The Limited Effectiveness of Neural Networks for Simple Question Answering on Knowledge Graphs

by

Salman Mohammed

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2017

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Simple factoid question answering (QA) is a task, where the questions can be answered by looking up a single fact in the knowledge base (KB). However, this QA task is difficult, since retrieving a single supporting fact involves searching many alternatives given a query expressed in natural language. We use a retrieval-based approach to QA. We decompose the problem into four sub-problems: entity detection, entity linking, relation prediction, and evidence integration. Entity detection and linking rely on detecting the entities in a question and linking them to the candidate entities in the KB. Relation prediction classifies a question as one of the relation types in the KB. Finally, evidence integration combines scores from entity linking and relation prediction to predict an (entity, relation) pair that answers the question. Most of the research community has explored complex neural network architectures for this task without establishing baselines to compare the results with 'non-neural-network' approaches. We explore several different models for entity detection and relation prediction; a few different scoring functions for entity linking and evidence integration. Our findings show that deep learning does help for the QA task, but not as much as the research community has portrayed it to be. We also present two simple yet very competitive baselines: one based on a simple neural network architecture and one that does not use any neural networks.

Acknowledgements

First and foremost, I would like to extend my utmost gratitude to my advisor, Dr. Jimmy Lin, for his continuous support throughout my graduate studies and research. During the first few months of my graduate studies, Jimmy's mentorship and guidance enabled me to focus my efforts towards my current research. His passion for work and effervescent attitude inspired me to pursue my research, culminating in this thesis paper. As an advisor, Jimmy has provided key insights and research direction, always adding new perspectives and challenging me to improve upon my existing work. More importantly, however, he has been an ideal mentor and a personal friend.

I would also like to direct my sincere appreciation for my colleague, Peng Shi, who has been an excellent collaborator and a joy to work with. Most of our work at Data Systems is collaborative in nature, and this thesis presents findings from the collborative work conducted by Peng, Jimmy, and I.

My sincerest thanks are also reserved for the rest of the thesis committe - Dr. Ihab Ilyas and Dr. Gordon Cormack - for reading my thesis and providing insightful feedback. I have had the opportunity to know both of them personally during my time at Waterloo, and I consider myself extremely lucky to have them as my readers. In addition, I would like to extend my gratitude to my colleagues, Royal Sequeira and Michael Tu, for proof-reading my thesis and providing thoughtful comments and opinions.

Finally, I would like to thank my roommate Ahmed Jamal for proof-reading my paper, and other friends and family for providing moral support during times of need. Graduate studies can often be challenging but I could not be happier to have such a powerful source of support and encouragement from the loved ones around me.

Dedication

This thesis is dedicated to my parents
for their constant love and support throughout my life.

Table of Contents

List of Tables **ix**

List of Figures **xi**

1 Introduction **1**

1.1 Problem Definition 1
1.2 Challenges 2
1.3 Approach 2
1.4 Contributions 4
1.5 Thesis Organization 5

2 Background & Related Work **6**

2.1 Fully Connected Networks 6
2.2 Word Embeddings 7
2.3 Convolutional Neural Network 7
2.4 Recurrent Neural Network 9
2.5 Memory Networks 10
2.6 Attention-based Encoder-Decoder 12
2.7 Conditional Probabilistic Framework 13
2.8 CNN with Attentive Max-Pooling 13
2.9 Improved Relation Detection with Hierarchical LSTM 14
2.10 Retrieval-based Approach with RNNs 15

3 Dataset and Code **17**

3.1 Dataset . . . 17

3.2 Dataset for entity detection . . . 19

3.3 Code . . . 20

4 Method and Results **21**

4.1 Entity Detection . . . 21

4.1.1 RNN . . . 21

4.1.2 CRF . . . 24

4.1.3 Results . . . 24

4.2 Entity Linking . . . 25

4.2.1 Fuzzy matching . . . 25

4.2.2 Inverse document frequency (idf) . . . 26

4.2.3 Weighted combination . . . 26

4.2.4 Results . . . 27

4.3 Relation Prediction . . . 27

4.3.1 RNN . . . 28

4.3.2 CNN . . . 29

4.3.3 Logistic Regression . . . 30

4.3.4 Results . . . 31

4.4 Evidence Integration . . . 32

4.4.1 Entity node indegree . . . 32

4.4.2 Results . . . 33

4.4.3 Final Results . . . 34

5 Error Analysis **36**

5.1 Entity Linking . . . 36

5.2 Evidence Integration . . . 37

6 Conclusion **41**

6.1 Future Work 41

6.2 Summary 42

References **43**

List of Tables

3.1 A few samples of questions in the SimpleQuestions validation set 18
3.2 A few samples from Freebase 18
3.3 A few samples of entities mapped to their names in Freebase 18
3.4 Statistics of the Freebase subsets provided with the SimpleQuestions dataset 19
3.5 A few samples from the back-projected entity detection dataset 20

4.1 Some important parameters for the RNN that were tuned on the validation set 23
4.2 Results for the entity detection task 24
4.3 Comparison of different similarity functions for entity linking on the validation set 27
4.4 Results for relation prediction 32
4.5 Results comparing the effect of secondary sorting on entity node indegree on the validation set when top 50 entities were crossed with top 5 relations 33
4.6 Evidence integration results on the test set when different models and number of top entities and relations are considered 34
4.7 Comparison with state-of-the art models for this task on the SimpleQuestions test set 35

5.1 The cumulative density sum of exact match counts with the 'gold' query text on the validation set 37
5.2 Statistics of number of exact, partial matches with retrieval rate at different hits for the 'squery' function with the 'gold' query text on the validation set 37

5.3 Statistics of errors of validation set questions that were retrieved in the second position but not the top 38
5.4 Statistics of errors of validation set questions that were retrieved in the third position but not in the first or second position 38
5.5 Sample questions that were retrieved in the second position with the cause and reason of errors in the top position 40

List of Figures

1.1 A block diagram of our QA system pipeline 3
1.2 A small snippet of the Freebase graph 4

2.1 Example of convolutional network in sentiment analysis [6] 8
2.2 An unrolled recurrent neural network [22] 9
2.3 Structure of a deep bi-directional RNN [7] 10
2.4 Structure of a single hop end-to-end memory networks from Sumit Chopra's slides at NIPS 2015 [5] 11
2.5 Encoder-decoder architecture from Golub and He[12] 12
2.6 Fact selection system architecture from Yin et al. [31] 14
2.7 The HR Bi-LSTM model used for relation detection by Yu et al. [32] . . . 15

4.1 Illustrating the different layers of the entity detection model 22
4.2 Illustrating the different layers of the relation prediction model 28
4.3 CNN model architecture from Kim [15] 30

Chapter 1

Introduction

1.1 Problem Definition

Natural language processing (NLP) is one of the major branches in artificial intelligence (AI) and it focuses on computer understanding and manipulation of human language. Within NLP, the task of question-answering (QA) refers to building systems that automatically answer questions expressed in a natural language. There are two major paradigms of question answering: QA over free-text and QA using a knowledge base (KB). QA over free-text aims at providing the answers to questions formulated in natural language, without restriction of domain, for example, the Web. On the other hand, QA using a knowledge base provides answers to questions by looking up facts that are already stored in a knowledge base so the task is essentially to covert the natural language question into a database query.

This thesis tries to tackle the problem of question answering over knowledge base; specifically simple factoid question answering. Simple factoid QA refers to questions that can be answered by looking up a single fact in the knowledge base. There are large knowledge bases such as Freebase, which contain consolidated knowledge stored as facts, and extracted from different sources such as free text, tables in webpages or collaborative input [4]. For example, the question "Where was Sasha Vujacic born" can be answered by looking up a single fact/triple in Freebase - ('Sasha Vujacic', place of birth, 'Maribor, Slovenia'). Factoid questions are the most common type of questions observed in various community QA sites [31]. There is also more complex QA that may involve looking up more than one fact in the KB to get to the correct answer but we will not be looking at these questions. For the purpose of this thesis, we will be looking at factoid question

answering over Freebase on the SimpleQuestions dataset, which is a dataset consisting of questions that can be answered with a single fact.

1.2 Challenges

Although this task only involves retrieving a single fact in the KB, it is quite challenging in reality for a couple of reasons. The KB contains millions of entities and queries expressed in natural language can retrieve many candidate entities that are hard to distinguish. For example, the entity in the question, "Which city is JFK located in?" can be extremely hard to find in the KB because there are different nodes corresponding to 'JFK' - airport, film, person, etc. It is also challenging to detect the relation type of a question so that the correct predicate can be looked up in the KB. There exist different predicates in the KB referring to a music genre and a film genre but they are difficult to distinguish from a question phrased in English. For example, the question 'Which genre is teri meri kahaani under' is actually referring to a film genre.

1.3 Approach

Factoid question answering relies on finding a single fact/triple from the knowledge base that answers the question. We employ a retrieval-based approach to the factoid QA task, where we decompose the task into four main sub-problems: entity detection, entity linking, relation prediction, and evidence integration. Figure 1.1 shows the block diagram of our question-answering system pipeline.

- **Entity Detection:** Given a question in natural language, this module returns the entity mention in the text which is used as query in the linking phase to search for the entity node in the KB. This problem is formulated as a sequence-to-sequence (seq2seq) task similar to named entity recognition, where the task is to tag the entity words in the question.
- **Entity Linking:** Given a query, i.e., the entity mention in the text from the detection phase, this module tries to link the question to candidate entity nodes in the knowledge base using an inverted index. This is a search problem where the goal is to retrieve top candidate entities from the index whose name matches the query.

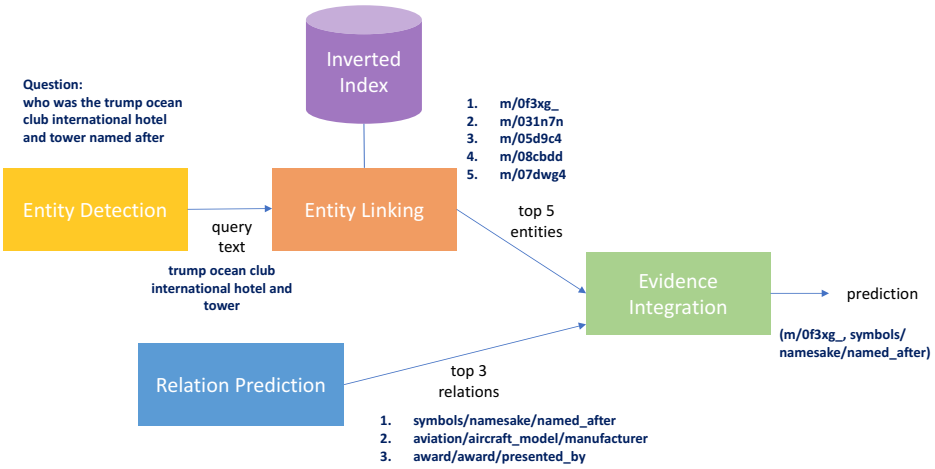

Figure 1.1: A block diagram of our QA system pipeline

- **Relation Prediction:** Given a question in natural language, this module returns the top candidate relations. This problem is formulated as a large relation classification task, where the top relations are selected based on the probability score assigned to each relation class.
- **Evidence Integration:** Given the top candidate entities and the top candidate relations, this module combines their score to make the top (entity, relation) prediction. We also evaluate the performance based on metrics such as accuracy and retrieval-at-k.

Example: Figure 1.1 shows a walk-through example of our system for the question, "who was the trump ocean club international hotel and tower named after". The entity detection phase returns the entity mention as the query - 'trump ocean club international hotel and tower'. This query still needs to be linked to the correct entity machine identifier (MID) in the KB. This is done by creating an inverted index for entities in the Freebase subset. The index is searched using the query and candidate entities are ranked using a relevance function to retrieve top candidate entities. Relation prediction on the question also retrieves the top candidate relations. Finally, the top candidate entities and relations are cross-linked and the scores from entity linking and relation prediction are combined to re-rank the possible candidate (entity, relation) pairs. The top ranked pair is selected as the prediction and compared against the ground-truth which in this case is (m/0f3xg_, symbols/namesake/named_after). As shown in Figure 1.2, this tuple can be easily looked

up in Freebase to get the entity that contains the answer to the question and we can that map that entity to its name to get the actual answer - 'donald trump'.

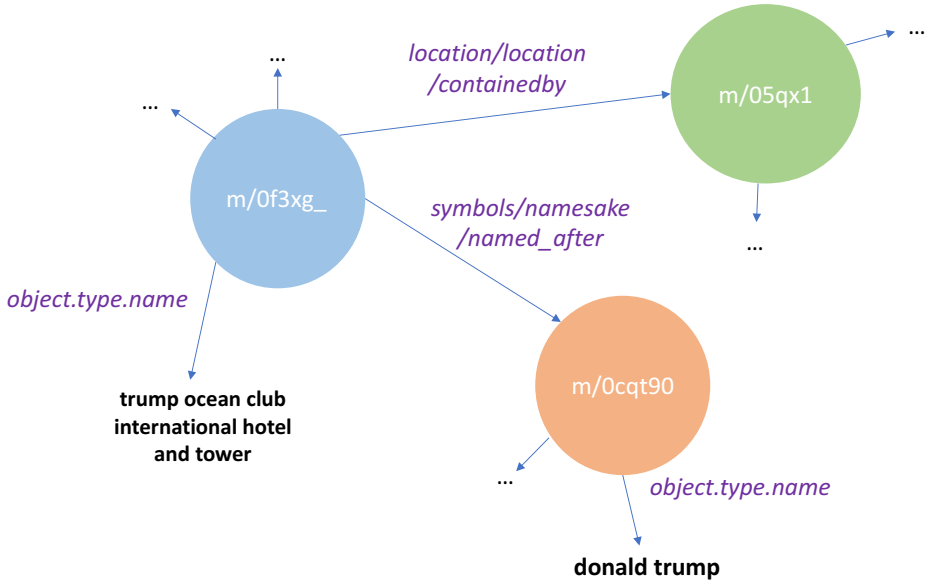

Figure 1.2: A small snippet of the Freebase graph

1.4 Contributions

There have been two main approaches to factoid QA in the recent past. The first is a more traditional approach that involves converting the question into a linguistically motivated representation (e.g., using syntactic and/or semantic parsing), which simplifies the problem into finding the answer that best fits this representation [30] [1]. More recent approaches involve retrieving candidate answers from the KB using distributed representations of both question and answer learned automatically in a data-driven manner by neural networks (NNs) [31] [12] [27]. Different neural network architectures have been used in the literature for this task - recurrent NNs, convolutional NNs, memory networks, attention-based networks, etc.

The main contribution of this thesis is to better understand the task of question answering using a KB. There is a lack of strong baseline scores in the literature to compare

the results of complex deep learning models. Almost all the work compares their models to baselines based on random selection. There is also a lack of error analysis done to understand where our models are doing well or where they are under-performing. Our work started out by replicating the work done by Ture and Jojic [27], who reported state-of-the-art accuracy in this task and making the code open-source. Our experiments and analysis show us where the main complexity of the factoid QA task comes from and where the deep learning models actually help. Finally, we report a few simple models (with and without the use of neural networks) that might serve as baselines for the factoid QA research community. These simple baseline results reveal that deep learning models do help but its success is over-hyped.

1.5 Thesis Organization

Chapter 2 reviews some of the background concepts regarding neural networks and gives an overview of related works on factoid question answering in the literature. Chapter 3 presents the datasets and the code for this work. Chapter 4 describes our method and the experimental results. Chapter 5 describes the error analysis that we performed. Chapter 6 concludes this thesis.

Chapter 2

Background & Related Work

Most of the work done in the past decade was on the WebQuestions dataset [1] and they are based on semantic parsing, where a question is mapped to its formal meaning representation (e.g., logical form) and then translated to a knowledge base (KB) query [1] [2] [25]. Although the WebQuestions dataset contained mostly factoid questions, 32% of the questions were complex and could not be answered with a single relation [29]. The SimpleQuestions dataset was recently introduced by Bordes et al. and it only contains simple factoid questions. This dataset (75,910 training set questions) is also much larger than the WebQuestions dataset (3,778 training set questions) and was created with the intention of testing out the performance of neural networks on this task [4]. Most approaches on this dataset have used neural networks in various ways and we will briefly describe some of the approaches in the literature that is related to our work.

First, we will briefly touch on some topics and concepts on neural networks (Section 2.1 - 2.4) that are used in the work covered in this thesis. These components are being briefly introduced to the reader but they do not provide a detailed description. Please consult the references or books on the material if more information is needed.

2.1 Fully Connected Networks

In fully connected networks (FCNs), each node in a layer of the neural network is connected to all the nodes in the previous layer. There are weights/parameters for each layer that are initially randomly initialized, and perform an affine transformation to the input features. This is followed by a non-linear transformation such as hyperbolic tangent (tanh)

or rectified linear units (ReLU) to allow the network to approximate complex functions. FCNs have a fixed input and output size and are usually used for classification tasks to output a value for each class.

2.2 Word Embeddings

A word embedding, $W : w \rightarrow \mathbb{R}^d$ is a parameterized function that maps a word, w, to a d-dimensional vector space. The function is a lookup table that indexes a row, n, in the matrix, θ, for each word [21]:

$$W_\theta(w_n) = \theta_n$$

Word embeddings are learned in an unsupervised fashion over a large corpus of text. They try to encapsulate the semantic meaning of words by exploiting the idea that words with similar semantics appear in similar contexts. A feed-forward neural network with one hidden layer takes words as inputs from a vocabulary, embeds them into a lower dimensional space and tries to make a prediction in the output layer. The weights are fine-tuned through back-propagation. The weights of the different layers are combined in different ways to output the word embeddings [21]. In NLP tasks, the words of the sentence represents the input sequence where each input is a *word embedding*. GloVe [23] and word2vec [19] are two popular word embedding methods. For our task, we used pre-trained GloVe word vectors from their official website[1].

2.3 Convolutional Neural Network

In convolutional neural networks (CNNs), filters are used to convolve over the input layer producing local connections, where each region of the input is connected to a neuron in the output. The filters share parameters and are randomly initialized and the values are tuned using back-propagation.

There are three main types of layers stacked to build a CNN architecture - convolutional layers, pooling layers and fully- connected layers. Figure 2.1 shows a CNN architecture used for sentiment analysis of movie reviews. The matrix contains the word embeddings for each word in the sentence in its rows. The convolutional layer computes features based on the local connections using filters of different height. We slide the filters over the

[1]https:// nlp.stanford.edu/projects/glove/

word embeddings so the width remains the same. The features are then passed through an activation function (e.g. ReLU) to provide non-linearity. The pooling layer performs a downsampling operation to pick out a signal. There are different types of pooling operations but max-pooling is the most popular where the maximum value from the convolutional feature is selected [6]. Pooling can be thought of as providing 'location invariance'. For example, we expect max-pooling to pick out the phrase 'like this movie' as a strong positive signal for the sentiment analysis task and the location of the text is not of much importance to make a classification. Finally, fully connected layers are used using the features from the pooling layer as inputs to make the final classification for the movie review. The softmax function in the final layer normalizes the scores into a probability distribution for the two classes: positive and negative.

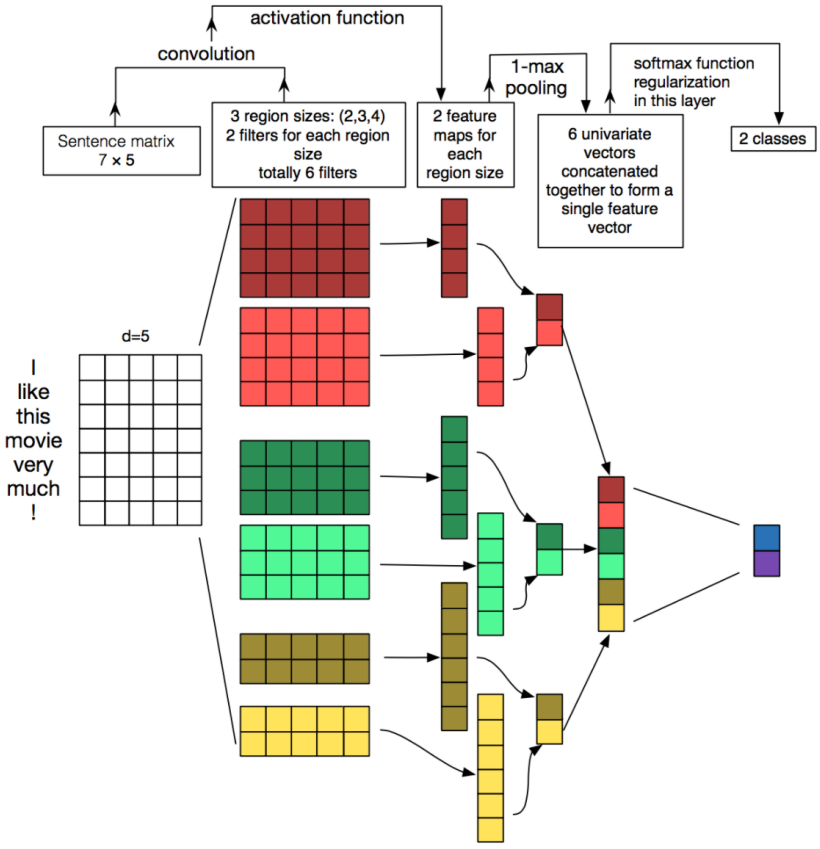

Figure 2.1: Example of convolutional network in sentiment analysis [6]

CNNs compute local features based on word embeddings and learned filters, similar to

the n-grams representation, but they are more efficient in terms of size. CNNs require fixed size inputs and outputs, so a few tricks have to be applied to adapt them into NLP tasks such as padding the sentences to make them of same length. CNNs can also be trained much faster than FCNs since they have fewer parameters.

2.4 Recurrent Neural Network

In many NLP tasks, we are required to capture a good sentence representation and sentences can be thought of as a sequence of words. However, traditional neural network accepts only a fixed size input and output and they have no sense of 'state' to reason about previous events to make a decision. Recurrent neural network (RNN) can operate over sequential input and produce sequential output depending on the task. For example, named entity recognition has a many-to-many mapping whereas sentiment classification has a many-to-one mapping. RNNs have become the de-facto baseline model in most NLP tasks in recent years.

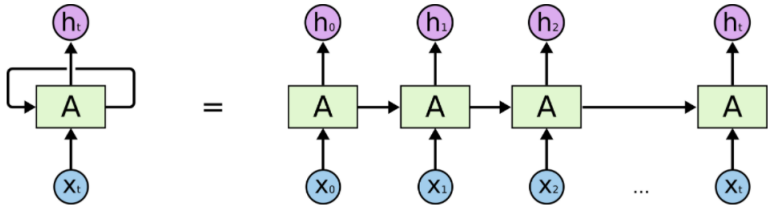

Figure 2.2: An unrolled recurrent neural network [22]

Figure 2.2 shows a vanilla RNN with an input sequence $x_0, x_1, ..., x_t$ and a hidden layer output sequence $h_0, h_1, ..., h_t$. For each time step t the RNN cell computes a hidden state, h_t, based on the current input, x_t, and the previous cell state, h_{t-1}, parametrized by shared weights across time sequence, W^{hx} and W^{hh}.

$$h_t = f(W^{hx} \cdot x_t + W^{hh} \cdot h_{t-1})$$

There are usually fully connected layers that take the hidden layer output from the RNN and make predictions (not shown in the figure): $y_t = g(W^{hy}.h_t)$. The input sequence represents a *word embedding* for each word in the sentence. Predictions can be made on each hidden layer output $(h_0, h_1, ..., h_t)$ for sequence-to-sequence tasks or on the last hidden layer output, h_t, that can be thought of as the 'sentence embedding' since its state is

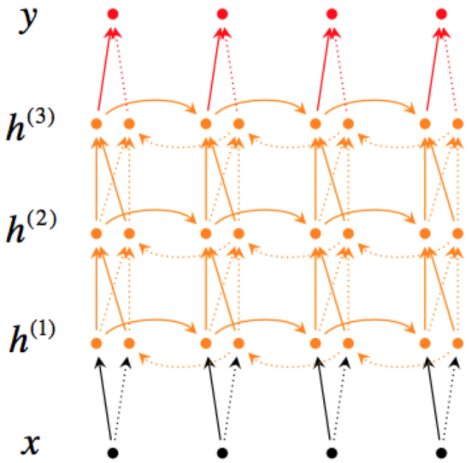

Figure 2.3: Structure of a deep bi-directional RNN [7]

influenced by all previous inputs [22]. RNNs can have a bi-directional structure to process the sequence in the other direction and it is also possible to go deeper: stack more layers on the network to give it more expressive power. Figure 2.3 shows the structure of a deep 3-layer bi-directional RNN.

Vanilla RNNs usually have problems capturing long-term dependencies due to a vanishing or exploding gradient problem while training these networks. These problems arise due to the product of the same weights multiple times during back-propagation. In practice, variants of RNNs with gated cells such as long short-term memory (LSTM) [14] and gated recurrent units (GRU) [8] are used. These variants have different modifications inside the cell, A (as shown in Figure 2.2). They have mechanism to forget irrelevant parts of the previous state, selectively update the cell and output certain parts of the state. These complex cells allow the LSTM and GRU to capture long-term dependencies better than vanilla RNNs.

2.5 Memory Networks

Memory networks [26] are a class of models which combine large memory with learning component which can read and write to it. They incorporate reasoning via attention over memory and the model framework is flexible enough to store rich representations of input in memory. Attention refers to the model outputting a distribution that describes how the model spread out the amount it cares about different memory positions [20]. In the

task of factoid QA, the model scales up to store and read the entire knowledge base in memory. The whole architecture is differentiable and only requires supervision at the final output, i.e. the model is initialized with random weights and these weights are learned by back-propagating the loss from the predicted answer.

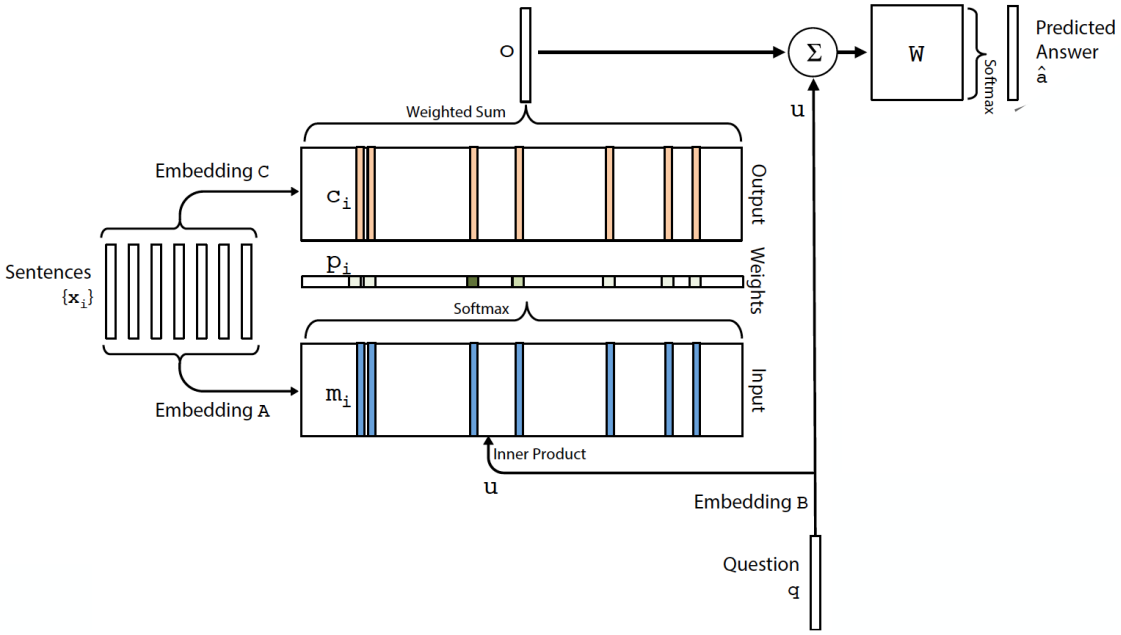

Figure 2.4: Structure of a single hop end-to-end memory networks from Sumit Chopra's slides at NIPS 2015 [5]

Figure 2.4 shows the structure of a single hop end-to-end memory network [26]. There are four modules: input (I), generalization (G), output (O), response (R). For factoid QA [4] , the input module pre-processes the question and Freebase and loads up the memory with the facts. The generalization module is used to extend the memory with Reverb facts. The output module performs the memory lookups via attention mechanism given the input to return a supporting fact to answer the given question.

$$p_i = softmax(u^T m_i)$$

$$o = \sum_i p_i c_i$$

$$\hat{a} = softmax(W(o + u))$$

To avoid scoring all the stored facts, they perform an entity linking step to generate a small set of candidate facts. The supporting fact is the candidate fact that is most similar

to the question according to an embedding model. The response module post-processes the result of the output module to compute the intended answer but in the factoid QA case, it just returns the entity and relation of the selected fact.

2.6 Attention-based Encoder-Decoder

Figure 2.5 shows the character-level attention-based encoder-decoder framework proposed by Golub and He [12].

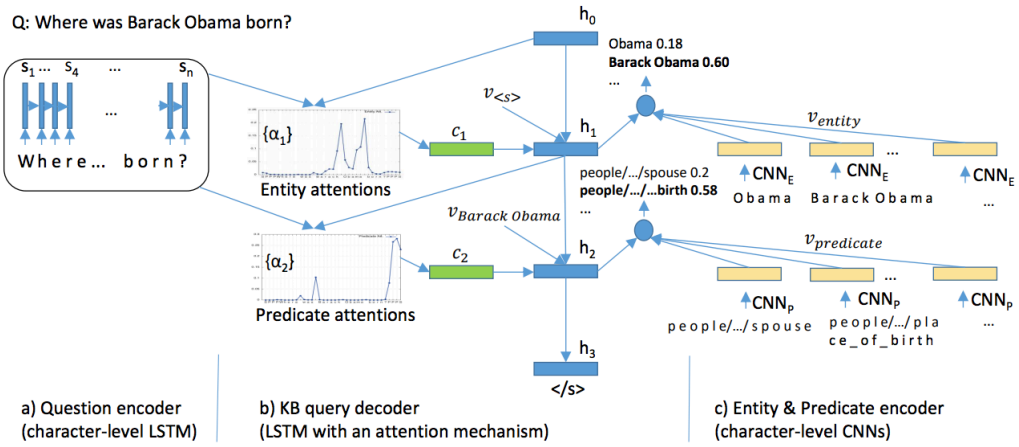

Figure 2.5: Encoder-decoder architecture from Golub and He[12]

Golub and He [12] encode the question, all the entities and relations/predicates in Freebase into embeddings using character-level LSTMs. All the embeddings are randomly initialized and jointly learned during end-to-end training to directly optimize the likelihood of generating the correct KB query. To predict the entity and relation for a question, an LSTM decoder with attention and pairwise semantic relevance function is used. The decoder is initialized with zero vector (h_0) and makes the prediction in two time-steps. At the first time step, the decoder performs entity attentions over the encoded question and selects the most likely entity based on the relevance function. In the second time step, it performs predicate attentions over the encoded question and uses the embedding of the entity (h_1) to predict the most likely relation.

2.7 Conditional Probabilistic Framework

Dai et al. [9] combine a unified conditional probabilistic framework with deep recurrent neural networks and neural embeddings. They propose a conditional factoid factorization to predict the most likely entity, e, and relation, r, pair, given the question, q:

$$p(e, r|q) = p(r|q).p(e|q, r)$$

Neural networks are used to parametrize $p(r|q)$ and $p(e|q, r)$. They use a GRU for the relation network to output the most likely relation for a question. The relation is factorized before the entity since there are fewer relations to consider. They perform entity detection with a bi-directional GRU (Bi-GRU) followed by a linear-chain conditional random field (CRF) to find the entity mention in the question. Candidate entities are retrieved based on n-gram matches with entity name and if the predicted relation exists for that entity in the KB. Out of the cadidate entities, another GRU is used to predict the entity given the question and the relation. However, the embeddings for entities are not randomly initialized but with type vector representation, which is a bag of words representation for all the predefined types in the KB.

2.8 CNN with Attentive Max-Pooling

Yin et al. [31] performed entity detection using a Bi-GRU CRF model similar to Dai et al. [9] to find the entity mention in the question. They split the question into entity mention and question pattern where the entity mention in the question is masked out with a token. Based on the entity mention and the names of the entities in the KB, entity linking is performed based on lexical matching to retrieve a set of candidate entities, C_e. Then, all the triples/facts in the KB that include the entities from C_e are retrieved to form a fact pool. The entity and predicate pair from all the facts in the pool are matched with the entity mention and the question pattern and the highest scored pair is chosen as the prediction to the question.

Figure 2.6 shows the fact selection system architecture, where a candidate fact from the pool is scored. The entity mention from Freebase and the question is encoded using a character-level CNN and their similarity is calculated using the cosine score, m_e. The relation pattern is encoded using a separate word-level CNN and their cosine similarity, m_r, is computed. Both the CNNs also implement attentive max-pooling, which re-weights

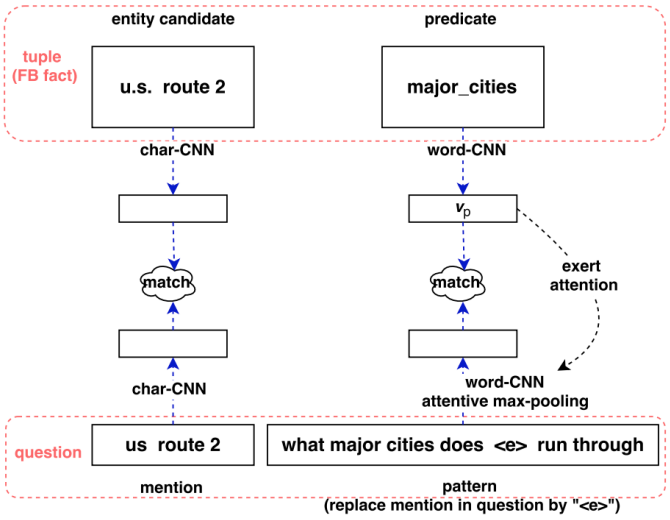

Figure 2.6: Fact selection system architecture from Yin et al. [31]

the features based on the cosine similarity of the embeddings. The predicted entity and relation pair is taken from the fact with the highest overall ranking score, s_t:

$$s_t = m_e + m_r + s_e$$

where s_e is the score from the entity linking phase. During training phase, there was 1 positive example and 99 negative examples were sampled from the fact pool to avoid including too many negative examples.

2.9 Improved Relation Detection with Hierarchical LSTM

Following the work of the attentive CNN model, Yu et al. [32] used a hierarchical residual Bi-LSTM (HR Bi-LSTM) on the raw question text to rank all the relations in the KB that are associated with the entities in C_e. Figure 2.7 shows the HR Bi-LSTM model used for relation detection. The relations were encoded by concatenating the entire token at the relation level and also by splitting them into words at the word-level.

The relation detection scores are used to re-rank and prune candidate entities. Then, they perform relation detection on the formatted question text where the entity mention in

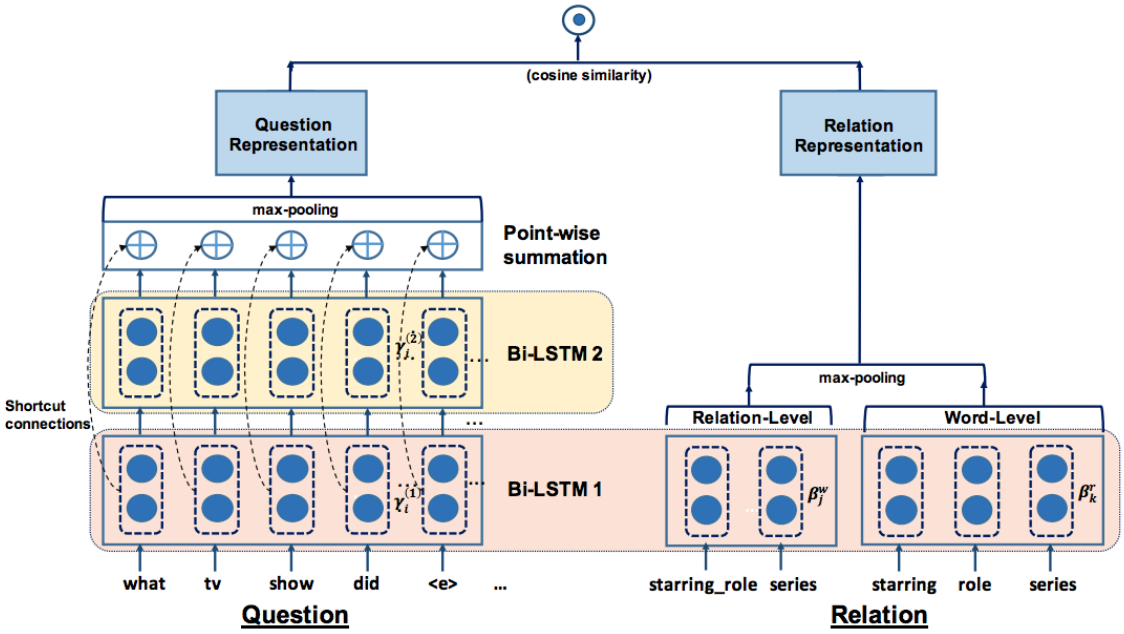

Figure 2.7: The HR Bi-LSTM model used for relation detection by Yu et al. [32]

the question is masked with a token. Finally, they combine the scores from the entity re-ranking and the relation detection on formatted question text to rank the candidate (entity, relation) pairs and the highest ranked pair is the predicted answer. The analysis shows that their hierarchical architecture helps the model learn different levels of abstraction and reduces the over-fitting problem.

2.10 Retrieval-based Approach with RNNs

Ture and Jojic [27] formulates the task as two machine learning problems: detecting the entities in the question, and classifying the question as one of the relation types in the KB. They perform entity linking by using the entity mention as the query to a search problem. They use indexes to efficiently search for candidate entities that contain n-grams of the query and use tf-idf scoring to rank the entities. They also a few tricks such as early termination in search and relation correction to prune the candidate entity space. Ture and Jojic experiment with different RNNs and found that Bi-LSTMs performed best for entity detection and Bi-GRUs performed best for the relation prediction task. Finally,

the top candidate entity (from the linking phase) and the top relation (from the relation prediction task) pair is selected as the prediction for the question.

Chapter 3

Dataset and Code

3.1 Dataset

We demonstrate the effectiveness of our approach on the SimpleQuestions dataset [4], which is a part of the bAbI project from Facebook AI Research[1] that was created for better text understanding and reasoning. The dataset contains factoid questions (also referred to as simple questions) and the corresponding ground-truth fact. Each line in the dataset contains the question text and (subject, predicate, object) triple from Freebase that answers the question. However, the dataset does not identify the entities in the question, it only provides the Freebase identifier (MID). The dataset is split into training (75,910 questions), validation (10,845 questions) and test sets (21,687 questions) and also provides two subsets of Freebase. There are 1,837 unique relation types in the dataset. Table 3.1 shows a few samples of questions in the SimpleQuestions validation set.

In first-order factoid question answering, questions can be answered by looking up a single subject-predicate pair in the knowledge base, in our case - Freebase [3]. Freebase is a structured knowledge base in which entities are connected by predefined predicates or "relations". All predicates are directional, connecting from the subject to the object. Table 3.2 shows a few samples of Freebase triples. A triple (subject, predicate, object) describes a fact; e.g., (m/07f3jg, people/person/place_of_birth, m/0565dl) [31]. The predicate is self explanatory - it refers to the place of birth.

However, the subject and object entities have to be mapped to their names to get a meaningful fact in real-life. Table 3.3 shows a few samples of the entities mapped to their

[1] https://research.fb.com/downloads/babi/

Table 3.1: A few samples of questions in the SimpleQuestions validation set

Entity	Relation	Answer	Question
m/0f3xg_	symbols/namesake /named_after	m/0cqt90	Who was the trump ocean club international hotel and tower named after
m/07f3jg	people/person /place_of_birth	m/0565d	where was sasha vujai born
m/02p_vkx	people/person /profession	m/04gc2	What was Seymour Parker Gilbert's profession?

Table 3.2: A few samples from Freebase

Subject	Predicate	Object
m/07f3jg	sports/pro_athlete/teams	m/09wc0v
m/07f3jg	people/person/place_of_birth	m/0565d
m/02p_vkx	people/person/place_of_birth	m/0xmk3
m/02p_vkx	people/person/gender	m/05zppz

names in Freebase using the predicate *type.object.name* or *common.topic.alias*. In this example, the subject entity (m/07f3jg) refers to the basketball player, Sasha Vujacic and the object entity (m/0565dl) refers to his birth place - Maribor, Slovenia.

Table 3.3: A few samples of entities mapped to their names in Freebase

Entity	Name
m/0f3xg_	trump ocean club international hotel and tower
m/07f3jg	sasha vujacic
m/02p_vkx	seymour parker gilbert
m/0565d	slovenia, maribor
m/05zppz	man

The SimpleQuestions dataset also provides two subsets of Freebase to make the task easier. Table 3.4 shows the statistics of the Freebase subsets. In comparison, the entire Freebase data dump is much larger and contains 1.9 billion triples[2]. In this paper, we have

[2]https://developers.google.com/freebase/

only reported the results with the 2M-subset of Freebase.

Table 3.4: Statistics of the Freebase subsets provided with the SimpleQuestions dataset

	2M-subset	**5M-subset**
# entities	2,150,604	4,904,397
# relations	6,701	7,523
# triples	14,180,937	22,441,880

Text pre-processing: we perform minimal text processing to the questions and the entity names in Freebase. The test is lowercased, stripped off accents and tokenized using the NLTK Penn Treebank tokenizer. For example, the text 'Sasha Vujačić' gets converted to 'sasha vujacic' and the text 'What was Seymour Parker Gilbert's profession?' gets converted to 'what was seymour parker gilbert 's profession ?'.

3.2 Dataset for entity detection

As mentioned before, the SimpleQuestions dataset does not provide the ground truth for the entities in the question so we had to back-project the name of the entity MID from Freebase to create a training set. We used the names file (FB5M.name.txt) from Dai et al. [9][3] that contains the name (*type.object.name*) and alias (*common.topic.alias*) predicate for the entities from Freebase. We split the question and the entity name into tokens. First we try to do exact matching on the tokens but if there are no matches found, we do fuzzy matching to assign entity labels to a question token that match the entity name token with low Levenshtein distance. Table 3.5 shows a few samples from the entity detection dataset.

Following from the previous example, back-projection maps the question "where was sasha vujacic born" to the tags [O O I I O] where I represents an entity word and O represents a non-entity word. Since 'sasha vujacic' is the name of the entity in question, the words 'sasha' and 'vujacic' are marked as entity words. Creating the training set for entity detection is a difficult task since there is no consistent way to map an entity MID to a name. There are different predicates that lead to a name literal for an entity in Freebase. In our case, we could not find the name or alias mapping for 611 entities in Freebase, so we skipped those questions.

[3] https://github.com/zihangdai/cfo

Table 3.5: A few samples from the back-projected entity detection dataset

Question	Entity Labels
who was the trump ocean club international hotel and tower named after	O O O I I I I I I I O O
where was sasha vujacic born	O O I I O
what was seymour parker gilbert 's profession ?	O O I I I O O

3.3 Code

Our code is open-source on GitHub[4]. The implementation uses Python 3, the deep learning framework, PyTorch and the text processing package, torchtext.

[4]https://github.com/salman1993/BuboQA

Chapter 4

Method and Results

We briefly discussed our approach in the introduction chapter. Figure 1.1 showed how the different components fit together in our question-answering system pipeline. In this chapter, we will dive deeper into each of these components and look at the different models or techniques that we have experimented with and also provide the results to compare the performance of each method.

4.1 Entity Detection

In this section, we describe how we do entity detection - the process of detecting entities in the question so that we can query the index later to link the correct entity MID. The query is formed in two steps: first, tag each word in the question as entity or not and second, combine the predicted entity words to form a query. The entity detection task is a seq2seq machine learning task, where both the input and output represents a sequence, for example, named entity recognition. We have experimented with two models for this task: recurrent neural networks, and conditional random field (CRF), as implemented in Stanford Named Entity Recognizer (NER).[1]

4.1.1 RNN

A RNN naturally models dependencies among all words in the input sequence (i.e. question) without any feature engineering but suffer from long-term dependency problem. We

[1] https://nlp.stanford.edu/software/CRF-NER.shtml

have used variants of RNNs - LSTM and GRU, which are better at handling long term dependencies. They can learn contextual features that are useful for the task, while learning to ignore irrelevant parts. Figure 4.1 illustrates the different layers of the entity detection model for the question (after pre-processing): 'where was sasha vujacic born'. The beige colored boxes represent the embedding layer where each input word is mapped to a 300-dimensional GloVe embedding[2]. The green colored boxes represent the recurrent layer that combines the current word embedding with the hidden layer representation from the previous word to compute the hidden layer representation for the current word. Finally, the blue colored boxes represent the fully-connected (FC) layers that project the hidden representation of the final recurrent layer to the output space (entity, not-entity).

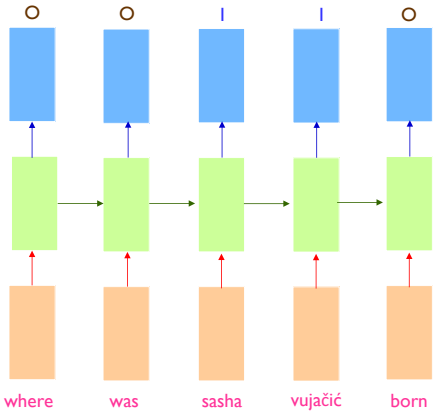

Figure 4.1: Illustrating the different layers of the entity detection model

We experimented with different model architectures and hyper-parameters and found that 2 layer bi-directional LSTM worked the best. Table 4.1 lists some of the parameters that we tuned on the validation set using a random search. Listing 4.1 shows the RNN architecture that achieved the best validation set F1-score for our entity detection task. The model takes in a batch of questions as input that contain the indices of the words that appear in the question text. The embedding layer maps the word indices to the embedding space. The LSTM outputs a hidden state for each word which is then mapped to the output space through linear layers followed by ReLU activation, dropout and batch normalization layers. The last layer has 4 outputs instead of 2 because there are two extra tokens - 'unk' and 'pad' . The 'unk' token is not used but is provided by default by the *torchtext* package that we used for pre-process the dataset. The 'pad' token is required

[2]https://nlp.stanford.edu/projects/glove/

Table 4.1: Some important parameters for the RNN that were tuned on the validation set

Parameters	**Type**	**Range**
RNN type	factor	(LSTM, GRU)
Learning rate	float	(1e-6, 1e-2)
Number of layers	int	(1, 5)
Hidden layer size	int	(100, 500)
Dropout	float	(0.1, 0.8)
Gradient clipping	float	(0.2, 0.7)
Batch size	factor	(32, 64, 128)

for mini-batch gradient descent. Our models benefit from batching since it leads to faster convergence by taking more advantage of the GPU memory.

```
EntityDetection (
  (embed): Embedding(59480, 300)
  (rnn): LSTM(300, 200, num_layers=2, dropout=0.3, bidirectional=True)
  (dropout): Dropout (p = 0.3)
  (relu): ReLU ()
  (hidden2tag): Sequential (
    (0): Linear (400 -> 400)
    (1): BatchNorm1d(400, eps=1e-05, momentum=0.1, affine=True)
    (2): ReLU ()
    (3): Dropout (p = 0.3)
    (4): Linear (400 -> 4)
  )
)
```

Listing 4.1: Entity detection model architecture

Implementation details: For training the models, we used the Adam optimizer to optimize the negative log-likelihood loss with default parameters and a learning rate of 1e-4. The model ran for 30 epochs with early termination with a patience parameter of 5 epochs. We also do gradient clipping if the norm goes over 0.6 to avoid exploding gradient problem in RNNs.

4.1.2 CRF

The Stanford Named Entity Recognizer (NER) is a tool that can label sequences of words into four classes - person, organization, location, not-entity. It extracts features such as current/previous/next word, POS tag, character n-gram, etc. and trains a conditional random field (CRF) model. A CRF is a conditional sequence model which represents the probability of a hidden state sequence given some observations. It can do sequence modeling, allowing both discriminative training and the bi-directional flow of probabilistic information across the sequence. [11]

We trained the Stanford NER on the training set and labelled the test set questions. The questions were tagged into the four classes, but we considered the three class labels (person, organization, location) as entity, so ultimately, there were two classes - entity, not-entity.

4.1.3 Results

Table 4.2 summarizes the results of the models on the entity detection task. The precision, recall and F1-score are evaluated on the token span level. For example, a true positive span means that the predicted entity token span exactly matches the ground truth from the back-projected dataset. The results show that RNN performs the best on the test set with F1-score of 91.51%. However, the CRF results are comparable and perform only a little worse. Our model effectiveness is lower than the F1-score reported by Ture and Jojic [27] but we cannot make a direct comparison between these two scores since they skip some questions in the dataset (personal communication with the author). The asterisk in the dataset column in Table 4.2 indicates that the dataset created for entity detection are different.

Table 4.2: Results for the entity detection task

Model	**Dataset**	**Precision (%)**	**Recall (%)**	**F1 score (%)**
RNN	validation	91.79	92.77	92.28
RNN	test	91.07	91.94	91.51
CRF	validation	90.65	89.89	90.27
CRF	test	90.47	89.88	90.17
RNN (Ture & Jojic)	validation (*)	-	-	94.1

4.2 Entity Linking

After entity detection, we can identify the entity words in the question and obtain a query but the main objective is to select the correct entity in the KB. This section describes how we use the query to link to the entity MID in Freebase.

Creating the indexes: In order to make the knowledge base searchable by a given query, we built three indexes: a names index, an inverted entity index, a graph reachability index. The names index maps all the entity MIDs in the Freebase subset to their names in the names file mentioned in Section 4.1. The inverted entity index maps n-grams of an entity for $n \in \{1, 2, 3\}$ to the entity MID. For example, if we have an entity MID 'm/07f3jg' with name 'sasha vujacic', the inverted index will add the MID to the postings list of three terms - 'sasha vujacic' for $n = 2$, and 'sasha' and 'vujacic' for $n = 1$. A graph reachability index maps each entity node in the Freebase subset to all nodes that are reachable by a single relation (one hop away). The indexes are created using dictionaries in Python and then dumped as pickle for faster loading.

Early termination: When we form the query, we iterate over the n-grams of the entity words in decreasing order of n. Early termination stops searching for entities for smaller values of n if we already found candidate entities. For example, if our query 'sasha vujacic' for $n = 2$ finds candidate entities, then we will terminate our search and only include those entities. Therefore, this prunes the entities that would have been otherwise retrieved for the queries 'sasha' and 'vujacic' for $n = 1$.

Ranking entities: Given a query and all the possible entities in Freebase, we split up the query into n-grams and use those n-grams to search the inverted index. Candidate entity MIDs are retrieved from the index and appended to a list, C_e and checked for the early termination condition. The entities in the candidate list, C_e are scored by a function and then ranked in terms of descending order. We explore a few ways to score candidate entities in the following sub-sections:

4.2.1 Fuzzy matching

We used the *fuzzywuzzy*[3] package to compute string matching scores between the query and a candidate entity name. The package uses Levenshtein Distance to calculate the differences between sequences. We mainly used two functions from the package: *ratio()*, which is a simple similarity score, and *partial_ratio()*, which is a similarity score based

[3]https://github.com/seatgeek/fuzzywuzzy

on substring matching to check if one string is included in the other. We used *ratio()* to compute string matching score between the candidate entity name and the query - denoted **s-query** in Table 4.3. We used *partial_ratio()* to compute substring matching score between the candidate entity name and the question - denoted **p-question** in Table 4.3 (note that we are doing partial matching on the question and not the query).

We experimented with other functions and parameters in the package as well, but we do not report results in this paper since they did not seem to improve the results.

4.2.2 Inverse document frequency (idf)

To compute the idf, we take the n-grams of the query text in decreasing order of n, i.e. $n \in \{3, 2, 1\}$. For example, our query is 'sasha vujacic' composed of two terms. Then, we get the query 'sasha vujacic' for $n = 2$, 'sasha' and 'vujacic' for $n = 1$ (given that early termination does not occur). The idf score is calculated on each term of the query and summed up to form the total idf score.

$$idf(t) = \log(\frac{N}{n_t})$$

where, N represents the total number of entities, and n_t represents the number of entities containing the term t.

The idf term for a term in the query, t, with respect to the document collection, D, is inversely proportional to the document frequency of that term. This is important since the idf score ranks rarer terms higher which improves entity ranking. For example, there may be some other entities that contain the name 'sasha' in Freebase but very few entities that contain 'vujacic' in their name so in this example, we would expect the MID 'm/07f3jg' to be ranked higher which is the right thing to do.

We experimented with tf-idf as well, where the term frequency was the number of times the candidate entity appeared in C. However, including the tf did not make any difference so we excluded them from the results.

4.2.3 Weighted combination

We took the above features (s-query, p-question, idf) and trained a logistic regression classifier to try to predict the correct entity. Then, we took the coeffiecients (w_i) and the

intercept (b) of the classifier to create a scoring function.

$$score = w_1.(squery) + w_2.(pquestion) + w_3.(idf) + b$$

This score is denoted as combined in Table 4.3.

4.2.4 Results

Table 4.3 compares the retrieval-at-k results for the similarity functions on the validation set, based on the query from entity detection using RNN. The results show that there is little difference in the retrieval scores no matter what similarity function is used. What is surprising is that even a weighted combination of the features tuned by the logistic regression classifier hardly gives us any gains. We explore the reason behind this further in the error analysis chapter. For the rest of the paper, we report results using s-query as the similarity function for entity linking.

Table 4.3: Comparison of different similarity functions for entity linking on the validation set

Similarity	**Ret@1**	**Ret@5**	**Ret@20**	**Ret@50**	**Ret@100**
s-query	64.00	78.64	84.86	88.09	90.09
p-question	63.89	79.10	85.51	88.67	90.50
idf	62.63	77.90	84.45	87.67	89.88
combined	65.16	80.09	86.05	88.95	90.74

4.3 Relation Prediction

Relation prediction is the task of predicting the relation type of the qestion. For example, for the question "where was sasha vujacic born", the relation is 'people/person/place_of_birth' referring to the place of birth. The predicate/relation is given in the dataset and there are 1,837 unique relation types in the dataset. Therefore, the objective is to do large scale classification with 1,837 possible labels and to assign a relation type to the question.

4.3.1 RNN

Similar to the previous model, we use LSTM and GRU to model dependencies among words in the question. Since this is not a tagging task, we only need to use the last hidden layer output from the RNN to make the prediction. Figure 4.2 shows the different layers of the relation prediction model and similar to the entity detection model, the beige colored boxes map the words to the GloVe embeddings and the recurrent layer goes over the entire sequence and the predictions are made based on the last hidden layer of the RNN.

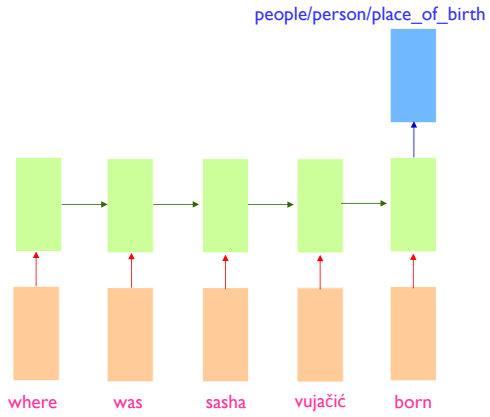

Figure 4.2: Illustrating the different layers of the relation prediction model

We experimented with different model architectures and hyper-parameters and found that 2 layer bi-directional GRU worked the best. We picked the model architecture and parameters by random search by following a similar approach to the RNN model in entity detection. The only difference is we use the RNN here to encode the question by using only the last hidden layer of the RNN here to make a classification. Listing 4.2 shows the RNN architecture that achieved the best validation set accuracy for relation prediction.

```
RelationClassification (
  (embed): Embedding(61767, 300)
  (encoder): Encoder (
    (rnn): GRU(300, 200, num_layers=2, dropout=0.3, bidirectional=True)
  )
  (dropout): Dropout (p = 0.3)
  (relu): ReLU ()
  (out): Sequential (
    (0): Linear (400 -> 400)
    (1): BatchNorm1d(400, eps=1e-05, momentum=0.1, affine=True)
    (2): ReLU ()
    (3): Dropout (p = 0.3)
    (4): Linear (400 -> 1838)
  )
)
```

Listing 4.2: Relation prediction model architecture

4.3.2 CNN

From a human standpoint, relations can be deduced from the question patter or from the use of certain words. For example, questions of the type, "where was *entity* born" is likely to be linked to the relation 'people/ person/place_of_birth'. This intuition led us to believe that CNNs may be a good choice to model the relation prediction task. CNNs cannot model sequences but they are good at extracting local features by sliding the filters over the word embeddings. We used the CNN model from Kim [15]. Our model is a little simpler since we only use a single static channel instead of the multi-channel technique described in the paper. Figure 4.3 shows a diagram of the model architecture from the paper [15]. Listing 4.3 lists the model configuration that we used. The embedding layers projects the word indices to word embeddings. The word embeddings are passed through three convolutional layers followed by ReLU activation, max-pooling and dropout, before the linear layer projects the pooled-features to the output space. Each convolutional layer has 200 filters, stride of 1, filter height of 2, 3, 4 respectively and zero-padding of 1, 2, 3 respectively.

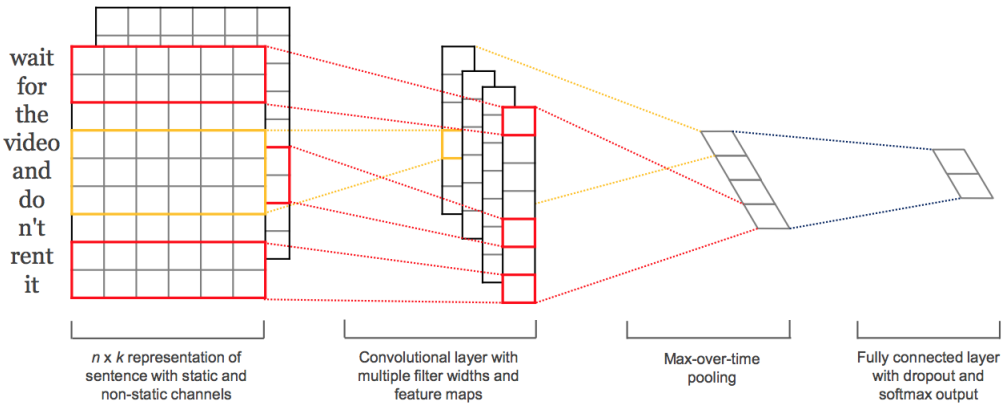

Figure 4.3: CNN model architecture from Kim [15]

```
RelationPrediction (
  (embed): Embedding(59480, 300)
  (conv1): Conv2d(1, 200, kernel_size=(2, 300),
                                    stride=(1, 1), padding=(1, 0))
  (conv2): Conv2d(1, 200, kernel_size=(3, 300),
                                    stride=(1, 1), padding=(2, 0))
  (conv3): Conv2d(1, 200, kernel_size=(4, 300),
                                    stride=(1, 1), padding=(3, 0))
  (dropout): Dropout (p = 0.5)
  (fc1): Linear (600 -> 1838)
)
```

Listing 4.3: CNN model architecture for relation prediction

4.3.3 Logistic Regression

We also used the *LogisticRegression* classifier from the *scikit-learn* package[4] with default parameters. Logistic regression is a regression model where the dependent variable is categorical. Since we have multiple classes, the model uses the one vs. rest scheme for

[4] http://scikit-learn.org/stable/

training. We run logistic regression and report results on two types of features extracted from the questions:

tf-idf on unigrams and bigrams: tf-idf means term-frequency times inverse document-frequency. We used the *CountVectorizer* and *TfidfTransformer* classes in *sciki-learn* to extract unigram and bigram features from the question.

word2vec and top 300 relation words: We split up the words in the question and averaged the word2vec embeddings to obtain a vector representation for the sentence. Out-of-vocabulary words were set as zero vector. We also split up the words in the relation class, for example, the relation 'people/person/place_of_birth' was split up into the words [people, person, place, of, birth] and made a vocabulary of the top 300 relation words. Then, we obtained a bag of words representation of the question from these top 300 relation words and concatenated the vector to the averaged word embeddings. The concatenated vector formed the features for the question.

We have experimented with feature engineering and tried out other representations such as different ranges of character and word n-gram combinations of bag-of-words and tf-idf, but they did not perform better so we have not reported their results in this paper.

4.3.4 Results

Table 4.4 reports the performance of the different models discussed in this chapter on the relation prediction task. The results show that CNNs perform slightly better than RNNs on this task. However, they both perform much better than logistic regression. This tells us that these deep learning models are significantly better than logistic regression in the task of relation prediction.

One thing to notice here is that logistic regression does better on the test set than the validation set. This is due to the fact that we used logistic regression with default parameters so there were no parameters to tune on the validation set. As a result, we used the (training + validation) set data for training when we tested on the test set. However, we only used the training data when we tested on the validation set. This is not the case for RNNs and CNNs where we used only the training data to test both on the validation and test set, since there were many parameters to tune.

Table 4.4: Results for relation prediction

Model	Dataset	Accuracy(%)	Retrieval@3(%)	Retrieval@5(%)
RNN	validation	81.77	93.59	95.73
RNN	test	81.32	93.52	95.46
CNN	validation	82.58	93.84	95.79
CNN	test	82.09	93.66	95.59
LR (tf-idf)	validation	72.36	84.67	87.58
LR (tf-idf)	test	72.64	85.39	88.16
LR (w2v+rel)	validation	70.63	85.70	89.12
LR (w2v+rel)	test	71.34	86.11	89.63
RNN (Ture & Jojic)	validation (*)	81.6	-	-

4.4 Evidence Integration

Using the top m entities and their entity linking scores, and the top n relations and their relation prediction scores, we combine the scores to output the final prediction in this module [24].

Given the all the possible combinations of the (entity, relation) pairs, we look up the graph reachability index to check if this combination of (entity, relation) exists in the KB. We prune all the combinations that do not exist in the KB. After that, we score each (entity, relation) pair using a log-linear scoring model. Given the entity linking score, s_{el}, and the relation prediction score, s_{rp}, the log-linear score is:

$$score(e, r) = s_{el}(e)^{\alpha} * s_{rp}(r)^{\beta}$$

We tuned the parameters, α and β, on the validation set and selected $\alpha = 0.7$ and $\beta = 0.1$. All the (entity, relation) pairs are then ranked in descending order of their score.

4.4.1 Entity node indegree

However, we found that there are many pairs with the same score since there can be many entities (over hundreds) in the KB with the same entity name. Therefore, we considered the entity node indegree in the KB to break these ties. For (entity, relation) pairs with the same score, we broke the ties by doing a secondary sort on the indegree count of the entity node in the KB.

Table 4.5: Results comparing the effect of secondary sorting on entity node indegree on the validation set when top 50 entities were crossed with top 5 relations

EntDet	RelPred	Indegree	Accuracy	Ret@2	Ret@3	Ret@inf
RNN	RNN	TRUE	72.69	79.32	81.37	85.91
RNN	RNN	FALSE	71.29	78.26	80.25	84.21
RNN	CNN	TRUE	72.66	79.46	81.36	85.88
RNN	CNN	FALSE	71.28	78.37	80.23	84.19
RNN	LR	TRUE	65.85	72.91	75.16	80.10
RNN	LR	FALSE	64.65	71.98	74.15	78.61
CRF	RNN	TRUE	71.30	77.85	79.86	84.06
CRF	RNN	FALSE	70.17	76.78	78.86	82.57
CRF	CNN	TRUE	71.36	77.98	79.88	84.07
CRF	CNN	FALSE	70.31	76.99	78.92	82.60
CRF	LR	TRUE	64.85	71.68	73.91	78.47
CRF	LR	FALSE	64.00	70.88	73.07	77.17

4.4.2 Results

Table 4.5 reports the results of different models with and without secondary sorting on indegree when the top 50 entities were crossed with the top 5 relations on the validation set. The results show that secondary sorting on the entity node indegree improves the retrieval scores. We follow the evaluation method by Bordes et al. [4] and mark a prediction as correct if both the entity and the relation are the same for that question in the ground truth. The main evaluation metric used is accuracy, which is the same as retrieval-at-1 (Ret@1).

We experimented with different hits on entities and relations and found that accuracy improves at higher depths. This is due to the fact that there can be many entities with the same name so sometimes the correct entity is retrieved at a greater depth. Table 4.6 reports the results for different entity detection and relation prediction models on the test set when the evidence of top 50 entities were combined with the top 5 relations. The best model combination is RNN + CNN (RNN for entity detection and CNN for relation prediction). However, CRF + CNN also produces competitive results. CRF + LR is a model with no 'non-neural-network' baseline which under-performs due to the ineffectiveness of LR compared to RNN and CNN. The results show that the deep learning

Table 4.6: Evidence integration results on the test set when different models and number of top entities and relations are considered

EntDet	RelPred	Ret@1	Ret@2	Ret@3	Ret@inf
RNN	RNN	71.4	78.48	80.4	85.25
RNN	CNN	**71.54**	78.54	80.53	85.35
RNN	LR	65.52	72.9	75.01	80.36
CRF	RNN	70.47	77.37	79.08	83.69
CRF	CNN	**70.54**	77.43	79.29	83.78
CRF	LR	**64.83**	72.01	74.07	78.92

models are important at relation prediction but much less important at entity detection. Another finding is that CNNs are slightly better or atleast perform as well as RNNs, which means that the relations are likely predicted by certain patterns in the question, and not by taking into accounts the dependencies in the sequence.

4.4.3 Final Results

Finally, we compare our results with other models in the literature. Table 4.7 compares end-to-end evaluation of the state-of-the art models to our models on the SimpleQuestions test set.

The table results show that our model outperforms complex neural network models such as memory networks from Bordes et al. and the encoder-decoder framework from Golub and He. What is surprising is that our 'non-neural-network' model of CRF + LR also outperforms the memory networks for this task. Our simple retrieval based approach based on RNN and CNN get us an accuracy of 71.54%, and seems to be performing better than the complex neural network model and rich features used by Lukovnikov et al. [18]. This model is 5.5% behind the best reported result on this task from Yu et al. As a side note, we were unable to reproduce the result reported by Ture and Jojic and we have doubts that those results are a true comparison to any of the other methods in the table.

Table 4.7: Comparison with state-of-the art models for this task on the SimpleQuestions test set

Model	**Description**	**Accuracy (%)**
Baseline	random guess	4.9
Bordes et al. [4]	memory networks	62.7
Golub and He [12]	character based encoder-decoder	70.9
Lukovnikov et al. [18]	end-to-end learning using neural embeddings	71.2
Dai et al. [9]	conditional probabilistic framework	75.7
Yin et al. [31]	attentive max-pooling	76.4
Yu et al. [32]	HR-BiLSTM	77.0
Ture and Jojic [27]	LSTM and GRU	86.8
Our approach	RNN and CNN	71.54
Our approach	CRF and CNN	70.54
Our approach	CRF and LR	64.83

Chapter 5

Error Analysis

In this chapter, we try to gain insights into the type of mistakes our approach produces in an attempt to improve our model in the future. We perform our analysis at two different stages of the pipeline: after entity linking and at the end after evidence integration.

5.1 Entity Linking

Table 4.3 in the entity linking section showed that the choice of similarity function did not have much of an effect on the retrieval results. After inspection, we realized that this was because all the similarity functions were getting the same questions right due to exact matches and missing the other ones.

To understand why this is the case, we used the correct entity names as query in the entity linking phase - we call this the 'gold' query text. In reality, it is not possible to have such query text after entity detection since the questions do not always mention the entire entity name. Using the 'gold' query text on the validation set, we counted the number of entities that had the exact name as the query. Table 5.1 shows the cumulative density sum of exact match counts.

Then, using the 'gold' query text and the 's-query' similarity function, we performed entity linking on the validation set. Table 5.2 shows the number of exact matches, partial matches and retrieval rate at different hits. A name was considered to be a partial match if the *fuzzywuzzy.ratio()* between the name and the query was above 0.6. Compared to the histogram above, the retrieval results seem to line up perfectly. At each hit, our method seems to be getting all the exact matches and plus random matches from greater depths.

At hits = 1, we get all exact matches accounting for 67.6% and around 6.8% matches due to random selection from hits=2,3,...,∞. This becomes more as number of hits increases. For example, at hits=20, we 90.6% exact matches and only 1.9% due to random selection from hits=21,22,...,∞.

Table 5.1: The cumulative density sum of exact match counts with the 'gold' query text on the validation set

Hits	1	2	3	4	5	10	15	20
Cum. Density (%)	67.66	76.23	80.16	82.49	84.11	87.61	89.43	90.63

Table 5.2: Statistics of number of exact, partial matches with retrieval rate at different hits for the 'squery' function with the 'gold' query text on the validation set

Hits	**# exact match**	**# partial match**	**Retrieval (%)**
1	10774	0	74.46
2	14258	2090	81.57
3	16818	4149	85.10
4	18955	6110	86.72
5	20841	7971	87.73
10	28508	16080	90.33
15	34748	22989	91.78
20	40130	29293	92.54

We repeated this analysis with other similarity functions and the query text from our entity detection using RNNs. The results showed similar pattern and we came to the conclusion that better features were required to disambiguate between entities with the same name as the query. We explain some of the other ideas that we have for future work in the next chapter.

5.2 Evidence Integration

We performed error analysis using RNN for entity detection and RNN for relation prediction when top-50 entities and top-5 relations were considered for evidence integration (sorting on indegree) on the validation set. Our main goal was to understand the causes

of error by our approach. We inspected the questions that were retrieved in the second position (hits = 2) but not in the top position (hits = 1). We found that there were 733 such questions. Table 5.3 shows the cause, counts and percentage of the different errors. The results tell us that entity linking is the biggest source of error.

Table 5.3: Statistics of errors of validation set questions that were retrieved in the second position but not the top

What went wrong?	**Count**	**Percentage**
Both MID and relation wrong	61	8.3%
Correct MID, wrong relation	324	44.2%
Correct relation, wrong MID	348	47.5%

We also inspected the questions that were retrieved in the third position (hits = 3) but not in the first or second position and there were 205 such questions. Table 5.4 shows the statistics for the different errors and the results show that entity linking becomes an even bigger source of error at greater depths accounting for more than 75% of these mistakes.

Table 5.4: Statistics of errors of validation set questions that were retrieved in the third position but not in the first or second position

What went wrong?	**Count**	**Percentage**
Both MID and relation wrong	22	10.7%
Correct MID, wrong relation	29	14.2%
Correct relation, wrong MID	154	75.1%

We manually inspected some of the questions that were retrieved in the second position but not the first. Table 5.5 shows a few samples that were retrieved in the second position with the cause and the reason for the errors in the top position.

For questions where we got the correct relation and the wrong MID, we noticed that it was due to two main reasons: *incorrect linking* or *incorrect query*. Incorrect query is due to the errors propagated from the entity detection phase. For incorrect linking, we noticed that in almost all cases, both the MIDs had the same name and our approach was unable to disambiguate the correct entity from the incorrect one. For example, there are over 20 entities in KB with the name 'charlie chaplin' so the question 'which films did charlie chaplin direct?' becomes really difficult to answer. From the question, we can understand the question is likely talking about someone in the film industry but there still could be

multiple entities who meet that criteria. This raises a philosophical question of whether we are actually wrong in this case just because the ground truth does not contain that answer. For questions where we got the correct MID but not the relation, we found many of them were quite hard to distinguish even by humans. For example, the question 'which release was reading on' is referring to a relation *'music/release_track/recording'* but our classifier predicted the relation *'music/release_track/release'*. For questions where we got both the MID and the relation wrong, we noticed that it was due a combination of the reasons noted above or due to incorrect evidence combination where we score an incorrect pair higher than the correct pair. We also noticed that incorrect evidence combination usually occurs due to incorrect relation prediction where the incorrect relation is scored much higher than the correct relation.

Table 5.5: Sample questions that were retrieved in the second position with the cause and reason of errors in the top position

Cause	**Reason**	**Question**
Both MID and relation	Same entity name but different MID. Relation predicted 'music/release/track' instead of 'music/artist/track'.	what is a track name from the the corrs
Both MID and relation	Incorrect evidence combination. Correct linking. Relation predicted 'location/location/ containedby' instead of 'location/ location/contains'.	which prairie style house in chicago is located on sheridan road
Only MID	Incorrect query - 'cbs'. Top entity retrieved referred to 'cbs films inc' but correct entity referred to 'columbia broadcasting system'.	what is a film produced by cbs
Only MID	Incorrect linking - same entity name.	what group of people were involved in the siege of paris
Only MID	Incorrect linking - same entity name.	which films did charlie chaplin direct?
Only relation	Indistinguishability. Relation predicted 'influence/influence_ node/influenced_by' instead of 'influence/influence_node/influenced'	who was influenced by lewis black

Chapter 6

Conclusion

6.1 Future Work

The error analysis performed after entity linking and evidence integration points to entity disambiguation in entity linking as the biggest source of error. Only considering lexical match on the surface form does not give our method enough power to distinguish between two entities with the same name. Hence, one direction for future work is to include richer features from Freebase that give us more information about the entities:

- We can use a neural model, such as StarSpace[1], to learn entity embeddings in Freebase. In StarSpace [28], the embeddings of all entities and relation types are learnt in two ways: one for predicting tail entity given head entity, one for predicting head entity given tail entity. Afterwards, we can use another train another model to pick the entity embedding that links best to the question.
- We can include more information from Freebase fields such as the predicate *'common.topic.notable_types'* can be used to extract the type of entities, for example, scientist, businessman, actor, etc. Dai et al. [9] used a bag-of-words representation for the type information to encode the entities and Lukovnikov et al. [18] used the type information and name and encoded the entity embeddings with a GRU.

Relation prediction accounted for more than 40% of the errors for questions retrieved in the second position. To improve relation classification, we can perform a multi-stage

[1]https://github.com/facebookresearch/StarSpace

retrieval, where initially we can use our current method to retrieve the top 10 candidates. In the next stage, we can adopt a Siamese network style architecture similar to Yin et al. [31] where both the question and the relation is encoded using a neural network and their features are combined to make a prediction. The relation name can be encoded using the entire relation name and also by splitting the relation into tokens to give more expressive power.

Another direction of improvement is in entity detection by reducing the amount of errors that occur due to incorrect queries. We could use NeuroNER [10], the current state-of-the-art model for named- entity recognition, to do entity detection and that should improve our queries and therefore, entity linking.

6.2 Summary

In this thesis, we attempted to tackle the factoid QA task on the SimpleQuestions dataset, where the questions can be answered by retrieving a single fact from Freebase. We used a retrieval-based approach and converted the problem into subtasks namely: entity detection, entity linking, relation prediction and evidence integration. We experimented with different models for the entity detection and relation prediction task with the aim of trying to condense down to simplest possible models that perform really well on this task. The experimental results show that neural networks help a lot in the relation prediction task but not so much in entity linking. We also experimented with different scoring functions to rank entities during linking and with a log-linear model to combine evidence scores from entity linking and relation prediction. Our main contribution from this work is establishing two strong baselines for the factoid QA task: one using a simple neural network architecture and one without using any neural networks. We believe these two models will be helpful to the community to compare the performance of future models. We also performed error analysis which revealed the difficulty of the task and also a weakness of our approach. We were linking entities mostly based on exact name matches and we need a richer set of features to disambiguate entities - we keep this as a direction for future work.

References

[1] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, volume 2, page 6, 2013.

[2] Jonathan Berant and Percy Liang. Semantic parsing via paraphrasing. In *ACL (1)*, pages 1415–1425, 2014.

[3] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1247–1250, New York, NY, USA, 2008. ACM.

[4] Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. Large-scale simple question answering with memory networks. *CoRR*, abs/1506.02075, 2015.

[5] Antoine Bordes and Jason Weston. Reasoning, attention, memory (ram) nips workshop 2015. *Reading*, 2017, 2017.

[6] Denny Britz. Understanding convolutional neural networks for nlp, 2015.

[7] Denny Britz. Recurrent neural networks tutorial, 2016.

[8] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[9] Zihang Dai, Lei Li, and Wei Xu. Cfo: Conditional focused neural question answering with large-scale knowledge bases. *arXiv preprint arXiv:1606.01994*, 2016.

[10] Franck Dernoncourt, Ji Young Lee, and Peter Szolovits. Neuroner: an easy-to-use program for named-entity recognition based on neural networks. *arXiv preprint arXiv:1705.05487*, 2017.

[11] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

[12] David Golub and Xiaodong He. Character-level question answering with attention. *CoRR*, abs/1604.00727, 2016.

[13] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1994.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[15] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[16] Donald Knuth. *The TeXbook*. Addison-Wesley, Reading, Massachusetts, 1986.

[17] Leslie Lamport. *LaTeX — A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[18] Denis Lukovnikov, Asja Fischer, Jens Lehmann, and Sören Auer. Neural network-based question answering over knowledge graphs on word and character level. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1211–1220. International World Wide Web Conferences Steering Committee, 2017.

[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[20] Chris Olah and Shan Carter. Attention and augmented recurrent neural networks. *Distill*, 1(9):e1, 2016.

[21] Christopher Olah. Deep learning, nlp, and representations. *GitHub blog, posted on July*, 7, 2014.

[22] Christopher Olah. Understanding lstm networks. *GitHub blog, posted on August*, 27, 2015.

[23] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[24] Jeffrey Pound, Alexander K Hudek, Ihab F Ilyas, and Grant Weddell. Interpreting keyword queries over web knowledge bases. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 305–314. ACM, 2012.

[25] Siva Reddy, Mirella Lapata, and Mark Steedman. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics*, 2:377–392, 2014.

[26] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448, 2015.

[27] Ferhan Türe and Oliver Jojic. Simple and effective question answering with recurrent neural networks. *CoRR*, abs/1606.05029, 2016.

[28] Ledell Wu, Adam Fisch, Sumit Chopra, Keith Adams, Antoine Bordes, and Jason Weston. Starspace: Embed all the things! *arXiv preprint arXiv:1709.03856*, 2017.

[29] Xuchen Yao. Lean question answering over freebase from scratch. 2015.

[30] Scott Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. 2015.

[31] Wenpeng Yin, Mo Yu, Bing Xiang, Bowen Zhou, and Hinrich Schütze. Simple question answering by attentive convolutional neural network. *arXiv preprint arXiv:1606.03391*, 2016.

[32] Mo Yu, Wenpeng Yin, Kazi Saidul Hasan, Cicero dos Santos, Bing Xiang, and Bowen Zhou. Improved neural relation detection for knowledge base question answering. *arXiv preprint arXiv:1704.06194*, 2017.