End-to-End Entity Resolution and Question Answering Using Differentiable Knowledge Graphs

Armin Oliya Amir Saffari Priyanka Sen Tom Ayoola
Amazon Alexa AI Cambridge, UK
{aooliya,amsafari,sepriyan,tayoola}@amazon.com

Abstract

Recently, end-to-end (E2E) trained models for question answering over knowledge graphs (KGQA) have delivered promising results using only a weakly supervised dataset. However, these models are trained and evaluated in a setting where hand-annotated question entities are supplied to the model, leaving the important and non-trivial task of entity resolution (ER) outside the scope of E2E learning. In this work, we extend the boundaries of E2E learning for KGQA to include the training of an ER component. Our model only needs the question text and the answer entities to train, and delivers a stand-alone QA model that does not require an additional ER component to be supplied during runtime. Our approach is fully differentiable, thanks to its reliance on a recent method for building differentiable KGs (Cohen et al., 2020). We evaluate our E2E trained model on two public datasets and show that it comes close to baseline models that use hand-annotated entities.

1 Introduction

The conventional approach for Question Answering using a Knowledge Graph (KGQA) involves a set of loosely connected components; notably, an entity resolution component identifies entities mentioned in the question, and a semantic parsing component produces a structured representation of the question. The programs resulting from combining these components can be executed on a knowledge graph (KG) engine to retrieve the answers.

While this approach can be effective, collecting training datasets for individual components can be challenging (Dahl et al., 1994; Finegan-Dollak et al., 2018). For example, supervised semantic parsing requires training data pairing natural-language questions with structured queries, which is difficult to obtain. This has motivated many efforts in weakly supervised training (Chakraborty et al., 2021). Following recent breakthroughs in machine translation (Bahdanau et al., 2015), a new goal is to directly optimize the entire chain of components end-to-end, without the need for intermediate annotations.

However, entity resolution (ER) is by and large a neglected component of E2E learning, and existing weakly supervised solutions mostly assume question entities are either given or extracted by an external system. In practice, there’s a scarcity of quality training data for ER on questions, and poor entity extraction by out-of-domain models affects the overall performance of a KGQA system (Singh et al., 2020; Han et al., 2020).

In this work, we present an end-to-end model for KGQA that learns to jointly perform entity resolution and inference. Our work leverages the differentiable KG proposed in Cohen et al. (2020), which allows all the components of our model to be trained using a dataset of only questions and answers. This eliminates the need for labelled ER data for questions and allows our model to run independently, without relying on external components. Furthermore, the tight integration of ER into our
solution allows uncertainties about entities to be directly reflected in our confidence in answers.

2 Related Work

Traditional approaches to KGQA rely on semantic parsing (SP) to translate natural language into a logical form. Weakly supervised SP is a well-studied topic with increasing interest in applying Reinforcement Learning (RL) (Hua et al., 2020; Agarwal et al., 2019). ER is rarely considered in the scope of surveyed solutions, and if it is, it’s treated as an independent component and not included in the weak supervision scope (Ansari et al., 2019). In general, RL algorithms for QA are hard to tune and have large variances in their results. The exploration-exploitation issue also lets models settle on high-reward but spurious logical forms, leading to poor generalization (Chakraborty et al., 2021). Including ER with a discrete output space is the element-wise multiplication.

3 E2E Model

3.1 Differentiable Knowledge Graph

A traditional knowledge graph (KG) stores facts as triples and uses a symbolic query engine to extract answers. A differentiable KG stores facts in tensors and makes query execution over facts differentiable.

We use the approach presented in ReifiedKB (Cohen et al., 2020) to create a scalable and differentiable knowledge graph supporting multi-hop relation following programs. We provide an overview here but full details can be found in the original paper. Assume the set of all triples in a knowledge graph \( T = \{t_i\}_{i=1}^{N_T} \), \( t_i = (s_i, p_i, o_i) \) are represented by the following sparse matrices:

\[
M_s \in \{0,1\}^{N_T \times N_E}, \quad M_o \in \{0,1\}^{N_T \times N_E}, \quad M_p \in \{0,1\}^{N_T \times N_R},
\]

where \( M_s, M_o, \) and \( M_p \) are denoted as subject, object, and relation index matrices. \( N_T, N_E, \) and \( N_R \) are the number of triples, entities, and relations, respectively.

Given an entities vector \( x_{t-1} \in \mathbb{R}^{N_E} \) at \( t - 1 \)-th hop, the entities vector \( x_t \) resulting from following a relation vector \( r_t \in \mathbb{R}^{N_R} \) can be computed by:

\[
x_t = \text{follow}(x_{t-1}, r_t) = M_o^T (M_s x_{t-1} \odot M_p r_t), \tag{1}
\]

where \( \odot \) is the element-wise multiplication.

3.2 Multi-hop Inference

Given an input question \( q = q_1 \cdots q_n \) of length \( n \), we first use the pretrained language model RoBERTa (Liu et al., 2019) to extract contextual embeddings for each token:

\[
[h_q, q_1 \cdots q_n]^T = LM(CLS, q_1 \cdots q_n) \tag{2}
\]

where \( h_q \in \mathbb{R}^D \) corresponds to the \( CLS \) token and is used as the question embedding. We compute the relation vector for the \( t \)-th hop using a hierarchical decoder and subsequent entities vector by following that vector as:

\[
r_t = \text{softmax}(W_t^{\text{att}} [h_q | r_{t-1} | \cdots | r_1]^T), \tag{3}
\]

\[
x_t = \text{follow}(x_{t-1}, r_t). \tag{4}
\]

Since the follow operation (Equation 1) can be chained infinitely, we set a maximum number of hops and use an attention mechanism to combine answer entities across all hops. We compute attention across all hops by:

\[
c_t = W_t^{\text{att}} [h_q | r_{t-1} | \cdots | r_1]^T, \tag{5}
\]

\[
a = \text{softmax}([c_1, \cdots, c_{T_{\text{max}}}]^T), \tag{6}
\]
where \( T_{\text{max}} \) is the predefined maximum number of hops. The final answer entities vector will be:

\[
\hat{y} = \sum_{t=1}^{T_{\text{max}}} a_t x_t. \tag{7}
\]

Compared to ReifiedKB, our decoder uses RoBERTa for embedding questions, simplifies the stopping mechanism, and allows returning more than one answer entity.

### 3.3 Entity Resolution

We approach Entity Resolution by estimating the likelihood of all the plausible spans in the question and then selecting the most likely candidate entity for each span.

Given an input question \( q = q_1 \cdots q_n \) of length \( n \), the likelihood of each span \( [i, j] \) in the question (\( i \)-th to \( j \)-th tokens of \( q \)) is calculated as:

\[
q_{ij} = \frac{\sum_{k=1}^{j} q_k}{(j - i + 1)} \in \mathbb{R}^D, \tag{8}
\]

\[
s_{ij} = \frac{\exp(q_{ij} w_s)}{\sum_{m,n} \exp(q_{mn} w_s)}, \tag{9}
\]

where \( w_s \in \mathbb{R}^{D\times1} \) is a learnable matrix and \( q_k \) are contextual token embeddings from Equation 2.

For a given span, candidate entities that could be referred to by that span are extracted by exact search against a lookup table, built using titles and aliases of entities in the KG. Candidate generation can further be improved by considering other approximate or fuzzy search methods, but we leave this as future work.

If there are overlapping candidates between two spans, they are assigned to the longer one. For example, consider the question "what position does carlos gomez play?". If candidates of the three spans "carlos", "gomez", and "carlos gomez" all contain 2747238 (the Wikidata entity ID referring to Carlos Gomez, the Dominican baseball player), the entity ID will be assigned to the longest span only ("carlos gomez"). This is to avoid having duplicate entities across spans and comes from the intuition that longer spans are more specific and should be preferred. We have not seen errors arising from this preprocessing step.

Assume \( c_{ij}^k \) is the \( k \)-th candidate for span \([i, j]\). We embed each candidate entity by learning a dense representation of its KG neighbourhood:

\[
F_{ij}^k = \{ f_m([p|o]) \mid (c_{ij}^k, p, o) \in G \}, \tag{10}
\]

\[
z_{ij}^k = \sum_{u \in F_{ij}^k} v \quad \frac{z_{ij}^k \cdot q_{ij}}{|F_{ij}^k|} \in \mathbb{R}^D, \tag{11}
\]

where \( G \) is the knowledge graph, \( [p|o] \) is the string concatenation of \( p \) and \( o \), and \( f_m \) is an embedding function that maps a string to a dense vector.

For example, assume Q2747238 is the candidate entity we want to embed. It is on the left-hand side of two triples in the knowledge graph: (Q2747238, occupation, baseball player) and (Q2747238, instance-of, human) and (Q2747238, instance-of, human) and (Q2747238, instance-of, baseball player).

We first create the two strings "instance-of: human" and "occupation: baseball player" and pass them to \( f_m \) to embed. These strings are treated as features of Q2747238, and we are able to learn embeddings effectively since they are also used for other humans and baseball players. Finally, we take the average of these two feature embeddings to get to the entity embedding for Q2747238. These operations are implemented using \texttt{torch.nn.EmbeddingBag} in PyTorch with random initialization. Our approach is not limited by knowledge graph features or this specific embedding approach; for instance, a RoBERTa encoding of entity descriptions could be used as an alternative. We leave experiments with other entity representations as future work.

Given the embedding, the likelihood of a candidate entity is estimated by considering the span likelihood and the likelihood of other candidates in that span:

\[
e_{ij}^k = s_{ij} \frac{\exp(|z_{ij}^k \cdot q_{ij}|)}{\sum_{v} \exp(|z_{ij}^k \cdot q_{ij}|)}. \tag{12}
\]

To get to the final entity vector, we re-score candidate entities across all spans:

\[
x_{ij}^k = \frac{\exp(e_{ij}^k)}{\sum_{u,v,w} \exp(e_{uv}^w)}; \tag{13}
\]

\[
x_0 = x_{ij}^k \mapsto \bar{0} \in \mathbb{R}^N_e; \tag{14}
\]

where \( \mapsto \) maps each candidate entity likelihood to its corresponding index in the zero vector \( \bar{0} \). The resulting \( x_0 \) vector is used in Equation 4 and captures uncertainties about entity resolution. It is different from Cohen et al. 2020 where \( x_0 \) is assumed to be given with \{0, 1\} as the only possible values.
3.4 Training

We train the model using the binary cross-entropy loss function:

$$\mathcal{L}(\hat{y}, y) = \frac{1}{N_E} \sum_{i=1}^{N_E} y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i),$$

(15)

where $y \in \mathbb{R}^{N_E}$ is a k-hot label vector. While RiefiedKB uses cross-entropy loss, we instead use a multi-label loss function across all entities. This is because the output space in a majority of cases contains multiple entities, so cross-entropy loss is inadequate. During training, the entity resolution and inference modules are trained jointly and uncertainties about each module are propagated to final answer entities vector $\hat{y}$.

4 Experiments

We call our model Rigel and evaluate 3 versions at different E2E learning boundaries. The baseline model, Rigel-Baseline, is given gold entities and no entity resolution is involved, demonstrating the performance of the inference module alone. Rigel-ER is given the gold spans, but still has to learn to disambiguate between candidate entities for that span. Finally, in Rigel-E2E, we provide the question text only, requiring the model to attend to the right span and disambiguate between candidates for each span.

4.1 Datasets

We evaluate our models on two open-domain Question Answering datasets: SimpleQuestions (Bordes et al., 2015) and WebQSP (Yih et al., 2016). Both datasets were constructed based on the outdated FreeBase. Therefore, to generate better candidates and entity representation, we chose to use a subset of these datasets that are answerable by Wikidata (Diefenbach et al., 2017). This is different from other baselines we compare against, which do not include an ER component. For WebQSP, this leads to 2349 train, 261 dev, and 1375 test set samples. For SimpleQuestions the number of samples are 19471 train, 2818 dev, and 5620 test.

Questions in SimpleQuestions and WebQSP can be answered in 1 and 2 hops respectively, so we set the maximum number of hops $T_{max}$ in Equation 7 accordingly. For each dataset, we also limit Wikidata to a subset that is $T_{max}$-hop reachable from any of the candidates $c_j^k$ in Equation 10. This results in a subgraph with 3.7 million triples, 1.0 million entities, and 1,158 relations for SimpleQuestions; and 4.9 million triples, 1.1 million entities, and 1,230 relations for WebQSP.

4.2 Results

Results of our experiments are shown in Table 1. We don’t directly compare to other related work since their performance is reported with access to gold-entities and their quality when building a practical QA system with an external ER is unknown.

Compared to Rigel-Baseline, there is approximately a 3% drop in performance when gold question entities are not provided to the model (Rigel-ER). We realized this is mainly due to cases where it is not possible to distinguish between all possible candidate entities based on the question alone. This is consistent with earlier studies that conclude 15-17% of questions in these datasets cannot be answered due to context ambiguity (Han et al., 2020; Petrochuk and Zettlemoyer, 2018). For example, in the question “What position does carlos gomez play?” ("carlos gomez" given as correct span), Rigel-ER learns to give higher likelihood to athletes compared to art performers; but since the question does not include discriminative information such as sport or team name, all athletes called "Carlos Gomez" will receive very similar likelihood scores.

There is a further drop in performance when we go from Rigel-ER to Rigel-E2E, which performs full E2E learning. This time, the errors can be explained by the fact that different spans produce candidates with overlapping entity types, leaving the model with little signal to prefer one span over another.

For example, given the question “who directed the film gone with the wind?”, Rigel-ER is given the correct span “gone with the wind” and just needs to disambiguate between Q2875 (the Wikidata entity ID for the American film “Gone with the Wind”) and the other candidates stemming from that span. Rigel-E2E will additionally need to learn to maximize the span score (Equation 9) for “gone with the wind” compared to other spans in the question, such as “the film”, “the wind”, and “wind”, which are all film titles as well. This is a difficult task since all these spans produce film entities, and relying on the loss from following the director relation is not enough to effectively disambiguate between them.

We are working on a few solutions to allevi-
ate this span ambiguity issue with Rigel-E2E. The main question is, what should we do when span scores are diffused and not spiked? This, for example, happens in the above question and the 4 spans: “gone with the wind”, “the film”, “the wind”, and “wind”. A simple post-processing step to merge overlapping spans seems to be quite effective. In the example above, “the wind” and “wind” fall under “gone with the wind”, and given that their scores are similar we can decide to assign all child span scores to their parent. Diversity or entropy of candidates produced by a certain span also seems to be helpful in pruning bad spans. In the above question, candidate entities for the span “wind” include movies, companies, music bands, and even a satellite, among others. On the other hand, candidate entities for “gone with the wind” are mostly works of art, suggesting that it may be a better choice. We are looking into using this information as part of training, as well as post-processing.

While we don’t directly compare, the gap between our results and other related work is partly due to the inference mechanism used. At this time, ReifiedKB only supports a relation following operation (Equation 1), while, for instance, EmQL (Sun et al., 2020) additionally supports set intersection, union, and difference. These additional operations allow answering more complex questions present in the WebQSP dataset. We are working on adding support for intersection, union, count, min, and max operations to our model as future work.

We’d like to emphasize that although including the ER component adversely affects the results, extracting question entities is a necessity for real world applications, and alternatives with off-the-shelf models do perform worse. Hence, we believe our approach is more practical, especially given the lack of training data for ER on questions.

5 Conclusion

In this work, we proposed a solution for KGQA that jointly learns to perform entity resolution (ER) and multi-hop inference. Our model extends the boundaries for end-to-end learning and is weakly supervised using pairs of only questions and answers. This eliminates the need for external components and expensive domain-specific labelled data for ER. We further demonstrate the feasibility of this approach on two open-domain QA datasets.


Yuncheng Hua, Yuan-Fang Li, Guilin Qi, Wei Wu, Jingyao Zhang, and Daqing Qi. 2020. Less is more: Data-efficient complex question answering over knowledge bases. Journal of Web Semantics, 65:100612.


A Model Hyperparameters

We train Rigel models using the hyperparameters below on a single GPU machine with 16GB GPU memory (AWS p3.2xlarge). WebQSP requires more than 1 hop for question answering, leading to a larger knowledge graph, so we use a smaller batch size to avoid out of memory issues. Training with early stopping completes in approximately 4-7 hours depending on the model configuration used (Rigel-baseline, Rigel-ER, Rigel-E2E).

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>SIQ</th>
<th>WQSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch Size</td>
<td>32</td>
<td>6</td>
</tr>
<tr>
<td>Gradient Accumulation</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td>Max Training Steps</td>
<td>20000</td>
<td>30000</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>1e-4</td>
<td>1e-4</td>
</tr>
<tr>
<td>Max Number of Hops</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2: Hyperparameters for training on WebQuestionsSP (WQSP) and SimpleQuestions (SIQ)

B Examples

The table below shows outputs of Rigel-E2E model on two questions from SimpleQuestions. In the first example, the model assigns high likelihood to the correct span and candidate entity. The inference module also assigns a high likelihood to the right relation (instance of), which leads to the correct answer entity.

In the second question, the model assigns higher likelihood to the sam edwards span, but it’s not very confident and other spans such as sam and edwards receive similar scores. In addition, there’s a large overlap between candidate entities of these spans (i.e. all produce candidates which are human and have place of birth property). This ambiguity in context leads to the ground truth question entity receiving a low likelihood. Even though the right relation is predicted by the inference module, the final answer entity is different from the answer label.
Question: what is the category of the celestial object 1241 dysona?
Answer Entity Q3863 (asteroid)

Span Likelihoods:
('1241 dysona', 0.965), ('celestial’, 0.013), ('celestial object’, 0.011),
('object’, 0.009), ('1241’, 0.002), ('the category’, 0.0), ('what is’, 0.0), ('category’, 0.0)

Candidate Entity Likelihoods:
('Q137259', 0.998), ('Q6999’, 0.0), ('Q66311333’, 0.0), ('Q488383’, 0.0), · · ·

Top Prediction:
('1241 dysona’, 0.965) → (instance of, 1.000) → ('Q3863’, 0.998)

Question: what is the place of birth of sam edwards?
Answer Entity Q23051 (Swansea)

Span Likelihoods:
('edwards’, 0.366), ('sam edwards’, 0.332), ('sam’, 0.301), ('what is’, 0.0), ('the place’, 0.0),
('place’, 0.0), ('the place of birth’, 0.0), ('place of birth’, 0.0), ('birth’, 0.0)

Candidate Entity Likelihoods:
('Q3470479’, 0.25), ('Q835638’, 0.111), ('Q911493’, 0.058), ('Q2691159’, 0.017),
('Q20812281’, 0.016), ('Q1816301’, 0.014), ('Q47465190’, 0.013), ('Q1118055’, 0.011),
('Q58317511’, 0.01), ('Q472382’, 0.01), ('Q27925002’, 0.01), ('Q52852726’, 0.009), · · ·

Top Prediction:
('Q3470479’, 0.25) → (place of birth, 1.000) → ('Q219656’, 0.250) ✗

Table 3: Example outputs of Rigel-E2E on SimpleQuestions