example, it is not directly possible to implement the temporal sorting constraint (argmax).

We apply GGNNs to the problem of semantic parsing. Li et al. (2016) have developed the gated architecture based on the graph neural network formulation of Scarselli et al. (2009). Recently, a slightly different design of Graph Convolutional Networks was proven effective on a KB completion task (Schlichtkrull et al., 2018). Kipf and Welling (2017) introduced Graph Convolutional Networks, while Marcheggiani and Titov (2017) employed them for natural language processing for the first time and compared them to other formulations. Graph Convolutional Networks have a similar gated architecture and share most of the same properties with the Gated Graph Neural Networks used.

7 Conclusions

In this work, we have used Gated Graph Neural Networks to encode the structure of the target semantic parse for KB QA. We have shown that disregarding the semantic structure leads to a falling performance on questions that require complex semantic parses to get the correct answers. Our GGNN architecture was able to successfully model the structure of semantic parses. We have compared the performance of GGNNs against the previous work and non-graph models on two data sets and have broken down the results by question complexity. The analysis has shown that the suggested graph architectures do not have the same drop in performance on complex questions and produce better overall results.

Recently, Peng et al. (2017) and Yu et al. (2017) have attempted to incorporate entity linking into a feature based QA model. In the future, we plan to follow their work and integrate GGNNs with entity linking into a single model. We also see possible applications for GGNNs on other semantic parsing tasks, such as text comprehension.

Acknowledgments

This work has been supported by the German Research Foundation as part of the Research Training Group AIPHES (grant No. GRK 1994/1), and via the QA-EduInf project (grant GU 798/18-1 and RI 803/12-1). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X GPU.

References

Junwei Bao, Nan Duan, Zhao Yan, Ming Zhou, and Tiejun Zhao. 2016. Constraint-Based Question Answering with Knowledge Graph. In *Proceedings of the 26th International Conference on Computational Linguistics (COLING)*, pages 2503–2514, Osaka, Japan.

Jonathan Berant and Percy Liang. 2014. Semantic Parsing via Paraphrasing. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1415–1425, Baltimore, MD, USA.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic Parsing on Freebase from Question-Answer Pairs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1533–1544, Seattle, WA, USA.

Kyunghyun Cho, Bart Van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation Kyunghyun. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar.

Li Dong, Furu Wei, Ming Zhou, and Ke Xu. 2015. Question Answering over Freebase with Multi-Column Convolutional Neural Networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 260–269, Beijing, China. Association for Computational Linguistics.

Federico Fancellu, Siva Reddy, Adam Lopez, and Bonnie Webber. 2017. Universal Dependencies to Logical Forms with Negation Scope. In *Proceedings of the Workshop "Computational Semantics Beyond Events and Roles"*, pages 22–32, Valencia, Spain. Association for Computational Linguistics.

Sarthak Jain. 2016. Question Answering over Knowledge Base using Factual Memory Networks. In *Proceedings of the NAACL Student Research Workshop*, pages 109–115, San Diego, CA, USA.

Diederik Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization.

Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *In Proceedings of the 5th International Conference on Learning Representations (ICLR)*, pages 1–14, Toulon, France.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated Graph Sequence Neural Networks. In *In Proceedings of the 4th International Conference on Learning Representations (ICLR)*, pages 1–19, San Juan, Puerto Rico.

Percy Liang. 2016. Learning Executable Semantic Parsers for Natural Language Understanding. *Communications of the ACM*, 59(9):68–76.

Diego Marcheggiani and Ivan Titov. 2017. Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1506–1515, Copenhagen, Denmark. Association for Computational Linguistics.

Haoruo Peng, Ming-Wei Chang, and Wen-Tau Yih. 2017. Maximum Margin Reward Networks for Learning from Explicit and Implicit Supervision. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2358–2368, Copenhagen, Denmark. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar.

Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale Semantic Parsing without Question-Answer Pairs. *Transactions of the Association for Computational Linguistics*, 2:377–392.

Siva Reddy, Oscar Täckström, Michael Collins, Tom Kwiatkowski, Dipanjan Das, Mark Steedman, and Mirella Lapata. 2016. Transforming Dependency Structures to Logical Forms for Semantic Parsing. *Transactions of the Association for Computational Linguistics*, 4:127–140.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, M. Hagenbuchner, and Gabriele Monfardini. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne vanden Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In Aldo Gangemi, Roberto Navigli, Maria-Esther Vidal, Pascal Hitzler, Raphaël Troncy, Laura Hollink, Tordai Anna, and Mehwish Alam, editors, *The Semantic Web*, pages 593–607, Cham. Springer International Publishing.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. 2017. Fast and Accurate Entity Recognition with Iterated Dilated Convolutions. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2670–2680, Copenhagen, Denmark.

Yu Su, Huan Sun, Brian Sadler, Mudhakar Srivatsa, Izzeddin Gur, Zenghui Yan, and Xifeng Yan. 2016. On Generating Characteristic-rich Question Sets for QA Evaluation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 562–572, Austin, Texas.

Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Bastian Haarmann, Anastasia Krithara, Michael Röder, and Giulio Napolitano. 2017. 7th Open Challenge on Question Answering over Linked Data (QALD-7). In Mauro Dragoni, Monika Solanki, and Eva Blomqvist, editors, *Semantic Web Challenges*, pages 59–69, Cham. Springer International Publishing.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: A Free Collaborative Knowledgebase. *Communications of the ACM*, 57(10):78–85.

Yi Yang and Ming-Wei Chang. 2015. S-MART: Novel Tree-based Structured Learning Algorithms Applied to Tweet Entity Linking. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 504–513, Beijing, China.

Xuchen Yao, Jonathan Berant, and Benjamin Van Durme. 2014. Freebase QA: Information Extraction or Semantic Parsing? In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*, pages 82–86, Baltimore, MD, USA.

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 *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 1321–1331, Beijing, China.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. 2016. The Value of Semantic Parse Labeling for Knowledge Base Question Answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 201–206, Berlin, Germany.

Mo Yu, Wenpeng Yin, Kazi Saidul Hasan, Cicero dos Santos, Bing Xiang, and Bowen Zhou. 2017. Improved Neural Relation Detection for Knowledge Base Question Answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 571–581, Vancouver, Canada.