Question Answering on Google Knowledge Graph via Entity Linking and Relation Extraction

Stephanie Uduji
Stanford University
suduji@stanford.edu

Hujia Yu
Stanford University
hujiay@stanford.edu

Abstract

In this paper, we focused on building a question answering (QA) model, more specifically, we reproduced QA models via entity linking and relation extraction introduced by paper from [4]. During implementations, we explored QA tasks with answer inferences based on a combination of structured and unstructured data. We completed a preliminary model with all functional pieces of a QA system, we have encouraging results on performance on all parts of the model and we are in the process of piping them together to make it one model. With Google Knowledge Graph API [11] as our structured data source, our entity linking model was able to achieve at least 86% for test accuracy. This is an improvement to the paper, which reached a test accuracy of 79.8%. On the other hand, the relation extraction model also performed surprisingly well, with features representations suggested by the paper [4], our relation extraction model is able to obtain 73.14% accuracy on the test set using a simple two-layer neural network model of 800 cells per layer. This is also a significant jump from the original paper, however we slightly pruned our training dataset to the classes with more number of samples to increase model learning.

1 Introduction

We started this project because we think this is an important topic to research. QA is a very common need in real life. Many tasks can be formulated as QA problems, such as machine translation and relation extraction. QA also helps us in understanding other related NLP tasks in general. In this project, we focused on question answering (QA) via relation extraction and validation with supporting textual evidence based on the *Question Answering on Freebase via Relation Extraction and Textual Evidence* by Xu, Reddy, et. al [4]. Our specific goals for this project were to explore QA tasks with a knowledge-based relation extractor first, using easy-to-understand-and-implement dependency parsing trees to parse questions into sub questions and find relations between input entities. We also explored using a more extensive knowledge graph (rather than a traditional RDF structured knowledge base) to improve entity linking accuracy and incorporating models with better predictive power, such as RNN structure or structured SVM, and to combine with external resource as textual evidence (such as Wikipedia) to achieve higher accuracy during the relation extraction process.

2 Background and Related Work

During our exploration of literature reviews, we found the following approaches to be inspiring to our current approach, our discussion of prior approaches will be divided into the approaches on QA systems and those on information retrieval/relation extraction, since both parts are equally important in building a QA model.

Firstly, Jurafsky and Martin provide a summary of the standard in QA in Speech and Language Processing Chapter 28 [1]. The task (specific to factoid questions) has multiple common methods. The first of which is Information Retrieval-based QA. In this method, answers are found by extracting related texts and sources on the web. The Question Processing step involves determining the answer type, marking keywords in the query, and determining the subject. The type of answer that the question is looking for is also determined with supervised learning. This requires a hand-labeled dataset; however, the paper includes a potential data source, Webcopedia QA Typology, which has 276 hand-written rules and 180 different answer types (Jurfasky, Martin 4). [1] Texts that hold answer candidates are found by performing a search for relevant documents either over a set corpus or a web search engine and relation extraction is used to filter out answers.

Another method, Knowledge-based question answering, involves running a query over a structured database. The previous method was a search over unstructured text. While this method involves the usage of hand-written rules, bootstrapping (like in relation extraction) can be used to find new patterns. Unlike the bootstrapping method in relation extraction, the data structure is now of a logical form in which entities are members of predicates (i.e. from relational logic). The challenge in this method lies in creating logical forms that accurately describe relations between entities.

The last stand-alone method, semi-supervised learning, involves generating multiple versions of paraphrases of questions and performing queries with each of these sets of paraphrased questions. The intuition behind creating paraphrases is to avoid confusion about syntax, word omissions/additions, etc. in the varied forms the same question may take. This involves manual work; however, another option is to use a public data source of questions and common paraphrases. The paper cites wikianswers.coms PARALEX corpus which has sets of millions of questions that have the same meaning. This will not do so well with very specific or uncommonly asked questions; however, it is an option that avoids the need to hand-label multiple sets of paraphrases.

Hybrid methods that utilize both Information Retrieval and KB-based question answering are also other options. The paper gives an example, the DeepQA system from IBMs Watson, a computer that can rival champions at Jeopardy. In future work and further research, it would be in our best interests to learn more about the specifics of the underlying methods used in the DeepQA system as it is an example of successful Question Answering. It is also another example of a successful combination of the most common methods in QA.

In addition to pure QA systems, an essential step is relation or information extraction. The goal of information extraction is to create structured data from unstructured data. For example, one particular task might be determining the relationship between two people (if any) mentioned in block of text. In Speech and Language Processing Chapter 21 [1], by Jurafsky and Martin (2017) [2], a general overview of modern techniques in Information Extraction provides a good starting point for our project. The paper cites five main classes of algorithms for relation extraction: hand-written patterns, supervised learning, semi-supervised learning, distant supervised learning, and unsupervised learning.

3 Dataset

According to our source paper, the utilized datasets include Freebase (structured RDF data) for the entity linking and extraction steps, Wikipedia pages (processed with Wikifier) for the answer inference on unstructured data, and WebQuestions as the QA dataset.

Following the paper's implementation[4], our question and answer set is extracted from the WebQuestions dataset which contains 5,810 questions (from which we use only 2875 questions). We use the raw question text as input for the first step in our process, entity linking, and evaluate based on a structured set the paper provides. This set includes the question, entity, KB instance, and relation/type.

We initially proceeded in our implementation with a more limited version of Freebase as our KB to be used in the entity linking and extraction steps. While the paper's KB contained 4M entities and 5,323 relations, our small set held only 46,275 entities and 16 relations in RDF structured format. The decision to use this smaller set was made in consideration of the deprecation of the Freebase API. In replacement, we thought extracting a smaller portion of the set would suffice, as we were also hand-crafting training and test questions based on our smaller KB. In the future, we would expand the KB by downloading a larger set. However, in using this dataset in conjunction with the data extracted from WebQuestions for QA train and test data, we, unsurprisingly, achieved very poor accuracies (as we'll describe in detail later in our "Results" section) for the entity linking and extraction steps.

Further along in our implementation process, we decided to switch to a more extensive knowledge base and found Google's Knowledge Graph Search API [11]. The API allowed us to access around 570 million entities and 18 billion facts, so it would most definitely suffice for our purposes. However, incorporating this new dataset into our model required a significant amount of code revision and restructuring of intermediate data. Additionally, the lack of detailed documentation rendered it quite difficult to know the full capabilities of the API. In particular, it is not possible (based on the limited documentation) to extract entities from the knowledge graph based on relations. This is essential in the joint entity linking and relation extraction step, so we needed to alter the implementation of this step (as we'll describe later in our "Implementation" section).

In switching the Google's Knowledge Graph API, we were able to drastically improve entity linking accuracies. Although incorporating the API meant deviating from the paper's implementation, it was worth the workarounds.

We wanted to make sure our dataset contains sufficient information for training and is easy to use. Figure 1 and 2. give snapshots of two files of our training/dev dataset. Specifically, our dataset is formatted in question-answer pairs with annotated question entities. Moreover, in the KB-QA training part, the dataset also gives the KB relations corresponding to each question entities to train the relation extractor.

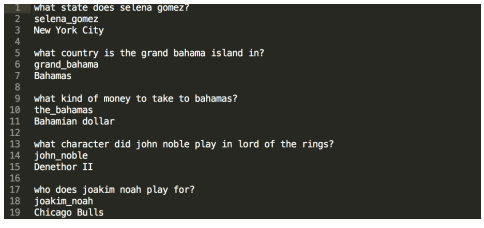

Figure 1: Training KB Relation Data and Question Answer Pairs. Question answer pairs with entity.

```
what state does selena gomez?
selena gomez
m.0gs6vr
people.person.places_lived..people.place_lived.location

who does joakim noah play for?
joakim noah
m.0c2yrf
sports.drafted_athlete.drafted..sports.sports_league_draft_pick.team

what kind of money to take to bahamas?
bahamas
m.0160w
location.country.currency_used

what country is the grand bahama island in?
grand bahama island
m.03st9j
location.location.containedby
```

Figure 2: Training KB Relation Data and Question Answer Pairs. Question Answer Pairs with KB relations.

4 Our Approach

Our current approach is to empower a knowledge-based question answering system that is built using entity linking and relation extraction and integrated with additional evidence from Wikipedia to give more accurate and expressive answers. Following the model of QA via relation extraction and validation with supporting textual evidence in Xu, et. al's paper [4], this approach fully recognizes the capability of using knowledge bases like Freebase to capture real world facts, yet the model tries to improve the accuracy by incorporating web resources to validate or support these facts. The idea is simply to answer our question using KB relation extractor first, and to filter out wrong answers using external web resources. This novel method for question answering which infers both on structured and unstructured data is then to be evaluated on a benchmark dataset WebQuestions in the end.

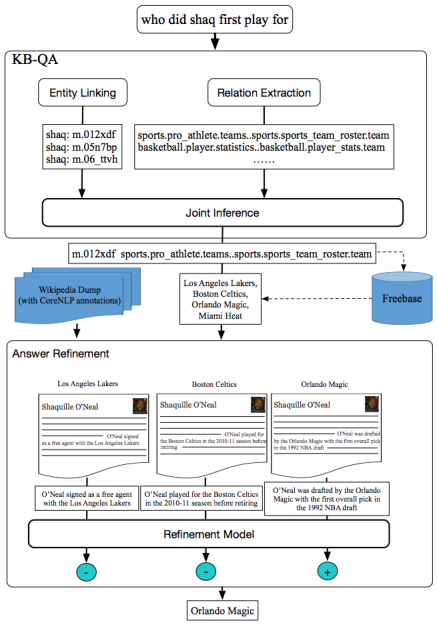

Figure 1: An illustration of our method to find answers for the given question *who did shaq first play for*.

Figure 3: Question Answering System

Figure 4 above gives an overview of our method for the example question "Who did shaq first play for? ". There are two main steps: (1) inference on Freebase (KB-QA box); and (2) further inference on Wikipedia (Answer Refinement box). Furthermore, there are three steps in our step 1 (KB-QA box), and they are entity linking, relation extraction, and joint inference. More specifically, given a question, we first divide it to sub questions if needed, and then perform entity linking to identify a top entity in the sub question and its possible Freebase entities. Next, we use a relation extractor to predict the potential Freebase relations that could exist between the linked entities from the question and the (potential) answer entities. After that, a joint inference is performed to step over the entity linking and relation extraction results to find the best entity-relation pairs which will produce a list of candidate answer entities. This list of answer pairs is then pruned to filter out wrong answers in step 2 using the answer refinement model, which takes the Wikipedia page of each answer entity pair and looks for validation.

4.1 Inference on Freebase

This is the part one of our model and is also the most important part. Namely the inference based on freebase can itself be a standalone QA system, and the second part (answer refinement box) are just some extra steps to improve accuracy. Therefore, a correct and effective implementation of inference based on Freebase is key to success of the entire model. This inference part involves three major steps: entity linking, relation extraction, joint entity linking and prediction. Below we explain our implementations for each step we have completed.

4.2 Entity linking

The goal of this initial part is to identify possible entities in a given question and extract the entities from the KB. The input is a question given in raw text form with standard English. Entity mentions are identified in the raw question text and then are used to search for and link to the actual mention in the KB. Because mentions could refer to many different entities, a ranked list of possibly relevant entities is returned as the output of this step. Due to the change in datasets used for this step, we will provide more detail for implementations using each in our "Implementations" section.

Question: "where is ruben rausing from?"
WP NN VB

Question word: ['where']
Search mentions: ['ruben', 'rausing', 'ruben rausing']

Linked Entities: [('Ruben_Rausing', inf), ('Aaron_Ruben', 1), ('David_Rubenstein', 1), ('Ruben_Santiago-Hudson', 1)]

Figure 4: Mention Identification and Entity Linking Ex.

4.3 Relation Extraction

After we implement entity linking, which can identify possible entities given a raw text question, we now want to infer the Freebase relation corresponding to question entities and the answer entities, which is the goal of relation extraction part.

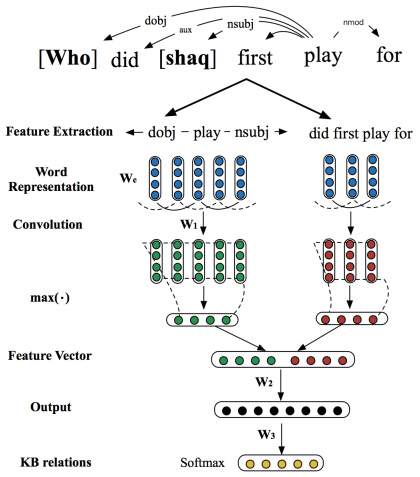

Figure 5: Overview of the Multi-channel Convolutional Neural Network for Relation Extraction.

We implemented a simplified version of Multi-Channel Convolutional Neural Network (MCCNN) for relation extraction. Figure 5 above gives an overview of the original neural network of MCCNN in the paper. According to this model, sentence is first parsed into features using dependency parser and the shortest path between the question word and entity are found and concatenated to feed into model. Specifically, two types of features are extracted from a given question: syntactic features and sentential features. Syntactic features are extracted using Stanford CoreNLP dependency parser [10] to get the shortest path between the question word and an entity word in the question, which is represented using a concatenation of dependencies between words, non-entity words and path arrows for any two given pairs of question word and the question entity. On the other hand, sentential features are simply words in the questions excluding the entities themselves. These two set of features are then both represented using a pre-trained 50d GloVe model and fed into a convolutional neural network separately. The output is then concatenated to be one fixed-length feature vector after max pooling, which is then used to predict KB relations. More details on specific algorithm will be given during implementation section below.

4.4 Joint Entity Linking & Relation Extraction

In this step, we attempt to combat error propagation from Entity Linking to Relation Extraction, like in the paper, by ranking a list of (entity, relation) pairs based on an SVM Classifier. The features we used are extracted from entities, relations, and candidate answers. This step uses a list of ranked (entity, relation) pairs as training input. After predicting rankings for test input, The highest ranked (entity, relation) pair in the ranked output will be used as input in the Inference step to extract candidate answers from the KB. As mentioned before, our change in datasets resulted in a revision of this step's implementation. We will describe implementations with each dataset in the next section.

4.5 Wikipedia Validation

This step builds on top of the success of all prior steps, after we use the best ranked entity-relation pair from the above step to retrieve candidate answers from Freebase(or Google Knowledge Graph API[11]). In this step, we validate these answers using Wikipedia as our unstructured knowledge to further improve accuracy.

5 Implementations

5.1 Entity Linking

We identify entities in a question by using part of speech tagging. We consider mentions as question words (i.e. "who", "what", "where", etc. with a POS tag that starts with "W") and nouns with POS tags shown in Table 1 below (i.e. POS tag that starts with "N"):

NN	noun, singular or mass	table
NNS	noun plural	tables
NP	proper noun, singular	John
NPS	proper noun, plural	Vikings

Table 1: Part of Speech Tagging used in Entity Linking.

Because some mentions might be phrases rather than single words, we also consider words with POS tags of "DT" (determiner, i.e. articles such as "the", "an", "a", etc.), "IN" (prepositions or subordinating conjunctions, i.e. "in", "of", etc.), and any that start with "J" (i.e. various forms of adjectives). We then string these words with adjacent noun-words together to form a complete phrase. This allows us to extract mentions of both single words and phrases to be used in search of relevant entities in our KB.

5.1.1 Entity Linking with Small Freebase Dataset

Our initial implementation with the smaller Freebase KB as our dataset diverges from the original implementation in the paper that includes an additional step of linking these mentions to entities in the Freebase with the entity linking tool, S-MART. This tool is not available for usage in python and the process of porting the tool would be extremely time consuming. Instead, we used the mentions identified with POS tagging to find matches in the KB based on substrings (i.e. if we've identified "smith" as a mention, we will check for "Sam Smith," "Adam Smith," "Smithsonian," etc. in the KB). Exact matches of the mentions will receive a count

ranking of float('inf'). Other matches will receive count rankings based on the number of times the entity is found to be a match. We retrieve up to seven entities using this ranking scheme. The output is an ordered list (based on the ranking) of entities with information containing the entity name, KB instance/id, relation, and ranking.

5.1.2 Entity Linking with Google Knowledge Graph

The final version of our implementation for this step uses the Google Knowledge Base Search API[11] to search a much larger dataset. Although the data is structured, it does not use the RDF format. This difference is not relevant in this step. We still proceed by identifying entity mentions in the raw question text by using POS tagging. Then, using those mentions, we formulate queries to send requests for entities in the knowledge graph. The response data format for entities returned includes the entity name, KB instance/id, description, entity type, and score. We rank the returned entities in the same way as described previously.

5.2 Relation Extraction

Implementation steps for relation extraction part are listed as the follows:

1. Dependency Parsing: given a question such as *Who did shaq play for?*, we first need to get its dependencies using a dependency parser, this part is done using Stanford parser 20140827 version, dependencies in forms of triplets are then extracted from the above sentence, one triplet looks like the following:

   (('play', 'VB'), 'aux', ('did', 'VBD'))

   indicating the dependencies between word play and did in the above sentence.
2. Shortest path: given triplet dependencies between all words in a sentence, features we want to extract is the shortest path between the question word and the entity word according to the paper, any search algorithm would work fine for this part. A feature path is represented using dependencies between words.
3. sentential features are represented using words in the sentence other than mentioned entity words and question words.
4. load pre-trained GloVe models to get word representations for feature words extracted above, and concatenate to a m by 50 matrix, since we use GloVe-50.
5. After we have our training data matrix, we map each training data with its label, in this case, our training data is the extracted features from each question, and label is the potential relation between this mentioned entities to its answer entities. After this step, we have (feature, label) pair, where features are m by 50 matrix, where m differs by different question lengths and dependencies.
6. Training:
   (a) In order to train in batches of data, we flatten each data point to a vector of length m by 50, and we concatenate $batch\_size$ of questions together each time, resulting in train size of $batch\_size$ by $max(m_1 * 50, m_2 * 50, ..., m_{batch\_size} * 50)$, namely we pick our feature size to be the longest feature vector size during concatenation(so that we can form a batch), and we pad the extra space with zeros.
   (b) Now we have a well defined training model where input and output are cleaned and clearly specified from prior steps, we can train any types of classifiers at this time. In this project, we have trained and explored with three different types of classifiers: two-layer feed-forward neural networks of 800 hidden units in each layer, Support Vector Machine(SVM) classifier, and Random Forest classifier. Results will be discussed in Results section below.

For this part of the model, we use the KBtraining dataset as shown in 1 and 2, during training we have discovered that the dataset is not evenly distributed among different classes, and therefore our classifiers cannot learn efficiently due to lack of sufficient exposure to the classes with few samples. To mitigate this problem, we dropped all classes with samples ≤ 30. After this step we are left with 20 classes, and a total of 1071 train dataset and 528 test set. Some of the classes look like the following: `[book.book_subject.works, book.author.works_written, location.location.containedby, people.person.profession, ...]`.

One modification that we made during this part of implementation is that when pre-trained GloVe model is used to represent the dependencies features extracted from a given question, some of the dependency terms are not found in the GloVe vocabulary. For example, dependencies like 'dobj', or 'nsubj' are not found in GloVe. As a result, vector representations of 'object' and 'subject' are used to represent those dependencies if they have a full service form, else a randomly initialized vector of the same dimension is used.

5.3 Joint Entity Linking and Relation Extraction

This step ranks (entity, relation) pairs using an SVM classifier. It takes a ranked list of pairs as input and trains the classifier on the features extracted from them. The input is checked against a gold standard list to produce a rank for each input. The ranking algorithm is as follows: rank 3 if both the entity and relation match the gold standard's entity and relation, rank 2 if only one of the entity or relation match, rank 1 if both the entity and relation do not match the

gold standard. We then extract features from the input after ranking. These features are as follows:

1. Entity Features:
   - Score of entity from entity linking step
   - # of word overlaps between mention and entity name
2. Relation Features:
   - Sum of tf-idf scores of words in a question with the relation
   - Presence of surface form of the relation in the question
3. Answer Features:
   - # of candidate answers for the (entity, relation) pair
   - # of co-occurrence of relation and question word

5.3.1 With the Small Freebase KB

We extract the entity features and relation features by using the data stored in the entity and relation objects. The entity object includes the entity name, question it was extracted from, mentions identified in the question, KB instance/id, relation/types, and score from entity linking.

The answer features are extracted by querying the KB with information stored in both the entity and relation objects. In this implementation version, we queried our small Freebase dataset. We extract candidate answers for a question from the KB by searching for (entity, relation, ?) triples where ? is filled in by possible answers (objects/subjects in a triple with matching relation and entity in either field). The co-occurence feature refers to the assumption that we can make of the type of answer type to expect for a given question word. For example, "who" might refer to a Person or Organization, while "where" might refer to a Place or City.

Each of these sets of features is combined, and we then normalize the feature vectors and train the SVM classifier to predict rankings for a test data set. Because we don't know the parameter values (i.e. C and gamma) that will give good results, we used hyper-parameter searching via Grid_Search (from the sklearn library) to identify optimal parameter settings.

5.3.2 With the Google Knowledge Graph API

Using the Google Knowledge Graph API results in a change of implementation for the answer feature extraction portion of this step. All other parts remain unchanged. While the Knowledge Graph is structured data with entities, it is not RDF structured and does not contain triples of (relation, subject, object). This different scheme means that we cannot search for triples to extract answer candidates. Instead, we use the API to query the knowledge graph for entities using the entity name and specifying a list of types using the relation parts (i.e. People, Person, Celebrity from People.Person.Celebrity) from the relation in the (entity, relation) pair. There is no alternative to search for candidate answers with this specific API. Rather, if we wanted to search in the same fashion as before with triplets, we would need to convert the data in the Knowledge Graph to RDF form. However, this didn't seem feasible given vast size of the data we needed, lack of documentation, and lack of time.

6 Results

In between each step, we ran evaluation metrics to determine the accuracy of each step. We used these scores to determine how well each step was performing as well as to assess any serious deficiencies or errors in our model. The non-hand-crafted training and test data utilized in each step was sourced from the paper's project github[12].

6.1 Entity Linking

In the entity linking step, we decided to switch datasets to improve accuracy, so we will be discussing the effects of those changes and the implications of these results in the following sections. Our evaluation consisted of marking the presence of the correct entity in the entity ranking list generated by our Entity Linking step. As mentioned, we used sample questions drawn from the WebQuestions dataset (a total of 2875 questions) for the input to the Entity Linking step. Our first implementation used structured data in the form of a small subset of the Freebase as a KB. Our second implementation used the Google Knowledge Graph Search API to access a much larger knowledge graph (though structured differently from what was required in the paper).

6.1.1 With the Small Freebase KB

Isolated Model	Entity Linking Accuracy
Hand-crafted	**93.8%**
KBTrain_Data	26.6%

Table 2: Entity Linking Results with Small Freebase KB

Based on the accuracy for the hand-crafted set2, it is obvious that the entity linking model performs well in extracting correct entities. However, given that the questions are based on information present in the small KB, these findings are trivial. Thought, this did help, initially, in making sure the entire Entity Linking process was working together. The questions from the KBTrain_Data are not based off of the small KB, and are instead based off of data from web crawling web-search queries (i.e. most commonly asked questions). The accuracy result for this input dataset is, unsurprisingly, very low. This is due to the very small size of the Freebase entities were extracted from. Most of the entities mentioned in the questions were most likely not included in our KB. Therefore, our model would not be able to extract the correct entity.

We also noticed that the mention identification portion of this step needed some improvement as well. Our initial implementation considered words with POS tags starting with "N" (i.e. nouns, noun phrases, etc) as mentions. However, this would only extract single words and would not include whole phrases (i.e. "dog" vs "big dog"). Additionally, some words (usually names) would be considered as nouns if strung together with another word, but would be considered as a different POS on its own. A notable example involves entities referring to people by first and last names: "where is ruben rausing from?" After POS tagging the question looks like this: "where/WP is ruben/NN rausing/VB from?" According to our initial implementation, the only entity mentions extracted would be "where" and "ruben". We are missing an extremely vital piece of information, Ruben's last name, "rausing." Many other instances of this type of mention exclusion most likely retrieved entities with names that partially matched mentions. Our next implementation addresses this issue as well as the limited dataset problem.

6.1.2 With the Google Knowledge Graph API

Switching our dataset to the Google Knowledge Graph drastically increased the amount of information we had at our disposal. The API allowed us to access around 570 million entities and 18 billion facts. Our input data remains the same from the last implementation. This switch resulted in an improvement in the hand-crafted set and an extreme jump in accuracy for the KBTrain_Data. In fact, we were able to achieve a higher accuracy than the paper, which reported a **79.8%** accuracy for isolated entity linking. Access to the larger Google Knowledge Graph might have played a part in performing better than the paper. Looking at the results, we see that the increased dataset most definitely helped in our own model improvement. Entities that previously were not linked before are linked now because they are actually present in the KG.

Isolated Model	Entity Linking Accuracy
Hand-crafted	100.0%
KBTrain_Data	**86.4%**

Table 3: Entity Linking Results with Google Knowledge Graph

Additionally, we addressed the mention identification issue in this version. This time, we string together mentions as described before in our Implementation section. Rather than identifying only singleton mentions, we expand the POS tags we consider as part of mentions (i.e. adjectives, verbs, etc) and attach them to adjacent nouns (that they might be referring to). For the example highlighted in the previous implementation, "where is ruben rausing from?", we would be able to extract "where," "ruben," "rausing," and "ruben rausing" as mentions. There are both pros and cons to this change, however. While we are now able to identify the whole mention phrase (i.e. "ruben rausing"), we also introduce the potential to identify many non-relevant words as mentions. To combat this, we only accept nouns as singleton mentions. Our revisions to mention identification allow us to use more specific information to find relevant entities in the KG. This might have also accounted for some of the improvement in accuracy.

6.2 Relation Extraction

Here we present train and test results for our relation extractor. Following implementation steps discussed in section 5, the accuracy is surprisingly high given that the classifier models we chose are fairly simple and easy to train. In Table 4 we present results for a simple two layer neural networks of 800 cells in each layer, trained over 100 iterations with learning rate of 0.1. Our train and test set are both WebQuestions, with pruned subsets to be of classes with at least 30 samples in training set. The model reached 73.1% on test set, which is significantly higher than that of the original paper (45.9%). Majoy reason would be that since we specifically chose data with more samples, leading us to higher model performance naturally. Also this has proved to some extent that our chosen feature representations (syntactic and sentential features, dependencies, word vectors, etc) embed strong features of the underlying entity relations.

	Neural Networks	SVM	Random Forest
Train	**99.86%**	80.25%	46.83%
Test	**73.14%**	66.86%	44.29%

Table 4: Relation Extraction Results using Two Layer Neural Networks, SVM, Random Forest Classifiers on WebQuestions Dataset

We also trained and compared prediction performance across different types of classifiers, the simple 2layer neural networks seem to outperform both SVM and random forest classifiers, which is impressive.

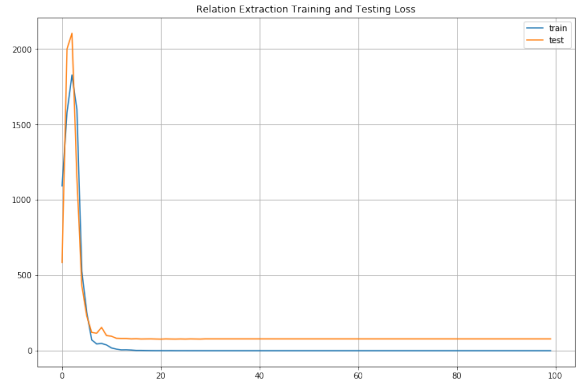

Figure 6: Train and Test loss of Two Layer Neural Networks with 800 cells each trained on WebQuestions Dataset

	precision	recall	f1-score	support
0	1.00	0.50	0.67	2
11	1.00	1.00	1.00	1
12	1.00	1.00	1.00	1
21	0.00	0.00	0.00	1
25	1.00	1.00	1.00	2
32	1.00	1.00	1.00	1
45	1.00	1.00	1.00	1
47	1.00	1.00	1.00	2
57	1.00	1.00	1.00	1
59	1.00	1.00	1.00	1
101	0.50	1.00	0.67	1
386	0.00	0.00	0.00	0
avg / total	0.89	0.86	0.86	14

Figure 7: Classification Report on top 14 Relation Classes for Simple Neural Network Model.

Figure 6 shows the convergences of train and test loss during iterations of the 2layer neural networks mode, We can see that the model quickly converged after 20 epochs, with train loss consistently slightly higher than test loss, indicating a healthy and efficient training process.

```
['people.deceased_person.cause_of_death',
 'people.person.spouse_s..people.marriage.spouse',
 'location.country.official_language',
 'people.person.education..education.education.institution',
 'sports.pro_athlete.teams..sports.sports_team_roster.team',
 'travel.travel_destination.tourist_attractions',
 'location.location.containedby',
 'people.person.profession',
 'film.film_character.portrayed_in_films..film.performance.actor',
 'film.actor.film..film.performance.film',
 'people.person.spouse_s..people.marriage.spouse',
 'people.person.profession',
 'location.location.containedby',
 'people.person.place_of_birth']
```

Figure 8: Top 14 Relation Classes for Simple Neural Network Model.

Figure 7 and Table 8 show the top classification report on the top 14 relation classes in the dataset, since they are the major relation types in the WebQuestions dataset. We see that

1. People related relation types gets high predictive performance from the model, maybe people relations are unique and distinct in the types of features they are usually related to, and thus easier for model to learn.
2. Location related relation types come closely to be the the second highest accurate class, again since location type questions are usually well defined, with clear entities

Overall, models for relation extraction part took a long time to train since dependency parser is slow (3 seconds / input), extracted features from inputs turn out to have significant representative power of the underlying relations, and a simple model of two-layer neural networks with 800 cells in each layer seems to perform well.

7 Conclusion and Future Work

In this paper, we focused on building a question answering (QA) model, more specifically, we reproduced QA models via entity linking and relation extraction introduced by paper from [4]. During implementations, we explored QA tasks with a knowledge-based relation extractor first with Stanford dependency parsing trees to parse questions into sub questions and find relations between input entities. We also explored using a more extensive knowledge graph (rather than a traditional RDF structured knowledge base) to improve entity linking accuracy and incorporating models with better predictive power, such as RNN structure or structured SVM, and to combine with external resource as textual evidence (such as Wikipedia) to achieve higher accuracy during the relation extraction process.

We completed a preliminary model with all functional pieces of a QA system, we have encouraging results on performance on all parts of the model and we are in the process of piping them together to make it one model. Google Knowledge Graph API [11] helped significantly with performance on entity linking, reaching at least 86% at this moment. Pruning dataset to have more well represented and evenly distributed dataset greatly helped with relation extraction model training. With features representations suggested by the paper [4], our relation extraction model is able to obtain 73.14% accuracy on the test set using a simple two-layer neural network model of 800 cells per layer. This is also a significant jump from the original paper, however we slightly pruned our training dataset to the classes with more number of samples to increase model learning. base our implementation on.[4] For future work, there are some parts of the process that we are are interested in experimenting with as well (i.e. using an entity linking tool, like S-MART, as in the paper rather than our own, etc.). We hope to use these variances from the paper as a means of comparison with their results as well as an opportunity to further explore this variant of QA so that it isn't limited by the coverage of any structured data set.

8 Code

The code for this project can be found in the following GitHub repository: `https://github.com/HujiaYuYoyo/CS224U_QA_project`.

References

[1] Jurafsky, Daniel and Martin, James H. *Speech and Language Processing Chapter 21.*

[2] Jurafsky, Daniel and Martin, James H. *Speech and Language Processing Chapter 28.* `http://web.stanford.edu/ jurafsky/`

[3] 2016. Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus and Richard Socher *Ask Me Anything: Dynamic Memory Networks for Natural Language Processing. arXiv:1506.07285.*

[4] Kun Xu, Siva Reddy, Yansong Feng, Songfang Huang, and Dongyan Zhao. 2016. *Question Answering on Freebase via Relation Extraction and Textual Evi-*

dence. In Proceedings of the Association for Computational Linguistics (ACL 2016).

[5] Weston, J., Bordes, A., Chopra, S. & Mikolov, T. *Towards AI-complete question answering: a set of prerequisite toy tasks.*

[6] Graves, A., Wayne, G.& Danihelka, I. *Neural Turing machines.*

[7] Weston, J., Chopra, S., and Bordes, A *Memory networks. In ICLR, 2015b.*

[8] Stanford NLP Group *Stanford Question Answering Dataset (SQuAD)* https://rajpurkar.github.io

[9] Facebook research *The bAbI project* https://research.fb.com/downloads/babi/

[10] Stanford NLP Group, The Stanford Parser: A statistical parser https://nlp.stanford.edu/software

[11] Google Knowledge Graph Search API https://developers.google.com

[12] Github User: syxu828 *QuestionAnsweringOverFB* https://github.com/syxu828