ID4100: Creative Engineering Project MultiHop Question Answering Final Report

Sahana Ramnath & Shreyas Chaudhari
Department of Electrical Engineering
Indian Institute of Technology Madras
EE15B109 & EE15B019

1 Introduction

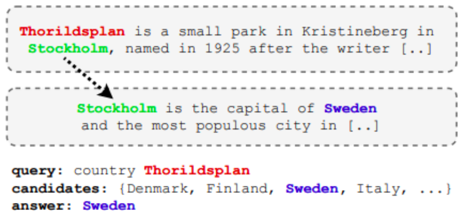
The aim of this project is to ideate about and implement a deep learning model to tackle a largely active topic in natural language processing - the task of question answering, more specifically, **multi-hop** question answering. Models have been implemented and tested on two recently released datasets to get a quantitative measure of the performance of the model: the WIKIHOP dataset and the HOTPOTQA dataset.
These datasets require and enforce multi-step reasoning and information combination from different documents to infer the correct answer. A hierarchical LSTM based model with query based attention at every level and query refinement is employed for this task.

The task of question-answering has been the a central problem in natural language processing for the last few years. The datasets used to benchmark models on the task have evolved over time. The Stanford Question Answering Dataset (SquAD) (Pranav Rajpurkar (2016)) has one-shot question answering where the answers can be found in one chunk of the support. The complexity of benchmarking has increased over time with multi-hop reasoning gaining special attention over the last two years. The two significant datasets on that front are WikiHop (Welbl et al. (2018)) and HotPotQA (Zhilin Yang (2018)) - both of which we have benchmarked on. The analysis that follows is for the models we used and how they perform on the two datasets; and how the datasets compare against one another.

2 Datasets

Two datasets were were used to evaluate the performance of the models. Both datasets require multi-hop reasoning for the prediction of a final answer. However, the dataset's hierarchical structure as well as required final prediction is different for both of them.

- WikiHop : WikiHop (Welbl et al. (2018)) requires the model to choose one amongst several candidates provided as options(where each candidate is a word/continuous span in the corresponding context given). For each question, the context is given as a set of paragraphs which contribute to two levels of hierarchy : within and across paragraphs.


Thorildsplan is a small park in Kristineberg in Stockholm, named in 1925 after the writer [..]

Stockholm is the capital of Sweden and the most populous city in [..]

query: country Thorildsplan
candidates: {Denmark, Finland, Sweden, Italy, ...}
answer: Sweden

Figure 1: WikiHop example data point (Nicola De Cao (2019))

- HotPotQA : HotPotQA requires the model to predict either a 'yes' or a 'no' or a continuous span of words from the corresponding context given. For each question, the context is given as a set of (title,paragraph) pairs where each paragraph is further already divided into a set of sentences. This contributes to three levels of hierarchy : words within a sentence, sentences within a paragraph and across paragraphs. In addition to predicting the answer, the model also has to be predict the set of **supporting facts** used to arrive at the answer; here the supporting facts are whole sentences from the context.

Paragraph A, Return to Olympus:
[1] *Return to Olympus is the only album by the alternative rock band Malfunkshun.* [2] *It was released after the band had broken up and after lead singer Andrew Wood (later of Mother Love Bone) had died of a drug overdose in 1990.* [3] Stone Gossard, of Pearl Jam, had compiled the songs and released the album on his label, Loosegroove Records.

Paragraph B, Mother Love Bone:
[4] *Mother Love Bone was an American rock band that formed in Seattle, Washington in 1987.* [5] The band was active from 1987 to 1990. [6] *Frontman Andrew Wood's personality and compositions helped to catapult the group to the top of the burgeoning late 1980s/early 1990s Seattle music scene.* [7] *Wood died only days before the scheduled release of the band's debut album, "Apple", thus ending the group's hopes of success.* [8] The album was finally released a few months later.

Q: What was the former band of the member of Mother Love Bone who died just before the release of "Apple"?
A: Malfunkshun
Supporting facts: 1, 2, 4, 6, 7

Figure 2: HotPotQA example data point (Zhilin Yang (2018))

Owing to the above differences, the models used for each dataset differs in details and output layer(s) although the underlying structure and flow is the same. Hence, we will analyse the model and performance for the two datasets separately.

3 WikiHop

3.1 Dataset Details

The WikiHop Dataset has 43,738 train datapoints, 5129 validation datapoints and 2451 hidden test datapoints.

Each datapoint consists of :

- a query of the form relation+entity
- a unique datapoint id
- a set of support documents/paragraphs, where each paragraph corresponds to a different topic
- a list of candidates out of which one is the answer
- annotations from three reviewers on whether the answer follows from the given data and whether single or multiple documents are required to infer it (train and validation only)
- the answer from the list of candidates (train and validation only)

Each candidate(and so the answer as well) is constrained to be a contiguous set of words from any of the supporting paragraphs. The relation part of the query is from a set of fixed query types given in the training set. Each query has multiple supporting paragraphs, some/all of which are relevant to answering this query. In the train/validation set, the maximum number of candidates a query has is 79, the maximum number of supporting paragraphs is 64 and the maximum number of words in a paragraph is 300.

3.2 Preprocessing

For usage of the datapoints by the tensorflow model, we had to make each datapoint to be uniformly padded/cut to the required size.

- Query : We use the relation and entity parts separately as described below.
  - Query Relation : This is just one term which consists of multiple words separated by underscores. However, since this is comes from a fixed set of query relation codes, we keep them as such and have a separate learnable embedding for them. For character embedding, we restricted/padded it to 50 characters including underscores
  - Query Entity : We optimally restrict/pad the query entity to 5 words, with 10 characters each.
- Candidates : We constrain each query to have 70 candidates, each of which has 10 words (with 10 characters each).
- Supporting paragraphs : We first filter the supporting paragraphs based on whether they have words from any of the candidates or the query entity. Then we restrict/pad each query to have a certain number of paragraphs each with a certain number of words. For most of our experiments, we constrained each query to have 15 paragraphs with 150 words each. In some of our later experiments, we increased it to 20 documents with 200 words each. Our final experiment model(our highest accuracy) used 15 documents with 300 words each. Each word has 10 characters.

We decided the above constraints based on the mean/median of their distribution in the train set, as well as observed performance in the model(in hyperparameter tuning). We created a separate word and character vocabulary for the support paragraphs, query relation and query entity. The candidates use the same vocabulary as the supports.

3.3 Model Overview

We started off with a simple hierarchical LSTM model. We then successively added normal query based attention, co-attention, self-attention, character embeddings, candidate embeddings and query refinement. Each of these additions gave us significant boosts towards our final validation accuracy of **63.9%**. We also ran a set of experiments on LSTM+GCN models(which are described in detail in Section 3.5.

3.4 Hierarchical LSTM Model

Since each supporting paragraph mainly describes a different topic, in our model they are not concatenated as in most of the previous models, but treated separately, i.e., **hierarchically**. Sections 3.4.1 to 3.4.11 describe each of the individual components of the model in detail. Section 3.4.12 gives the final model flow combining all these components.

3.4.1 Word and Character Embeddings

All the word embeddings(for query entity, supports and candidates) were used from the pretrained GloVe vectors (Pennington et al. (2014)) available publicly. For query relation, the "word" embedding was kept trainable since it is like a unique code. Word embeddings using character data was obtained by implementing a character-level CNN as described in Zhang et al. (2015). While GloVe vectors give appropriate embeddings for most of the commonly used words,character embeddings are useful for named entities which either don't exist or have very similar embeddings in GloVe. Let the word embedding size from GloVe/learnable matrix be d_1 and from charCNN be d_2.

$$W_{emb} = [W_{glove}, W_{char}] \in \mathrm{R}^{d_1+d_2}$$

3.4.2 Query

The query relation's embedding Q_r is a concatenation of the corresponding learnable vector and character embedding. The query entity word glove+character embeddings are passed through a bidirectional LSTM whose final state is used as the query entity Q_e. Q_r and Q_e are concatenated and passed through a fully connected layer to give the final query representation Q. Let the number of units in the LSTM be d.

$$Q_r = [Q_{learnable}, Q_{char}] \in \mathrm{R}^{d_1+d_2}$$
$$Q_{e1} = [Q_{e,glove}, Q_{e,char}] \in \mathrm{R}^{d_1+d_2}$$
$$Q_e = \text{BiLSTM}(Q_{e1}) \in \mathrm{R}^{2d}$$
$$Q = \text{FC}([Q_r, Q_e]) \in \mathrm{R}^{2d}$$

3.4.3 Word Level Encoder

Each support paragraph's words(glove+char+candidate positional embeddings) are passed through the same word-level bidirectional LSTM.Since the candidate word sequences come directly from the support, we also add a 3-dimensional learnable embedding to each word in the support which depends on whether it is a candidate word or not. The outputs W_1 here are used as the contextual word representations. The final states for each paragraph P_1 are used as initial paragraph embeddings. Let the number of units in the LSTM be d. Let the number of words in each paragraph be n_w, the number of paragraphs be n_p and the total number of words $n_w * n_p$ be n_{tot}.

$$W_{emb} = [W_{glove}, W_{char}, W_{iscand}] \in \mathrm{R}^{d_1+d_2+3}$$
$$W, P = \text{BiLSTM}(W_{emb}) \in \mathrm{R}^{n_w \times 2d}, \mathrm{R}^{2d}\text{(per paragraph)}$$
$$W_1 \in \mathrm{R}^{n_tot \times 2d}, P_1 \in \mathrm{R}^{n_p \times 2d}$$

3.4.4 Passage Level Encoder

In our initial experiments, the sequence of paragraph embeddings P_1 were passed through an LSTM to get P_2, the final passage embeddings. Later, a self-attention module(described below) was used to get P_2 from P_1 since the paragraphs have no natural sequence order to them.

$$P_{final} = \text{BiLSTM}(P_1) \in \mathrm{R}^{2d}$$
$$\text{(or)}$$
$$P_{final} = \text{Self-Attention}(P_1) \in \mathrm{R}^{2d}$$

3.4.5 Self-Attention

The initial embeddings of the words and paragraphs W_2 (obtained after co-attention as explained in Section 3.4.7) and P_1 contain contextual information about each word/paragraph obtained on reading the paragraph(s) *just once*. However, to understand the paragraph(s) better, it is necessary to re-read it keeping in memory the information already learnt, in order to clear any misconceptions there were previously; while this can be done using a follow-up LSTM, using another sequential layer may not be hugely reflective of how we as humans re-read paragraphs. Hence we use a gated self-attention module inspired from Zhao et al. (2018) within and across paragraphs.
Here, we give a common set of self-attention equations which are used at both levels of hierarchy and accordingly, D refers to W_2 or P_1, n refers to n_w or n_p and d refers to the embedding size.

$$\text{Input} : D \in \mathrm{R}^{n \times 2d}$$
$$D_1 = FC(D) \in \mathrm{R}^{n \times 2d} \text{ (input projected)}$$
$$A_1 = D_1 D_1^T \in \mathrm{R}^{n \times n} \text{ (attention weights)}$$
$$D_2 = A_1 D_1 \in \mathrm{R}^{n \times 2d} \text{ (attention weighed embedding)}$$
$$D_3 = FC([D_2, D]) \in \mathrm{R}^{n \times 2d} \text{ (linear projection)}$$
$$A_2 = FC(D_3) \in \mathrm{R}^{n \times 2d} \text{ (attention gate)}$$
$$D_{final} = A_2 * D_3 + (1 - A_2) * D \text{ (self-attended final embedding)}$$

W_{final} and P_{final} are obtained as above. Note that word-level self attention is done within each paragraph individually, maintaining the hierarchy.

3.4.6 Query Refinement

Here, we used query reduction module inspired from Query Reduction Networks (Seo et al. (2016b)) in order to get a better representation of the query based on the supporting paragraphs. On observing

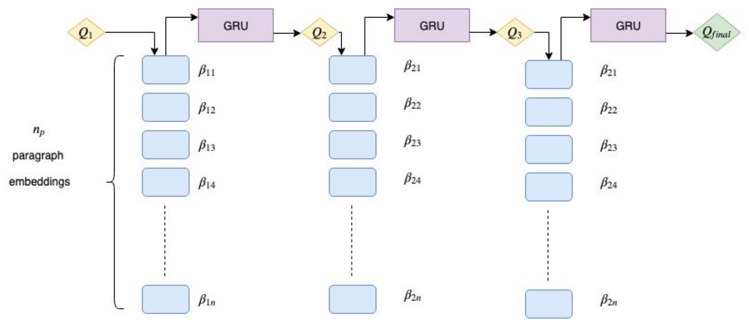


Figure 3: Query Reduction Module

the dataset, most of the questions seemed to have between 2-3 hops; hence we used 3 refinemnets of the query as shown in Figure 3. At each refinement, we have obtained attention weights of the paragraphs with respect to the query using Bahdanau attention (Bahdanau et al. (2014)).
Refinement 1 :

$$Q_1 = Q$$
$$\beta_1 = \text{Bahdanau}(Q_1, P_{final}) \text{ and tiled to embedding size } 2d$$
$$P_{weighed} = \sum(\beta_1 * P_{final}) \in \mathrm{R}^{n_p}$$
$$Q_2, S_1 = \text{GRUCell}(P_{weighed}, Q_1)$$

Refinement 2 :

$$\beta_2 = \text{Bahdanau}(Q_2, P_{final}) \text{ and tiled to embedding size } 2d$$
$$P_{weighed} = \sum(\beta_2 * P_{final}) \in \mathrm{R}^{n_p}$$
$$Q_3, S_2 = \text{GRUCell}(P_{weighed}, S_1)$$

Refinement 3 :

$$\beta_3 = \text{Bahdanau}(Q_3, P_{final}) \text{ and tiled to embedding size } 2d$$
$$P_{weighed} = \sum(\beta_3 * P_{final}) \in \mathrm{R}^{n_p}$$
$$Q_{final}, S_3 = \text{GRUCell}(P_{weighed}, S_2)$$

$Qfinal$ is the final representation of the query. Q_{final} and β_3 are used in Section 3.4.8.
Note : GRUCell was used here rather than LSTMCell for satisfying dimensionality.

3.4.7 Co-Attention

A co-attention module was used between the query and the supports to determine which words in the paragraph were most relevant to answer the query. The word embeddings W_1 were then changed and projected accordingly. The equations we used for this module was inspired from BiDAF, Seo et al. (2016a).

$$Q_1 = FC(Q_{final}) \in \mathrm{R}^{2d}$$
$$\text{Tile } Q_1 \text{ to total num words in support } Q_{tiled} \in \mathrm{R}^{n_{tot} \times 2d}$$
$$W_{temp} = [W_1, Q_{tiled}, W_1 * Q_{tiled}] \in \mathrm{R}^{n_{tot} \times 6d}$$
$$W_2 = FC(W_{temp}) \in \mathrm{R}^{n_{tot} \times 2d}$$

3.4.8 Query based Attention weights

The query representation Q was used to attend to the final word and paragraph representation to give attention weights α and β respectively. Bahdanau Attention (Bahdanau et al. (2014)) was used for this.

$$\alpha = \text{Bahdanau}(Q_{final}, W_{final})$$
$$\beta = \beta_3$$

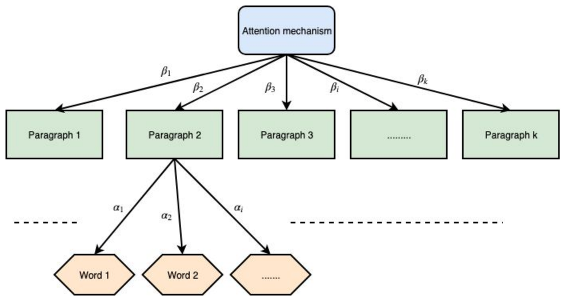


Figure 4: Bahdanau Attention with Query for Wikihop

These two sets of weights α and β are tiled and used to weigh the word embeddings to give $W_{weighed}$ to be used in the Candidate Processing layer (Section 3.4.9)

$$W_{weighed} = \alpha * \beta * W_{final}$$

3.4.9 Candidate Processing

In this QA task, the candidates are constrained to be directly from the paragraphs, i.e., surrounding context is important. We also found that, for many of the questions, previous knowledge about the entity was necessary to answer the query. Hence, to incorporate both surrounding context as well as previous knowledge, we get embeddings of each candidate from two source : directly taking the corresponding word embeddings from $W_{weighed}$, taking word+char embeddings of the candidate words and passing through an LSTM.
Here, we gave candidates a limit of 5 words with 10 characters each. For multiple word candidates and multiple occurence candidates, we take the average embedding C_2 of all corresponding words from $W_{weighed}$ and the final state C_1 of the LSTM. We allowed up to 70 candidates per query.

$$C_{emb} = [C_{glove}, C_{char}] \in \mathrm{R}^{70\times5\times(d_1+d_2)}$$
$$C_1 = \text{BiLSTM}(C_{emb}) \in \mathrm{R}^{70\times2d}\text{(taking final state)}$$
$$C_2 \in \mathrm{R}^{70\times2d} \text{ (as explained above)}$$
$$C = [C_1, C_2] \in \mathrm{R}^{70\times2d}$$

3.4.10 Logit Calculation

To get a probability distribution over the candidates so as to predict the final answer, we project C using two FC layers.

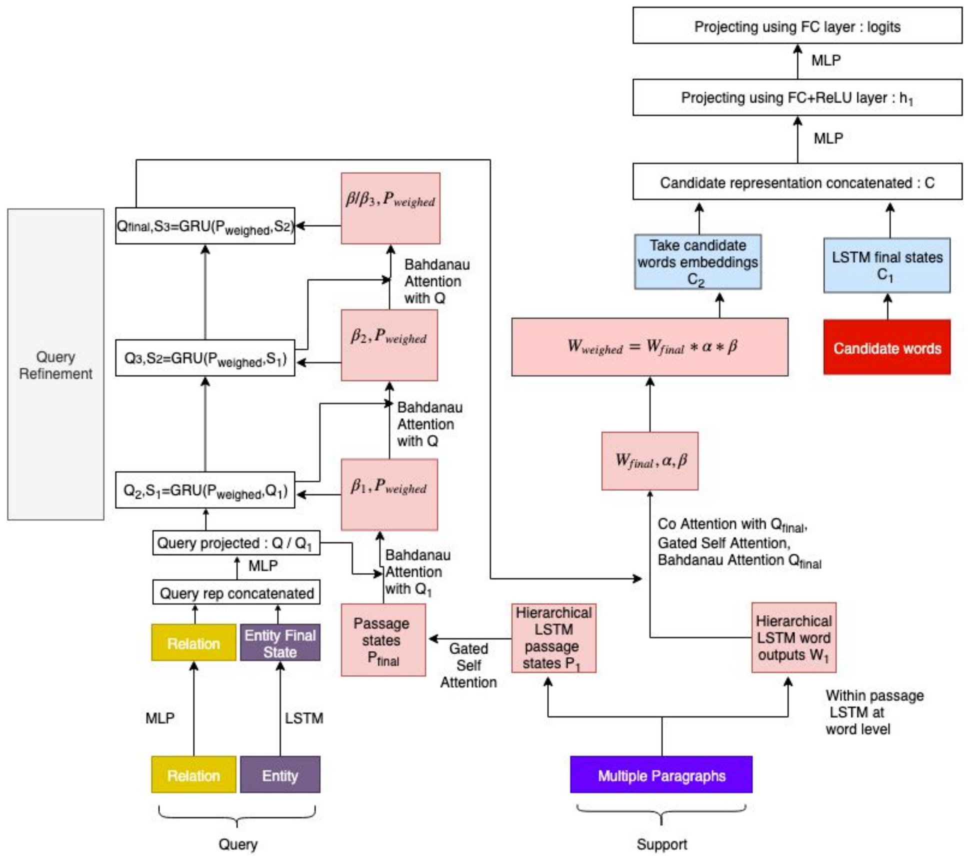
$$h_1 = FC(C) \in \mathrm{R}^{70\times d_{fc}}$$
$$\text{logits} = FC(h_1) \in \mathrm{R}^{70}$$

3.4.11 Prediction and Loss

The logits obtained above are further used to predict the answer using a softmax layer. They are also used to calculate the training CROSS-ENTROPY loss between the predicted and true answer labels to be minimized.

3.4.12 Final Model Flow

The final model flow is as in Figure 5.



Projecting using FC layer : logits
MLP
Projecting using FC+ReLU layer : h_1
MLP
Candidate representation concatenated : C
Take candidate words embeddings C_2
LSTM final states C_1
$W_{weighed} = W_{final} * \alpha * \beta$
Candidate words
W_{final}, α, β
Co Attention with Q_{final}, Gated Self Attention, Bahdanau Attention Q_{final}
Query Refinement
$Q_{final}, S_3 = GRU(P_{weighed}, S_2)$
$\beta/\beta_3, P_{weighed}$
Bahdanau Attention with Q
$Q_3, S_2 = GRU(P_{weighed}, S_1)$
$\beta_2, P_{weighed}$
Bahdanau Attention with Q
$Q_2, S_1 = GRU(P_{weighed}, Q_1)$
$\beta_1, P_{weighed}$
Query projected : Q / Q_1
MLP
Bahdanau Attention with Q_1
Query rep concatenated
Relation
Entity Final State
Passage states P_{final}
Gated Self Attention
Hierarchical LSTM passage states P_1
Hierarchical LSTM word outputs W_1
MLP
LSTM
Within passage LSTM at word level
Relation
Entity
Multiple Paragraphs
Query
Support

Figure 5: WikiHop Model Flow

We obtained a dev set accuracy of **63.9%** with this model. Our full experimental results and hyper-parameter configurations can be found in Section 5.

3.5 LSTM+GCN Model

The model aims to mimic a natural course of human reasoning and employs LSTMs and GCNs (Kipf & Welling (2016)), each for specific tasks. The bidirectional-LSTM is used to tune each word vector/embedding based on its surrounding words - thus given each embedding a *local context*. The GCN provides a more *global context* which is determined by the graphs it operates on. Two level of graphs were considered for this model:

- One that connects tokens within the paragraph: the intra-document graph. Since the answer candidate space of this dataset has nouns/entities/dates as elements, the noun tokens (common nouns, proper nouns and dates) within each paragraph are all connected to each other.

- One that connects tokens across documents: the inter-document graph. The co-occurent tokens across documents are connected, to share information about the same entity across documents. This serves are the crux of ***multi-hop reasoning***

The Figure 6 shows how the graph connections would act on an example.

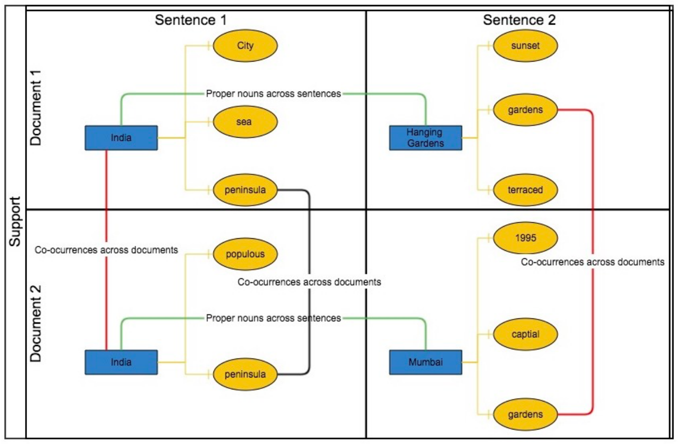


Figure 6: Example graph connections

In the simplest case, the transformation applied by a GCN can be written as:

$$f(H^{(l)}, A) = \sigma\left(AH^{(l)}W^{(l)}\right)$$

where A is the adjacency matrix that defines the graph structure, H is the input (usually embeddings) and W is the weight matrix. But for the degree of nodes to be normalised, D the diagonal node degree matrix is factored in into the form:

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

The final logit calculation and prediction is the same as the one decribed above in Sections 3.4.11 and 3.4.10.

3.5.1 Adjacency Matrix

This being a graph based model, the way the graph is defined largely governs the performance of the model. As previously described, there are two types of connections: inter-document and intra-document.
The most common way of representing a graph is the the form of an adjacency matrix. As these need to be passed as tensors to the training loop, a block diagonal structure was used as shown below:

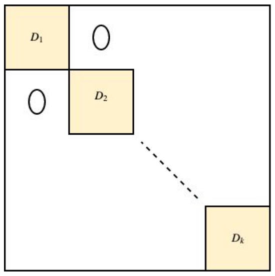


Figure 7: Adjacency Matrix structure

where D_is are the adjacency matrices for the intra document connections and the entire matrix shows inter document connections in a **block diagonal** form.

4 HotPotQA

4.1 Dataset Details

The HotPotQA dataset has 90,564 train datapoints, 7405 validation datapoints and 7405 hidden test datapoints.

Each datapoint consists of :

- a unique datapoint id
- a set of context paragraphs, where each paragraph corresponds to a different topic. Each paragraph consists of a title and a set of sequential sentences, each of which is a sequence of words.
- a natural language query which could either ask for an answer span from the context or a yes/no answer
- answer label, which says if it is a span answer or yes or a no
- the answer, which is a span from any of the sentences in the context or 'yes' or 'no'
- supporting facts which describe which sentences from which paragraphs lead to predicting the answer span starting from the query. This has to be predicted as well during test time
- Type of question : Whether entity type(answer span to be predicted) or bridge type(yes/no answer) [available only in train, not required to be predicted for test leaderboard]
- Difficulty level of question : easy/medium/hard [only in train set]

The maximum number of paragraphs a query can have is 10 and maximum number of sentences per paragraph is 10 with a maximum of 393 words each. Maximum number of words in a title is 16, in query is 108.

4.2 Preprocessing

For usage of the datapoints by the tensorflow model, we had to make each datapoint to be uniformly padded/cut to the required size.

- Query : We padded the query till a maximum of 40 words with 10 characters each.
- Context paragraphs : We restrict/pad each query to have a certain number of paragraphs each with a certain number of sentences with a certain number of words. For our preliminary experiments, we constrained each query to have 10 paragraphs with 10 sentences with 40 words each. Each word has 10 characters.
- We padded each title to the maximum of 16 words with 10 characters each.
- If the answer span has multiple occurences in the context, we unifomly choose the first occurence as the correct span. However, since this is potentially incorrect supervision, we will be changing this in future experiments.
- We take all the supporting fact labels available and do multilabel supervision for them.

We decided the above constraints on a preliminary analysis of the mean/median of their distribution in the train set (refer Tables 1,2 and 3) as well as *memory constraints*. However, we intend to increase these limits in our future experiments, since lack of information is a valid reason the model can't predict enough answers correctly. We created a separate word and character vocabulary for the context paragraphs, query and title. **For some of the queries, the supporting facts were not present in the context. Hence we removed these datapoints from the train set.** After removal of wrong datapoints, we had 87,555 train datapoints remaining which we used in our experiments.

Bins	Counts
3 - 13.5	33115
13.5 - 24.0	42031
24.0 - 34.5	10062
34.5 - 45.0	2736
45.0 - 55.5	1548
55.5 - 66.0	609
66.0 - 76.5	248
76.5 - 87.0	68
87.0 - 97.5	19
97.5 - 108.0	11

Table 1: Query

Bins	Counts
1 - 2.5	402719
2.5 - 4.0	217469
4.0 - 5.5	208926
5.5 - 7.0	41547
7.0 - 8.5	26434
8.5 - 10.0	2026
10.0 - 11.5	501
11.5 - 13.0	25
13.0 - 14.5	9
14.5 - 16.0	11

Table 2: Title

Bins	Counts
0 - 39.3	3473119
39.3 - 78.6	223912
78.6 - 117.9	5411
117.9 - 157.2	620
157.2 - 196.5	228
196.5 - 235.8	39
235.8 - 275.1	0
275.1 - 314.4	0
314.4 - 353.7	3
353.7 - 393.0	12

Table 3: Context

4.3 Model

We initially tested our WikiHop model directly on HotPotQA with an additional level of hierarchy added. While for Wikihop we had built the model from scratch and could see the benefit/problems of adding each component, we couldn't comment on the effect of those modules for HotPotQA since we directly started off with a complicated architecture. Hence we have **restarted** our experiments on HotPotQA, and we have started building the model from scratch.

A hierarchical LSTM model with LSTM encoding at word and sentence level(since there's a predefined sequence there) and self-attention and title LSTM-encoding at paragraph level is our base model now. We have also used Bahdanau Attention with respect to the query to get α, β and γ attention weights at each of the above levels(refer Figure 8). We have used these weights to weigh the word embeddings obtained after the LSTM to predict the start and end of the span and to weigh the sentence embeddings to predict the sentence labels.

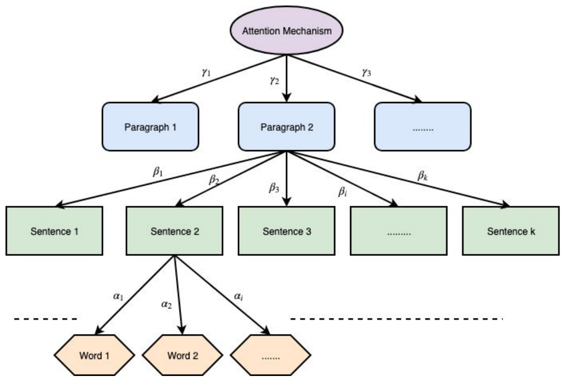


Figure 8: Bahdanau Attention with Query for HotPotQA

4.4 Model Flow

All our notations and symbols are similar to the ones used in Section 3.4. We use W to refer to word, S to refer to sentence, P to refer to paragraphs, T to refer to title and Q to refer to query embeddings. n_w, n_s, n_p refer to total number of words,sentences and paragraphs respectively.

$$\begin{gathered} Q = \text{BiLSTM}(Q_{emb}) \in \mathrm{R}^{2d} \\ W_1, S_1 = \text{BiLSTM}(W_{emb}) \in \mathrm{R}^{n_w \times 2d}, \mathrm{R}^{n_s \times 2d} \\ S_2, P_1 = \text{BiLSTM}(S_1) \in \mathrm{R}^{n_s \times 2d}, \mathrm{R}^{n_p \times 2d} \\ P_2 = [\text{BiLSTM}(T), \text{self-attention}(P_1)] \in \mathrm{R}^{n_p \times 4d} \end{gathered}$$

We used Bahdanau attention with query to get attention weights at each level.

$$\begin{gathered} \alpha = \text{Bahdanau}(Q, W_1) \\ \beta = \text{Bahdanau}(Q, S_2) \\ \gamma = \text{Bahdanau}(Q, P_2) \\ W_{weighed} = \alpha * \beta * \gamma * W_1 \end{gathered}$$

For span prediction, we used another layer of LSTMs inspired by BiDAF Seo et al. (2016a). For supporting fact prediction and answer label prediction, we are using a simple feed forward network as of now.

<u>Span Prediction</u>

$$\begin{gathered} W_i = \text{BiLSTM}(W_{weighed}) \in \mathrm{R}^{n_w \times 2d} \\ \text{span-start} = FC([W_{weighed}, W_i]) \in \mathrm{R}^{n_w} \end{gathered}$$

$$\begin{gathered} W_j = \text{BiLSTM}(W_i) \in \mathrm{R}^{n_w \times 2d} \\ \text{span-end} = FC([W_j, W_i]) \in \mathrm{R}^{n_w} \end{gathered}$$

<u>Answer Label Prediction</u>

$$\begin{gathered} P_{weighed} = \textstyle\sum_{\text{last dimension}} \gamma * P_2 \in \mathrm{R}^{n_p} \\ \text{ans-label-logits} = FFNN(P_{weighed}) \in \mathrm{R}^3 \end{gathered}$$

<u>Supporting Fact Prediction</u>

$$\begin{gathered} S_{weighed} = \textstyle\sum_{\text{last dimension}} \beta * S_1 \in \mathrm{R}^{n_s} \\ \text{sp-logits} = FFNN(S_{weighed}) \in \mathrm{R}^{n_s} \end{gathered}$$

Span start and end predictions and answer label predictions contributed to cross-entropy loss with respect to training labels. Supporting fact labels were trained with sigmoid multi label cross entropy. All these losses were added and minimized together with Adam Optimizer (Kingma & Ba (2014)).

At test time, answer label is predicted as max(logits). Span prediction from context words is done as in BiDAF Seo et al. (2016a). "The answer span (k, l) where $k \leq l$ with the maximum value of span-start∗span-end is chosen, which can be computed in linear time with dynamic programming." For this part alone, we used the code directly from BiDAF's official implementation available on Github. For supporting fact prediction, we select either all sentences with a probability value above a certain threshold (to be tuned, but currently using 0.3 as in the original paper) or the sentences with the top two predicted probabilities.

5 Experiments

In this section, we describe or various experiments and ablation studies, as well as our best hyper-parameter configuration. We have implemented our model(s) using **Tensorflow** and preprocessing was done using **numpy** and **spacy/nltk**.

5.1 WikiHop

For Wikihop, we started with a simple hierarchical LSTM model and then successively built up the model to the one in Figure 5. On addition of each of the components mentioned in Section 3.4 we observed a significant increase in the accuracy. The results are summarised in Table 4. LSTM+GCN results are summarised in Table 5.

Model Number	**Model Description**	**Best Validation Accuracy**
Model 1	Hierarchical Encoding of paragraphs, followed by Bahdanau Attention weights α and β . LSTM was used to get P_{final} from P_1 15 paragraphs with 150 words each were used, with only word and no character embeddings. Only candidate embeddings from support: C_2 were used to get logits	53.1%
Model 2	Model 1 with character embeddings added to query and support words and self-attention added at word and paragraph levels	56%
Model 3	Model 2 with word+char embeddings added for candidates : $C = [C_1, C_2]$ 20 paragraphs with 110 words each were used.	58.45%
Model 4	Model 3 with co-attention with query added	60.1%
Final Model	Model 4 with query reduction and candidate positional embeddings(to support) added 30 documents with 150 words each were used.	**63.9%**

Table 4: Wikihop Models and Validation Accuracies

Model Number	Model Description	Best Validation Accuracy
Model 1	Hierarchical encoding of paragraphs, followed by intra and inter-document graph(GCN) (refer Section 3.5) with query based α and β	36.38%
Model 2	Model 1 wihout the intra-document graph,i.e., just inter-document GCN	52.19%

Table 5: Wikihop LSTM+GCN Models

5.2 Analysis of Results on WikiHop

The model descriptions and obtained validation set accuracies can be seen in Tables 4 and 5. The current highest non-anonymous submission on the WikiHop leaderboard has a validation accuracy of 68.5%.

Our initial hierarchical LSTM model with just query-based attention weights got an accuracy of 53.1% on the validation set. On adding character embeddings and self-attention, we observed a huge 3% increase in val accuracy. Since **most** of the terms involved in the support and query relevant to predict the answer are **entities**, addition of character embeddings gave the model a huge boost. Replacement of the LSTM at the passage level with self-attention was also a logically intuitive boost since the paragraphs by default have no sequential order to them; plus, self-attention at both levels captures across the sequence information which an RNN cannot capture.

For many of the questions we sampled, we observed that having slight previous knowledge of the candidates(for example, the fact that the candidate in consideration is a city and not a country) was beneficial, or even **necessary** to answer the query. Hence, we added word+character embeddings to the candidate words as well. As expected, this gave a good boost of 2.45%.

To understand the paragraph alone is not enough; it has to be understood in the context of the query. Hence, co-attention was added at both the word and the paragraph level. Empirically, we found out that adding co-attention to just the word representations is better than adding it at both levels. This addition gave us a boost of approximately 1.5%.

For our final model, better interpretation of both query and support was required to predict more answers correctly. We added a **query reduction module** and **candidate position based** embeddings to the support words for the same purpose. The query reduction module contributed to much better understanding of the query in terms of the supports (and in turn, better understanding of supports in terms of the query) and the candidate positional embeddings led to greater understanding of the supports keeping in mind the presence of the candidates in the paragraphs. These additions gave a boost of almost **4%** leading us to our highest validation accuracy of **63.9%**.

We also ran multiple experiments on LSTM+GCN models. Although GCNs have been demonstrated to be highly successful in many fields including NLP (QA in NLP as well), they performed worse in our experiments. They proved to be highly sensitive to the graph used. Since the task is primarily related to entities, we ideated a variety of entity-based graphs rather than the trivial dependency-parse graphs. However, the dataset was too cluttered with non-relevant entities for the GCN to be of any success. Connecting mentions of entities within a paragraph severely brought down the validation accuracy; this is majorly because of the **high number of entities** that each paragraph had and the **non-relevance of most of them to the query asked**. Usage of only an inter-document graph gave better results, but one which was still lesser than the same model without the GCN.
To get conclusive results before we wrote off using GCNs, we ran experiments with multiple hyperparameter configurations. We also sampled the dataset and tried to create logical graphs which would lead from the query to the answer entity. As humans, even we couldn't find any reasonable path which could be automated across all the queries; the logical graphs we *could* make had to be

done by hand and depended on our prior knowledge of the entities involved (and in some cases, knowledge of the answer as well). In some cases, even these connections were not sufficient to arrive at an answer. We considered the usage of an external knowledge base at some point, but because there are derived from Wikipedia, those would be futile and very *heavy* to incorporate. Hence, for this dataset, we decided to proceed without the addition of a GCN.

5.3 Hyperparameter Configuration for Wikihop

Each datapoint in the Wikihop dataset was huge; hence memory issues placed heavy constraints on the model's architecture and hyperparameters.

For all our above models described above, we ran experiments with multiple hyperparameter settings throughout the course of our project. Each of our best validation accuracies mentioned in Tables 4 and 5 were obtained after thoroughly searching through the hyperparameter grid.

Since memory was an excessive constraint for us, we were able to experiment only with relatively small LSTM hidden/output layer sizes. We ran a grid search across 20,30,50,70 and 80, and 2-layer stacked LSTM with 50 units in each layer was our best across multiple model architectures.

Batch size 64 or 48 gave us our best results. Usage of higher batch size gave us memory issues, and usage of batch sizes lesser than 48 *gave much worse results*. Hence, we had to do preprocessing and decide the number of paragraphs and words to be used keeping in mind the batch size constraint.

Our best learning rate was 0.001 and best optimizer was Adam(Kingma & Ba (2014)) conclusively. For any of our model architectures, using any other optimizer, or a learning rate that was even *slighlty* more or less led to considerable worsening of results. Usage of learning rate decay did not help.

We did not fix any set number of epochs to train the model for, or set early stopping based on validation accuracy. Our dataset was extremely complex and sometimes, it would give better accuracies only after training for *many* epochs and overfitting as well.

We used Xavier initialization for all weights. As is general these days, this gave the best results across other initializations.

One of the hyperparameters we tuned excessively was the number of paragraphs and words and characters for the supports, query and candidates. The most problematic of these was what to choose for supports; the support paragraphs were huge in comparison to other non-multihop reasoning datasets, but we also couldn't cut down their size beyond a particular limit since that would we're giving the model less information than necessary. Our default configuration was to give 15 paragraphs with 150 words each, or 20 paragraphs with 110 words each; used with a batch size 64. However we realised that we were not giving enough information to the model to predict the answer; hence for our final,best model we gave 30 paragraphs with 150 words and 10 characters each. For query relation, we gave 50 characters since we have really long relation codes (example : located_in_the_administrative_territorial_entity). For query entity, we gave 5 words with 20 characters each. Ideally, we wanted 20 characters per word for supports as well, since some entity spellings are huge; but this led to memory issues, so we stuck to 10 characters per word for supports and 20 for query entity. For candidates, we allowed 70 candidates per query, with 5 words per candidate and 20 characters per word. For obtaining the embeddings from support, we took and averaged all occurences of the candidates; we ran experiments for *5 separate occurences* of the each candidate but this led to worse results.

For our character embeddings, we used a character CNN (Zhang et al. (2015)) with input embedding size 50, output embedding size and output channel size 100 and filter height 5. We used GloVe 300-dimension embeddings (glove.840B.300d.txt). For candidate positional embeddings, we used 3 dimensional and for query relation we used 300 dimensional learnabale embeddings.

We also experimented with using ELMo embeddings (Peters et al. (2018)) for words instead of GloVe, given that it gives the current SOTA on most datasets. However, memory constraints prevented us from using it to its full effect and hence we didn't gain any accuracy because of shifting to ELMo from GloVe. However, we would further like to try the method in which EntityGCN Nicola De Cao (2019) uses ELMo; that is, run ELMo as preprocessing and dump the embeddings, and then run the usual model starting from those dumped embeddings.

Bahdanau Attention gave us better results as compared to other attention mechanisms available on tensorflow. Gradient clipping did not give any effect. Dropout of 0.7 and 0.8 gave best results.

Our best configuration :

- 300d GloVe embeddings
- Char-CNN with input embedding size 50, output embedding size and output channel size 100 and filter height 5
- Candidate Positional Embedding size 3
- Xavier weight initialisation
- Bahdanau attention for α and β
- dropout of 30%
- 2-layer stacked LSTM encoders with number of hidden units 50 in both layers
- batch size 48
- Adam Optimizer with learning rate 0.001
- 30 documents with 150 words with 10 characters each for supports
- 50 characters for query relation and 5 words with 20 characters each for query entity
- 70 candidates for each query, each with all occurences, with 5 words and 20 characters per word

5.4 HotPotQA

Our experiment with the described model gave a span-prediction F1 score of 0.36 and a supporting fact F1 score of 0.32 with hyperparameter configurations similar to what was used for Wikihop. The baseline model's scores are 58.28 and 66.66 respectively; *however*, these scores are for a very complicated architecture including multiple levels of co-attention and self-attention and residual/loop connections within the model. Now that we have our baseline results, we will proceed into building up our model, similar to the way we did for the Wikihop dataset to achieve state of the art scores.

6 Related Work

Both datasets have leaderboards where many submissions provide alternative approaches to the problem.
Wikihop: The dataset has numerous anonymous submissions that are made public only after the paper is made public. Some of the significant results on the leaderboard follow: Entity GCN by Nicola De Cao (2019) shows the highest (public) score. The authors follow a novel approach of framing it as an inference problem on a graph. Mentions of entities are nodes of this graph while edges encode relations between different mentions (e.g., within- and across-document co-reference). The power of coattention at a coarse and fine level is shown by Victor Zhong (2019). Hiearchies of cottention and self-attention are used by the authors to give a high performance. Other submissions on the leaderboard use various architectures like Memory Networks (Jifan Chen (2018)), a combination of graph convolutional and recurrent networks (Linfeng Song (2018)) and gated recurrent units with corefernce (Bhuwan Dhingra (2018)).
HotPotQA: Almost all submissions on the leaderboard are anonymous and have no linked paper. Being a recent dataset, the best available token embedding is used - BERT. Most top models are a combination of BERT with some model, for example, HiPAR, QRE, GRN, BERT Plus etc. The baseline by the dataset aggregators (Zhilin Yang (2018)) is a stack model of RNNs, self- and bi-attention, which has been comprehensively beaten by the models mentioned above.

7 CONCLUSION AND FUTURE WORK

This project that started off with building a graph based model on Wikihop finally showed the most promising results using hierarchical (LSTM and otherwise) encoding, various types/levels of attention and query refinement. We have experimented extensively on Wikihop and we've come close to current SOTA on the leaderboard. We're exploring and building up an architecture for HotPotQA as well. We also aim to further inspect each component of the model to see exactly what and whether it is learning. One possible way to check the multihop reasoning would be to plot the attention weights between consecutive steps of the query reduction module and visualise the path they take. Our fuure aim in HotPotQA is to analyse the more plots for each model as well as add more interpretability to he model, since that is more important than blindly increasing the empirical accuracy.

The project was also instrumental in showcasing the importance of graph structure for graph based models. Simple changes in the graph structure and pre-processing led to significant performance improvements in the LSTM+GCN model. This claim can further be strengthened by the work on Entity GCN (Nicola De Cao (2019)): which has an interesting graph structure and uses GCNs to give the top public score.

This project opens up avenues to various further directions of work. Subtle improvements to the existing models would show remarkable changes, some of which would be co-reference resolution, word embedding initialisation using ELMo/BERT or even a new attention mechanism.
Another promising direction in which we did some preliminary research was on using *multiplex networks* for the multi-hop QA task. This would be especially interesting if inter- and intra- document graphs are fit into this framework.

8 ACKNOWLEDGEMENTS

First, we would like to thank Professor Mitesh Khapra for guiding us throughout the course of the project and for providing us with AWS resources to run our experiments. We would like to thank Professor Mitesh Khapra and Professor Ravindran Balaraman for providing us with the lab GPU cluster resources to run our experiments. We would like to thank Preksha Nema, for her extensive guidance and help in creating the model architecture and running experiments. Last but not least, we would like to thank Nikita Moghe and Priyesh Vijayan for their help with graph convolutional networks.

9 CONTRIBUTIONS OF AUTHORS

Both the authors, Sahana and Shreyas, worked equally on graph-based models(work on GCNs), on initial experiments in hierarchical LSTM models and on analysis and preprocessing of HotPotQA data. The remainder of the work was done individually by Sahana.

REFERENCES

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Zhilin Yang William W. Cohen Ruslan Salakhutdinov Bhuwan Dhingra, Qiao Jin. Neural models for reasoning over multiple mentions using coreference. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2018.

Greg Durrett Jifan Chen. How to learn (and how not to learn) multi-hop reasoning with memory networks. In -, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Mo Yu Yue Zhang Radu Florian Daniel Gildea Linfeng Song, Zhiguo Wang. Exploring graph-structured passage representation for multi-hop reading comprehension with graph neural networks. In *arXiv preprint arXiv:1809.02040*, 2018.

Ivan Titov Nicola De Cao, Wilker Aziz. Question answering by reasoning across documents with graph convolutional networks. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.

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)*, pp. 1532–1543, 2014.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

Konstantin Lopyrev Percy Liang Pranav Rajpurkar, Jian Zhang. Squad: 100,000+ questions for machine comprehension of text. *Empirical methods in natural language processing (EMNLP)*, 2016.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016a.

Minjoon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. Query-reduction networks for question answering. *arXiv preprint arXiv:1606.04582*, 2016b.

Nitish Shirish Keskar Richard Socher Victor Zhong, Caiming Xiong. Coarse-grain fine-grain coattention network for multi-evidence question answering. In *arXiv preprint arXiv:1901.00603*, 2019.

Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. Constructing datasets for multi-hop reading comprehension across documents. *Transactions of the Association of Computational Linguistics*, 6:287–302, 2018.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pp. 649–657, 2015.

Yao Zhao, Xiaochuan Ni, Yuanyuan Ding, and Qifa Ke. Paragraph-level neural question generation with maxout pointer and gated self-attention networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3901–3910, 2018.

Saizheng Zhang Yoshua Bengio William W. Cohen Ruslan Salakhutdinov Christopher D. Manning Zhilin Yang, Peng Qi. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *Empirical methods in natural language processing (EMNLP)*, 2018.