Contextual Question Answering with HotpotQA

Srinidhi Gopalakrishnan, Sai Divya Sangeetha Bhagavatula

Khoury College of Computer Sciences, Northeastern University
gopalakrishnan.s@northeastern.edu, bhagavatula.s@northeastern.edu

Abstract

The question answering task is commonly used in the field of natural language processing. It provides a quantifiable method to test the intelligence of NLP systems. In such tasks, the input to the system is typically a question and the output is an answer. Systems trained on the entire web can predict the answer from just a question. However, for models which do not have any prior knowledge from large amounts of web data, it is imperative to train the system with additional information, beyond what the input question contains. In this project, we develop such a system - one that takes an input question as well as an arbitrary number of paragraphs that may or may not provide supporting information that can guide the model to the answer. The objective of the system is to intelligently analyze the text in the supporting facts, understand the grammar and phrasing around probable answers and guess the answer from the facts. Hence, supporting facts for a question make the model more intuitive. In this project, we also deal with multi-hop questions that make it more challenging as it requires the model to find and retrieve information from different parts of the context. A good dataset that encompasses all requirements for the question-answer task is HotpotQA, where the questions are diverse and not constrained to any pre-existing knowledge bases or knowledge schemas and provide sentence-level supporting facts required for reasoning, allowing question-answer systems to reason with strong supervision and explain the predictions. We develop a model architecture using combinations of Recurrent Neural Network layers, Attention layers, and Linear layers that not only predict the answer but also whether each of the input paragraphs is a supporting fact that helps in answering the question. We experiment with different hyperparameters in the model to obtain the best model in terms of the closeness of the answers predicted with the actual answers. Finally, we retrospect on the results and suggest areas of improvement to build a more robust question-answer model.

Introduction

In this project, we build an effective question-answer model that can answer multi-hop questions, given a list of paragraphs that may or may not be a supporting fact. As shown in the following example, multi-hop questions are long, noisy, and require a lot of contexts to answer. The model needs to learn that to answer the overall question, one or more sub-questions need to be answered and those need to be used to generate the final answer. Previous question answering research has been primarily focused on single-hop questions which have relatively simple structures and can be answered using information contained in one paragraph. This is predominantly due to the lack of datasets that have a large percentage

of multi-hop questions. This has in turn not paved the way to more advanced NLP frameworks that can tackle this task effectively. Examples of single-hop question-answer datasets include SQuAD, TriviaQA, and QAngaroo (constructed using existing knowledge bases).

Question Type	Question	Method	Context
Single-hop	What is 2018's highest-grossing movie?	Locate the Movie	Straightforward, one paragraph
Multi-hop	When was the highest-grossing movie of 2018 released?	Locate the Movie -> Find the date	Long, noisy, multi-paragraph contexts

Table 1: Examples of single-hop and multi-hop questions

Thus, the HotpotQA dataset was created. It is a dataset with Wikipedia-based question-answer pairs with four key features: (1) the questions require finding and reasoning over multiple supporting documents to answer; (2) the questions are diverse and not constrained to any pre-existing knowledge bases or knowledge schemas; (3) we provide sentence-level supporting facts required for reasoning, allowing QA systems to reason with strong supervision and explain the predictions; (4) we offer a new type of factoid comparison questions to test QA systems' ability to extract relevant facts and perform necessary comparisons.

To build the question-answer model, we referred to past work in this space - a combination of bidirectional RNNs, Attention layers, and linear layers that take input word and character embeddings and map them to the output layers - answer span (can be an actual answer, yes, or no), as well as predicting whether the input paragraphs denote supporting facts. This model pipeline is known in the literature to be robust in training models on large amounts of QA data and is also known to work for multiple QA datasets. We were able to train a model on a smaller scale and although the results are subpar to the expectations set by research, the learnings from developing the model from scratch as well as experimenting with different model iterations were useful in understanding the deep learning concepts related to question-answering and the purpose of each part of the model as well as preprocessing steps.

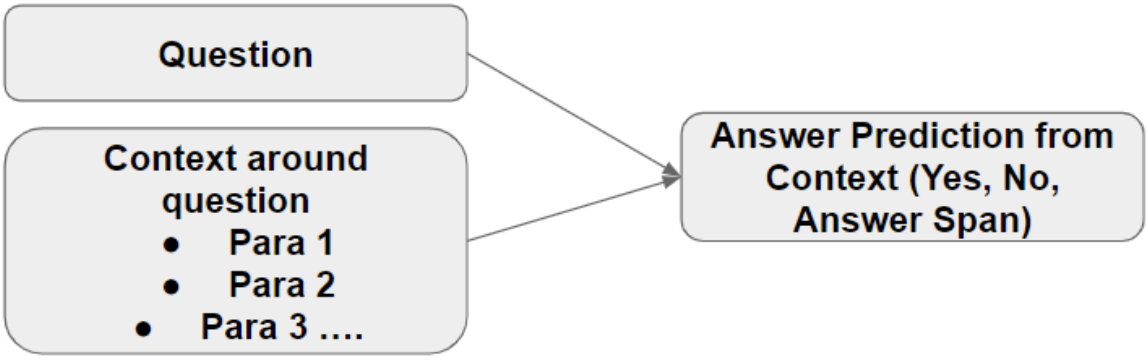

Figure 1: An overview of model inputs and output

Data and Preprocessing

The HotpotQA dataset contains 113k question-answer pairs. Including the diverse questions and providing supporting facts, this dataset has distractor facts that help the model identify the true supporting facts from false ones. This helps in building an intuitive and robust one. This dataset contains various types of answers. The below table describes the varied nature of the data.

Answer	% (In Sample)	Example
Person	30	King Edward II
Group / Org	13	Apalachee
Location	10	California
Date	9	10th Century
Number	8	79 million
Artwork	8	Die schweigsame Frau
Yes / No	6	-
Adjective	4	conservative
Event	1	Academy Awards
Other proper noun	6	Cold War
Common noun	5	comedy, men and women

Table 2: Types of answers in the dataset

The training data consists of three types, depending on the level of difficulty. Train-easy contains single-hop question-answer pairs whereas train-medium and train-hard consist of multi-hop and hard multi-hop questions respectively.

Pre-processing the data plays a vital role in any machine learning problem, especially while dealing with the text data. As a part of data pre-processing, we performed sentence and word tokenizations on all the question-answer pairs and their supporting facts. Recorded the title and number of supporting facts for each question. Among all the supporting facts, distinguished between the actual and distracting facts. Among the available answers, the best one is found and its indices are recorded. If the best answer is a paragraph, the best indices are the start and end tokens of the paragraph. Also, assigned best indices of the questions with answers as ‘yes’ or ‘no’. Missing answers are filled as ‘random’. GloVe embeddings are collected from here. The character embeddings are randomly allocated but will be trained in the modeling stage. All the processing has been performed on the entire data, including train and test. The pre-processed data contains the word and character-level embeddings, start and end tokens of the best answer, which are fed to the model as an input.

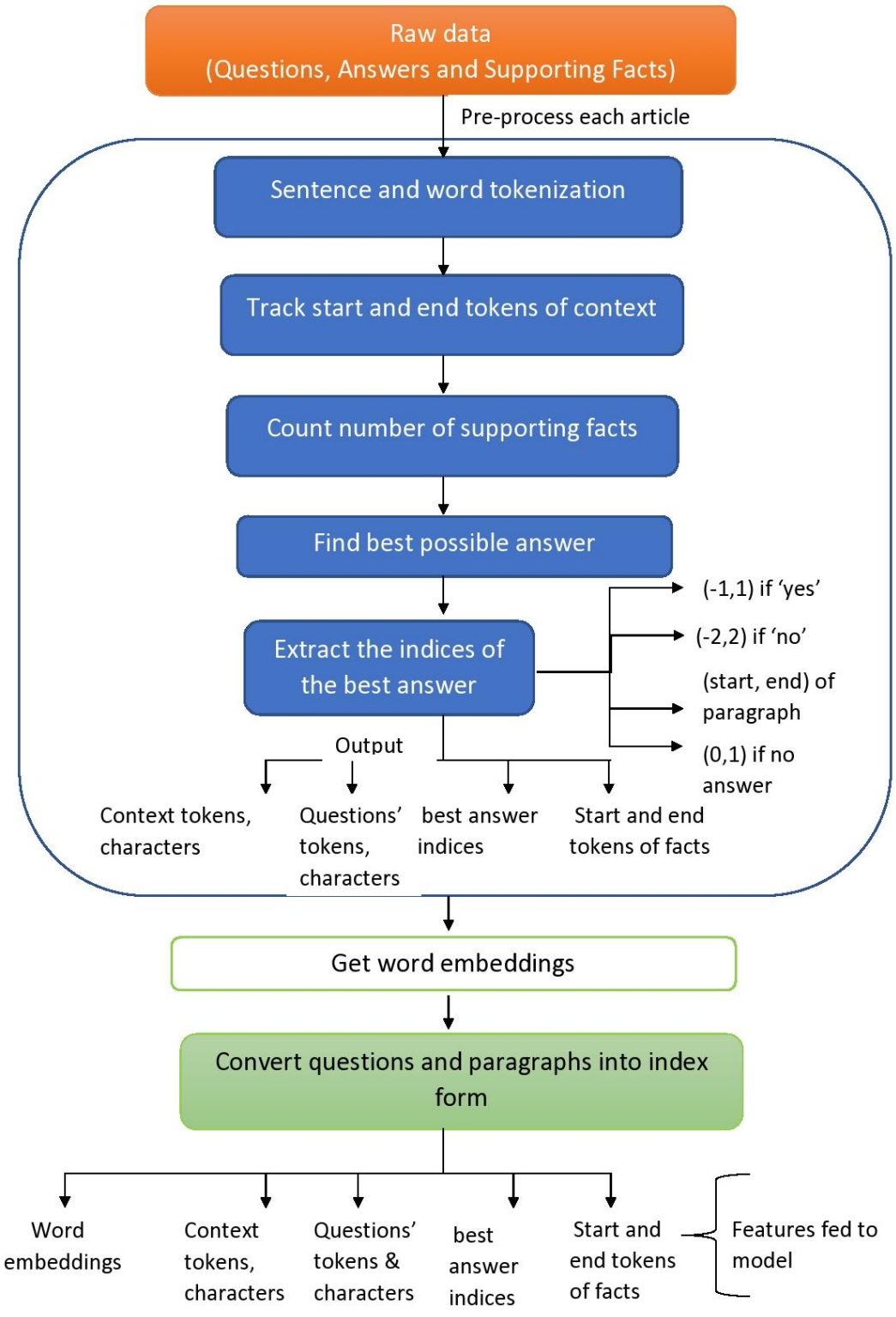

Figure 2: Pre-processing flow chart

Modeling

The output of the preprocessing step yields the word and character embeddings of the questions and paragraphs, as well as the start and end indices of each word in the paragraphs, including the answer. The prediction of the model comes from sequential linear layers that perform binary classification on the facts regarding whether it is a supporting fact or not, which leads to the prediction of the start and end tokens of the answer and then the answer itself. The answer can be 'yes', 'no', or a range of characters between the predicted start and end tokens.

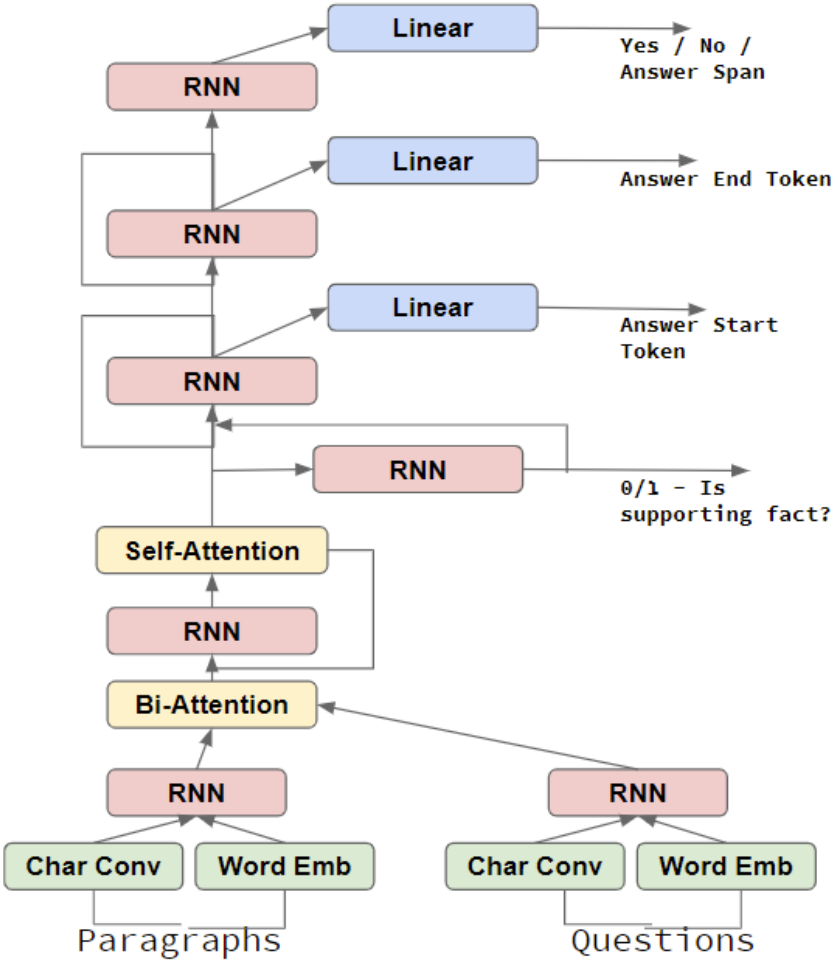

Figure 2: Model Architecture. Supervision of supporting facts employed in answer prediction.

To begin, the model converts the random character embeddings created during preprocessing into actual embeddings learned from the data by performing 1D convolution followed by max-pooling on the input embeddings. This is done separately for questions and paragraphs. The learned character and word embeddings (from GloVe, not updated during training) are concatenated and go to bidirectional encoder Gated Recurrent Unit (GRU) layers to map question and passage embeddings to context-aware embeddings. These outputs are sent to a bidirectional attention mechanism that generates the query-aware context representations by combining context words and question words. The query-to-context vectors are then sent to a residual bidirectional GRU and self-attention mechanism between the vectors and

themselves. This residual self-attention vector is appended to the output of the bi-attention layers. This output first goes to an RNN component that generates strong supervision in answer predictions. For each sentence, we concatenate the output of the self-attention layer at the first and last positions and use a binary linear classifier to predict the probability that the current sentence is a supporting fact. This is done by minimizing the binary cross-entropy loss of the classifier. This prediction is then sequentially used to predict first the start token of the answer (with another GRU and linear layer), second, the end token of the answer, and finally the answer itself, which could be generated from the start and end tokens, or predicted as 'yes' or 'no'.

Model Components

GloVe Word Embeddings: GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. We used these pre-trained word vectors on 2.2 million words, each word represented by a vector of 300 floating-point numbers. The Euclidean distance (or cosine similarity) between two-word vectors provides an effective method for measuring the linguistic or semantic similarity of the corresponding words. The GloVe model is trained on the non-zero entries of a global word-word co-occurrence matrix, which tabulates how frequently words co-occur with one another in a given corpus and the training objective of GloVe is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence.

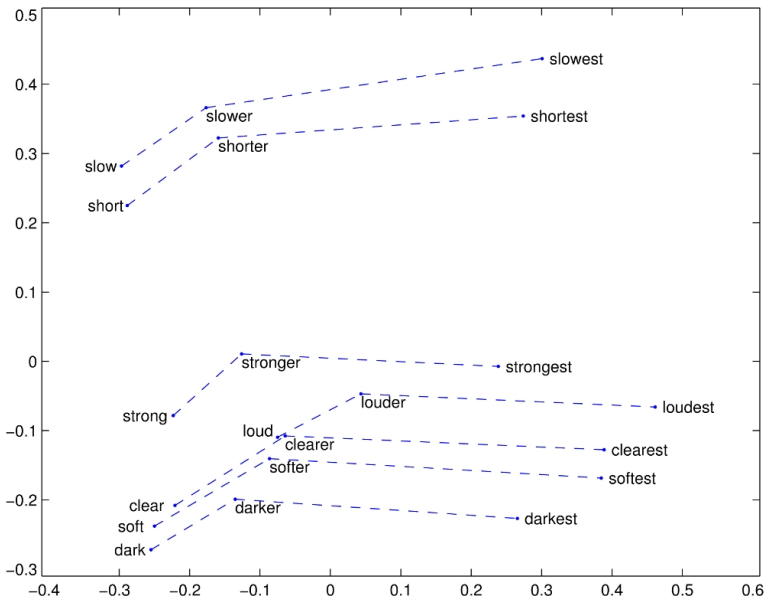

Figure 3: Linear relations between adjectives and their comparative and superlative representations

Gated Recurrent Units (GRU): As we know, Recurrent Neural Networks (RNN) possess the ability to learn sequences in the data and retain memory. However, if a sequence is too long, they have a hard time carrying information from earlier time-steps to later ones. During backpropagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks' weights. The vanishing gradient problem is when the gradient shrinks as it back propagates

through time. If a gradient value becomes extremely small, it doesn't contribute too much learning. Hence, they are not ideal for this use case where we have several long paragraphs to model, even if the questions themselves are short.

GRUs and LSTMs (Long Short Term Memory) were created to tackle this vanishing (or exploding) gradient problem. They have internal mechanisms called gates that selectively forget and retain parts of a long sequence, regardless of how recently the unit encountered that portion. GRUs have a computational advantage over LSTMs as they do not have additional hidden states to remember and are thus faster to train. They only have a reset gate to decide how much past information to forget and an update gate to decide what information to throw away and what new information to add.

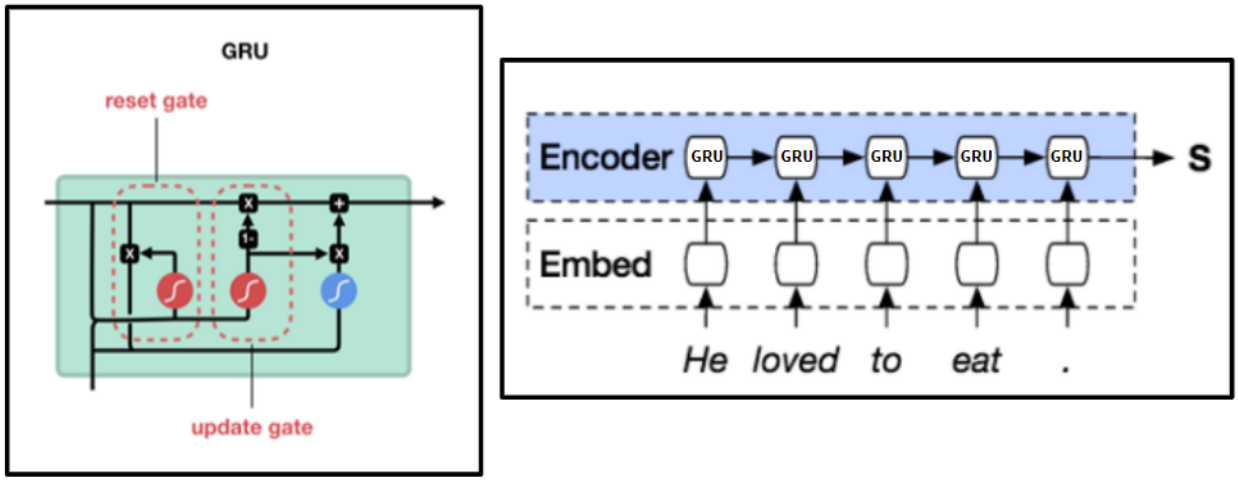

Figure 4: The inside working of GRUs

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

Equation 1: Formula of the update gate. When x_t is plugged into the network unit, it is multiplied by its own weight W(z). The same goes for h_(t-1) which holds the information for the previous t-1 units and is multiplied by its own weight U(z). Both results are added together and a sigmoid activation function is applied to map the result between 0 and 1.

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

Equation 2: Formula for the reset gate. Identical to the update gate, except for the weights.

$$h_t^{'} = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

Equation 3: Formula for the current memory content. Uses the output of the reset gate as weights to apply to output of the previous time step to determine which parts to forget. Nonlinear activation tanh applied.

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t^{'}$$

Equation 4: Final output of the timestep which is a weighted sum of past output and current memory based on weights from the update gate.

Attention Mechanism: Used to identify specific information in the context relevant to the question, that is, query-aware context representation. It takes the words in the context and questions and creates a summary of the context, focusing on the question. It returns the weighted mean of words in context,

which is chosen according to the relevance of each word in the context and is computed at each time step, along with representation from the previous layers. In the attention layer, when the model is trying to predict the next word, it searches for a set of positions in a source sentence where the most relevant information is concentrated. The model then predicts the next word based on context vectors associated with these source positions and all the previous generated target words.

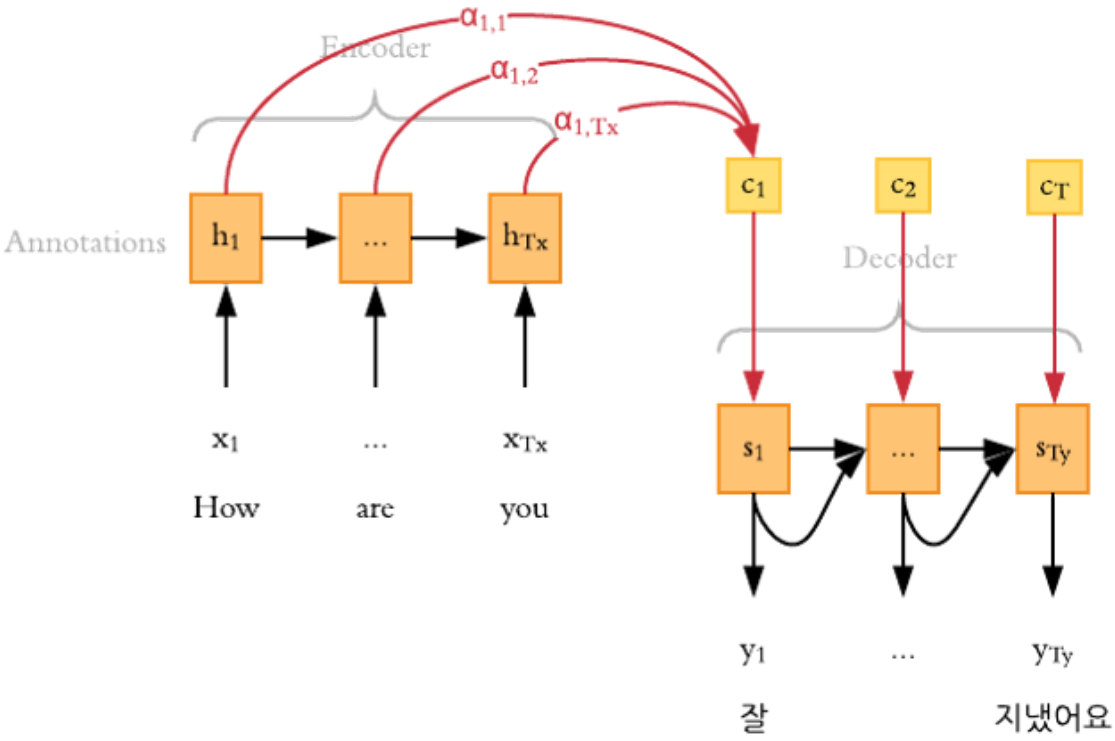

Figure 5: The attention mechanism, when translating English to Korean. In our case, the output is instead weights of each of the words in the paragraphs based on relevance to the question. This is sent to a linear layer to get the query-aware context representation.

Linear Classifier: In the last layers of our model a bidirectional GRU is applied, followed by a linear layer that computes answer start scores for each token. The hidden states of that layer are concatenated with the input and fed into a second bidirectional GRU and linear layer to predict answer end scores. The softmax operation is applied to the start and end scores to produce start and end probabilities, and we optimize a weighted negative log likelihood of selecting correct start and end tokens, where the weight depends on how much of the loss depends on the supporting fact prediction model.

Dropout: We also employ variational dropout, where a randomly selected set of hidden units are set to zero across all time steps during training. We dropout the input to all the GRUs, including the word embeddings, as well as the input to the attention mechanisms, at a rate of 0.2. We also mask future parts of the sequence representations from the self-attention layers during answer prediction to ensure we are only predicting from past tokens.

Experiments and Evaluation

Once we had the processed text and model architecture, we split the train and development datasets into batches based on different batch sizes (10, 25, 100). Due to computing constraints, we were able to train on different sample sizes (1000 and 2000 samples chosen at random out of 90k train examples) with only one epoch. Hence, the results cannot be used as definitive proof of the efficacy of the model. Past results show the model has very good performance (> 50% exact match between the actual and predicted answers) and we were not able to achieve that high score in our experiments. We also experimented with different GRU sizes - both in terms of the number of GRUs in character and word RNNs. We kept the learning rate and dropout rate constant (0.1 and 0.0 respectively).
The evaluation was done on train, development, and test sets with and without distractor paragraphs. The metrics to evaluate model performance were Exact Match, Precision, Recall, and F1 Score after normalizing actual and predicted answers (remove articles and punctuation, make lowercase, fix white spaces).

Exact Match: # Instances Actual Answer = Predicted Answer after normalizing text

Precision, Recall, F1
Precision: Number of words predicted correctly among predicted tokens
Recall: Number of words predicted correctly among ground truth tokens
F1 Score: 2PR/P+R

Figure 6: Definition of Evaluation Metrics

Results: From the graph below, we see that we have no cases where the actual and predicted answers exactly match each other, even in training. Also, precision, recall, and F1 scores are less than 1%, far less than research where exact match rates are ~52% and F1 scores are greater than 60%. The model performs better on data without distractors but has very poor performance overall.

Figure 7: Model evaluation on train and test sets, with and without distractors.

There are several reasons for this performance - the first being lack of data. When analyzing the predicted answers, we observed that 95% of them were just '<\t>' without any other text. This means that the model was not fed enough data and given enough time and epochs to learn the intricacies in the questions and paragraphs. The 1000 train samples may not have covered all topics and types of questions, which has made the model spit out random or empty answers. Also, none of the yes/no answers were predicted correctly and some of them had other answer spans. This can be improved through leveraging GPUs or Google Colab, which can hold tens of times more memory than a local machine and can also enhance the running time during training and prediction. In our experiments, training one epoch with 1000 samples took 7 hours. However, we believe the model architecture lends itself to the question answering task and minor tuning and larger memory will greatly enhance its performance.

Conclusion

Multi-hop question answering proves to be a challenging task that requires training samples in the millions and a complex architecture to understand language intricacies and question-context relationships. A lot of models have been developed in research using the HotpotQA dataset and it was a rewarding experience learning about the experiments done and why our particular architecture works well with this data. As explained in the results, the major bottleneck appears to be a memory to preprocess the data and train the model. Hence, we suggest investing in resources that can leverage the entire training dataset. State-of-the-art models, particularly BERT-based ones, have managed to achieve close to human performance on multiple datasets so it would be exciting to use the output supporting facts from the baseline model as input to BERT, which can then predict the answer. This is given a large amount of noise in the context that does not confuse BERT's bidirectional self-attention layers. Also, each component in the preprocessing and model architecture can be played around with (changing the word embeddings, hyperparameter tuning of GRU and Linear layers, using answer predictions to predict supporting facts, etc.). Once a more reliable system has been developed, the final model can be used to create an online chatbot that can answer any input question, provided it is provided with appropriate supporting facts.

Contributions

Srinidhi: Research on existing work, understanding data, preprocessing all datasets, coded model architecture, developed initial model, model evaluation, and calculation of metrics.

Sangeetha: Research on existing work, understanding data, ran multiple iterations after the initial model by making modifications to the training sample, batch size, hyperparameters, and ran predictions on the train and test sets.

References

Christopher Clark, Matt Gardner, 2017. Simple and Effective Multi-Paragraph Reading Comprehension. arXiv:1710.10723v2 [cs.CL]

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, Christopher D. Manning, 2018. HOTPOTQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. arXiv:1809.09600v1 [cs.CL]

Chris Wang, Yilun Xu, Qingyang Wang. Multi-hop Question Answering on HotpotQA.

Andrea Galassi , Marco Lippi , Paolo Torroni. Attention in Natural Language Processing. IEEE Transactions on Neural Networks and Learning Systems.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, 2014.

Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: CoRR abs/1409.0473 (2014). arXiv: 1409.0473

Wenpeng Yin, Katharina Kann, Mo Yu, Hinrich Schutze. Comparative Study of CNN and RNN for Natural Language Processing, 2017.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP).

Shimi Salant, Jonathan Berant. 2018. Contextualized word representations for reading comprehension. In Proceedings of the 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, Hannaneh Hajishirzi. Bidirectional Attention Flow for Machine Comprehension, 2017.

Link to project repository: https://github.com/srinidhig/QA_chatbot