Bidirectional Attention Flow with LSTM Networks for Question Answering

James Li
Department of Computer Science
Stanford University
Stanford, CA 94305
dawwctor@stanford.edu

Abstract

SQuAD is a reading comprehension challenge centering around answering questions given a context Wikipedia paragraph. In this paper we discuss our various methods and approaches to tackling the SQuAD Question Answering Challenge. We chose to implement a variety of GRU and LSTM RNN network models, all of which are derived from the provided CS 224N Baseline model. The improvements we added were heavily inspired by the Bi-Directional Attention Flow (BiDAF) model as described in [1]. In addition, we also added slight modifications to the context hyperparameters and start and end pointer decision processes. This allowed our final SQuAD RNN model to obtain a test F1 score of 75.226 and an Exact Match (EM) score of 65.32.

1 Introduction

1.1 Problem

SQuAD is a reading comprehension challenge centering around correctly answering questions given a context Wikipedia paragraph. This is an interesting problem from both a research and practical perspective, as this challenge provides a measure of how well different systems can understand and process text to make meaningful judgments and decisions, which could lead to future AIs answering more advanced questions and better deducing the wants and desires of people. One important thing to note is that the answers to the SQuAD questions are drawn directly from the text, so any model would only need to select the correct starting and ending point of the answer rather than produce any original answer text, as stated in [2]. Because of this distinction, model performance is measured in two different ways: F1 and Exact Match (EM). The EM score is based off of the proportion of questions that we answer correctly with an answer that precisely matches any of the human-provided answers, while the F1 score is more lenient and considers both the precision (what proportion of the model's answer is the true answer) and recall (what proportion of the true answer is contained in the model's answer) of the model's answer.

1.2 Overall Approach

While attempting the SQuAD Question Answering Challenge, our overall approach focused mainly on extending upon the most crucial parts of the system introduced in the baseline, namely the attention layer, rather than adding numerous new subsystems and modifying other parts of the data preprocessing. We centered around the BiDAF model, as it offered the most immediate relevant changes to the attention model. Furthermore, after implementing the bidirectional attention flow part as described in [1], we determined that we could further increase performance by adding their

other provided suggestions for changes rather than attempting to integrate different models together for more ambiguous improvements.

We also plotted the training and development context, question, answer, and span files to analyze the general structure of the given data. This allowed us to make smarter initial hyperparameter choices that greatly reduced the memory footprint of our model and informed us of the possible trade-offs that could be made if we wanted to further adjust any hyperparameters.

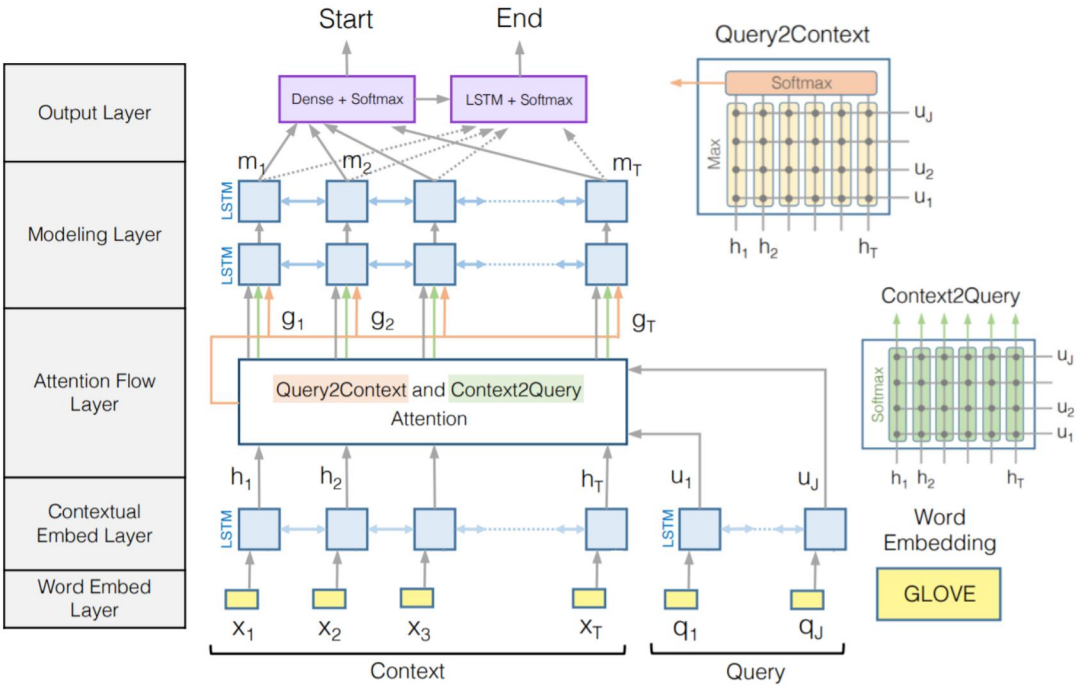
2 Background and Related Work

Although SQuAD is less than two years old, it has already led to many research papers and significant breakthroughs in building effective reading comprehension and question answering systems. There is a public online leaderboard (https://rajpurkar.github.io/SQuAD-explorer/) where anyone can see the various high-performing models and their F1 and EM scores. Many of the top performing models are ensemble methods, as this allows the weaknesses of any one particular model to be partially compensated by the strengths of the other models present, reducing overfitting on the training and development dataset and giving a higher test performance overall.

3 Approaches and Models

3.1 Model Architecture Details

Our model draws heavily from the provided CS 224N baseline model and the BiDAF model. Our final model consists of a Word Embedding layer, a Contextual Embedding layer, a Bidirectional Attention Flow layer, a Modeling layer, and an Output layer, which produces starting and ending span pointers [1].



3.2 Word Embedding Layer

The purpose of this layer is to map each word present in the context and question into a vector space of word embeddings so that we can extract more information than just its presence from a particular word. Our word embedding vectors are GloVe vectors of dimensionality 100, with a total of 400k unique lowercase words formed from our corpus [2]. These word vectors were pre-trained using

unsupervised learning on the aggregated global word-word co-occurrence statistics derived from 6 billion words of Wikipedia and Gigaword [2]. These embedding matrices are used to extract the word vector embeddings of the words present in both the questions and the contexts, which are then fed to the Contextual Embedding Layer.

3.3 Contextual Embedding Layer

For each SQuAD training example of context, question, and answer, the context is represented as an N length sequence of d-dimensional fixed GloVe word embeddings and the question is represented as an M length sequence of d-dimensional fixed GloVe word embeddings [2]. These embeddings are fed into a 1-layer bidirectional LSTM RNN, the weights of which are shared between the context and question, as the context clues that could help us refine the word embeddings present in the question are also likely to be present in the context, since the answer for the question is located within the context.

We use LSTMs instead of GRUs because both the hidden size and the context length are fairly large, so the increase in long-term attention dependency for question answering and decreased chance of vanishing gradients is worth the 20% increase in parameter number and model complexity.

The bidirectional LSTM RNN returns refined vectors of forward and backward hidden states for both the context and the question, which are then concatenated to form two sequences of $2h$-dimensional hidden vectors [2]. This gives us $\mathbf{c}_i \in \mathbb{R}^{2h}, \forall i \in \{1, \ldots, N\}$ for the context and $\mathbf{q}_j \in \mathbb{R}^{2h}, \forall j \in \{1, \ldots, M\}$ for the question. These context and question hidden states are then fed to our Attention layer.

3.4 Bidirectional Attention Flow Layer

Instead of utilizing the Basic Attention provided in the baseline, we chose to implement our Attention layer using Bidirectional Attention Flow as described in the BiDAF paper [1]. The reason we chose to use bidirectional attention flow is because we reasoned that information from both the beginning and the end of the context would be important for answering the question, especially since we have access to the entire context text beforehand. Thus, questions that are brought up early on in the text but only answered later on could still be reasonably answered without significantly neglecting any particular part of the context.

3.4.1 Basic Attention

To explain exactly how Bidirectional Attention Flow improves upon the baseline model, we will first explain how Basic Attention works. As described in [2], the context hidden vectors $\mathbf{c}_i$ and the question hidden vectors $\mathbf{q}_j$ are used in basic dot-product attention to produce an attention distribution $\alpha^i \in \mathbb{R}^M$, where

$$\mathbf{e}^i = [\mathbf{c}_i^T \mathbf{q}_1, \ldots, \mathbf{c}_i^T \mathbf{q}_M] \in \mathbb{R}^M$$
$$\alpha^i = \text{softmax}(\mathbf{e}^i) \in \mathbb{R}^M$$

This attention distribution α^i is then used to create a weighted sum of the question hidden vectors $\mathbf{q}_j$, producing the attention output $\mathbf{a}_i$, where

$$\mathbf{a}_i = \sum_{j=1}^{M} \alpha_j^i \mathbf{q}_j \in \mathbb{R}^{2h}$$

This attention output $\mathbf{a}_i$ is then concatenated to the context hidden vector $\mathbf{c}_i$ to obtain the blended representation $\mathbf{b}_i$, where

$$\mathbf{b}_i = [\mathbf{c}_i; \mathbf{a}_i] \in \mathbb{R}^{4h} \quad \forall i \in \{1, \ldots, N\}$$

This blended representation would then normally be fed into the Output layer. However, as our model utilizes Bidirectional Attention Flow, this is not the case.

3.4.2 Bidirectional Attention Flow

Bidirectional Attention Flow differs from Basic Attention in how it calculates the attention outputs and blended representations, having separate weights for the cases when the context attends to the question and when the question attends to the context. This could reveal more complex latent structures between the context and question and also adds asymmetry to the model, so that the question can ignore parts of the context that are irrelevant and that each context word can associate itself with the most important parts of the question. In addition, keeping the context to question attention and question to context attention separate reduces the information loss from summarizing the total attention at each time step [2].

As before, we have the context hidden state vectors $\mathbf{c}_i$ and the question hidden state vectors $\mathbf{q}_j$. We seek to compute a similarity matrix $\mathbf{S} \in \mathbb{R}^{N \times M}$, where $\mathbf{S}_{ij}$ is a similarity score between pairs of context and question hidden vectors. Creating a new weight vector $\mathbf{w}_{\text{sim}}^T \in \mathbb{R}^{6h}$, we get

$$\mathbf{S}_{ij} = \mathbf{w}_{\text{sim}}^T[\mathbf{c}_i; \mathbf{q}_j; \mathbf{c}_i \circ \mathbf{q}_j] \in \mathbb{R}$$

The calculation of this matrix initially gave our model Out-Of-Memory errors due to the massive tensors that needed to be constructed to accurately compute each element of the similarity matrix, necessitating a decrease in batch size to 75 and later 60 to be able to properly train the model. However, through egregious use of broadcasting and dot product decomposition, we were able to modify our model design to be more memory efficient and revert the batch size to the initial default value of 100.

3.4.3 Context-to-Question (C2Q) Attention

C2Q attention signifies which question words are most relevant to each context word, which is useful to determine how often a particular word should be considered for a question [1]. Similar to the baseline Basic Attention calculations, we calculate attention distributions and outputs $\mathbf{a}_i$

$$\alpha^i = \text{softmax}(\mathbf{S}_{i,:}) \in \mathbb{R}^M \quad \forall i \in \{1, \ldots, N\}$$
$$\mathbf{a}_i = \sum_{j=1}^{M} \alpha_j^i \mathbf{q}_j \in \mathbb{R}^{2h} \quad \forall i \in \{1, \ldots, N\}$$

3.4.4 Question-to-Context (Q2C) Attention

Q2C attention signifies which context words are most similar to the question words, which is useful to determine which words should be focused on to answer the question [1]. In order to obtain the Q2C attention output $\mathbf{c}'$, we calculate

$$\mathbf{m}_i = \max_j \mathbf{S}_{ij} \in \mathbb{R} \quad \forall i \in \{1, \ldots, N\}$$
$$\beta = \text{softmax}(\mathbf{m}) \in \mathbb{R}^N$$
$$\mathbf{c}' = \sum_{i=1}^{M} \beta_i \mathbf{c}_i \in \mathbb{R}^{2h}$$

Combining the two attention types, we obtain a final blended representation

$$\mathbf{b}_i = [\mathbf{c}_i; \mathbf{a}_i; \mathbf{c}_i \circ \mathbf{a}_i; \mathbf{c}_i \circ \mathbf{c}'] \in \mathbb{R}^{8h} \quad \forall i \in \{1, \ldots, N\}$$

This blended representation vector is then fed to the Modeling layer.

3.5 Modeling Layer

The purpose of the modeling layer is to use the blended representation from the attention layer to express how the context and question explicitly interact with each other. This differs from the Contextual Embedding layer, which captures the interaction amongst the context and question independently [1]. We use two layers of bidirectional LSTM RNN networks to get a matrix $M \in \mathbb{R}^{N \times 2h}$.

Because the blended representations are fed to an RNN and these outputs are fed into another RNN, we can obtain column representations of complex relationships in M about each word with respect to the entire context and question. This output matrix M is then passed to the output layer to predict the start and end pointers for the answer.

3.6 Output Layer

Our Output layer signifies the model finding a portion of the context to answer the question using the Modeling layer output [1]. We obtain the start position probability distribution over the context with the weight vector $\mathbf{w}_(p^{\text{start}}) \in R^{10h}$ through

$$p^{\text{start}} = \text{softmax}(\mathbf{w}^T_{(p^{\text{start}})}[\mathbf{b}; \mathbf{M}])$$

However, for the end position probability distribution, we first pass M to another bidirectional LSTM layer to obtain $\mathbf{M}^2 \in \mathbb{R}^{N \times 2h}$. This is partly to represent a possible relationship between the starting and ending predicted positions so that they are not decided independently when answering the question. Using another weight vector $\mathbf{w}_(p^{\text{end}}) \in R^{10h}$, we get

$$p^{\text{end}} = \text{softmax}(\mathbf{w}^T_{(p^{\text{end}})}[\mathbf{b}; \mathbf{M}^2])$$

3.7 Loss

For our loss function, we take the sum of the cross-entropy loss for the start and end locations. If the true start and end locations are $i_{\text{start}} \in \{1, \ldots, N\}$ and $i_{\text{end}} \in \{1, \ldots, N\}$, then the loss for a single example is

$$\text{loss} = -\log p^{\text{start}}(i_{\text{start}}) - \log p^{\text{end}}(i_{\text{end}})$$

During training, this loss is averaged across the batch and minimized with the Adam optimizer [2]. Although the BiDAF paper uses the AdaDelta optimizer, we chose to stick with the Adam optimizer, as we reasoned that the default hyperparameters would be initially optimized to perform well with the development set. The AdaDelta optimizer, on the other hand, would require additional hyperparameter tuning, which would consume our time and resources without a guaranteed increase in performance. Thus, we decided that we could gain more substantial performance improvements by focusing our efforts elsewhere.

3.8 Span Prediction

When our model is predicting an answer, given a context and a question, we simply take the argmax over p^{start} to obtain the predicted start pointer ℓ^{start}. For the predicted end pointer ℓ^{end}, we also take the argmax over p^{end}, but only in the positions of the distribution after ℓ^{start}, so that it is guaranteed that $\ell^{\text{end}} \geq \ell^{\text{start}}$ and we do not have any degenerate answers of empty string. This gives us the model's predicted answer span $(\ell^{\text{start}}, \ell^{\text{end}})$.

4 Experiments

4.1 Data

The context paragraphs are drawn from Wikipedia while the questions and answers were crowd-sourced using Amazon Mechanical Turk. In total, there are about 100K different question and answer pairs [2].

4.2 Experiment Details

We trained, tested, and compared five different models throughout our training: Baseline, BiDAF with GRU, BiDAF with LSTM, and BiDAF with the 2-layer LSTM Modeling layer with and without forced answer ending span positioning. The Baseline model is the initial model provided by the CS 224N staff. The BiDAF with GRU model is the Baseline model but with the bidirectional attention flow layer instead of the basic attention layer. The BiDAF with LSTM model is the BiDAF with

GRU model but with all of the GRU cells in the contextual embedding layer replaced with LSTM cells. The BiDAF with the 2-layer LSTM Modeling layer is the BiDAF with LSTM model but with the additional implemented Modeling layer as described [1]. The final model is the one described in this paper, and includes the enforced boundary conditions on the predicted answer end position.

4.3 Hyperparameters

All experiments were run with the same hyperparameters barring context length and batch size. We used a learning rate of 0.001, a maximum gradient norm of 5.0 for gradient clipping, a dropout probability of 0.15, a hidden layer size of 200, a question length of 30, and an embedding size of 100. For the baseline, we used a context length of 600, but we later reduced this to 400 for the rest of the models after performing some analysis. The batch size was 100 for the baseline, but due to Out-Of-Memory errors, we had to reduce the batch size to 75 for the BiDAF with GRU and LSTM cell models and to 60 for the BiDAF with the 2-layer LSTM Modeling layer model. However, due to code restructuring and modifications to our calculations, we were able to save more memory again and increase the batch size back to 100 for the final model with all implemented improvements, greatly increasing training speed.

4.4 Runtime Details

We trained all models for roughly 6-12 hours for at least 15,000-20,000 iterations for each model, as per the suggestion in the project handout for avoiding overfitting on the training dataset [2]. Thus, the majority of our best checkpoints and model states that were used for evaluation gravitated around the 12,500-17,500 iteration mark, with the sole exception of our final BiDAF model with both our additional improvements and the original batch size of 100 afforded from the earlier memory savings, which achieved its best performance at the 9,500 iteration mark. This was extremely surprising, as our model had greatly increased in complexity from the baseline with several additional RNN layers and yet was still able to reach convergence at a much better value much more quickly. This could be due to a combination of factors, the most notable of which were the increase in batch size to normal baseline levels and the decrease in context length by a third, decreasing the total search space of possible start and end pointer prediction locations.

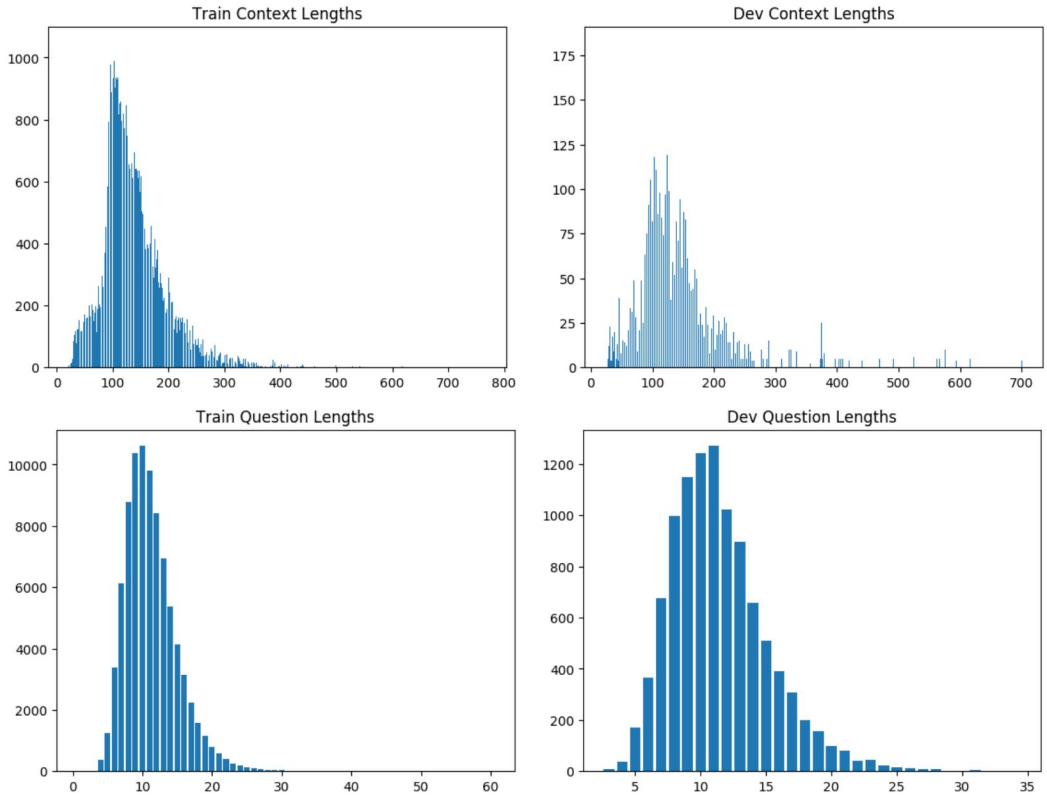
4.5 Evaluation Metrics

We measured model performance by calculating F1 and EM (Exact Match) scores. Given a set of true answer spans and our predicted answer span, the EM score is a binary classifier stating whether our predicted answer span exactly matches any of the true answer spans, while the F1 score is the highest harmonic mean of the precision and recall of our predicted answer span with each true answer span. The precision is the proportion of the predicted span that contains the true span while the recall is the proportion of the true span that is contained within the predicted span.

4.6 Data Preprocessing

As stated earlier, we plotted the training and development context, question, answer, and span data files to obtain useful meta-information for our hyperparameter tuning. The main motivation for doing this was because implementing the BiDAF model with the standard batch size of 100 caused my RNN model to generate Out-Of-Memory errors, as my original implementation of BiDAF created very large high-dimensional tensors when calculating the similarity matrix S.

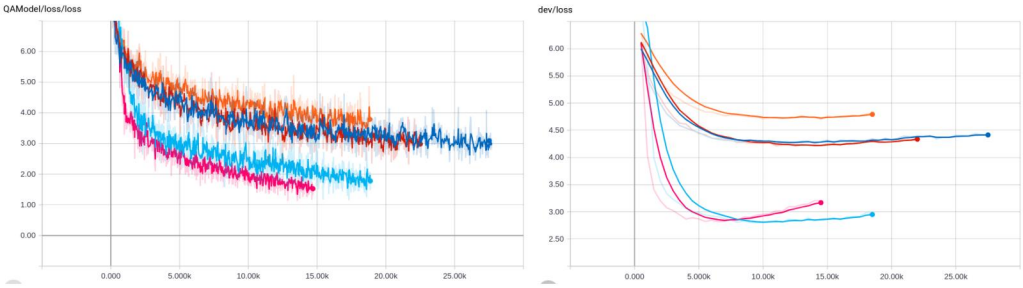
According to our graphs, the majority of the context paragraphs in the train and dev dataset were less than or equal to 400 words in lengths, allowing us to reasonably reduce the context length hyperparameter to 400 from 600 for savings in memory and complexity without a significant decrease in performance. Moreover, the majority of the questions in the train and dev dataset were less than or equal to 30 words in length, meaning that the prior hyperparameter value of 30 for the question length was not unreasonable.



4.7 Results

The performance of our various models on the development set are shown below. On the test set, our final BiDAF model achieved an F1 score of 75.226 and an EM score of 65.32, actually performing better on the test set than on the development and training set.

Model	Train F1	Train EM	Dev F1	Dev EM
Baseline	39.088	32.469	42.648	33.567
BiDAF with LSTM	46.761	39.753	50.403	40.833
BiDAF with 2-Layer Modeling and Forced End	66.543	59.877	74.873	64.74



In the above graphs, Orange represents the Baseline, Blue represents the BiDAF + GRU, Red represents the BiDAF + LSTM, Cyan represents BiDAF + Modeling Layer, and Pink represents our final BiDAF + Modeling + Forced End model. From these loss graphs we can see that the largest improvement came from implementing bidirection attention flow, hence the jump from Orange to Red/Blue, and from implementing the Modeling layer, hence the jump from Red/Blue to Cyan/Pink. We believe this is due to the large increase in representational complexity for the question-answer relationships offered by the additional trainable weight vectors and RNN networks layers.

For the baseline and BiDAF models without the additional Model layer and changes to the Output layer, the predictions made often placed the ending pointer before the starting pointer, giving the

empty string as a degenerate answer that was guaranteed to be wrong. It wasn't until we added the Model layer and introduced the attention dependency between the start and end pointer that these errors mostly disappeared, explaining the large gap between the plateaus of the baseline and BiDAF-only models and the BiDAF + Model and Output Layer models for both train and dev loss.

However, our final model still commits errors when the predicted answer to the question matches the "type" of answer that the question is looking for, but due to the large presence of other similar answers, the model focuses on the wrong area of the context, giving us an answer that is reasonable but still ultimately wrong. This can be seen for the question "Which architect, famous for designing london's st. paul cathedral, is represented in the riba collection?" The true answer is Sir Christopher Wren, but the model predicts Andrea Palladio, another artist that the context explicitly mentions is present in the riba collection. Because of this ambiguity, in order to resolve these types of errors, we would need to implement some semblance of an Answer Pointer system so that our model could more accurately pinpoint which possible answer out of the ones given would fit the question, rather than just giving just some answer span for context pieces that seem answer-like.

5 Conclusion and Future Work

Performing these experiments has shown me the importance and power of complex attention models and additional RNN modeling layers, as it is not enough to merely process large amounts of data in order to accurately answer a question; a good model must also be able to pinpoint which parts of the context and question are important and which ones are irrelevant so as to provide a more precise and complete answer. Moreover, thanks to the numerous Out-Of-Memory errors that I experienced and resolved over the course of building this model, I learned that it is very important to understand the data one is applying a model towards so that better decisions and preliminary performance improvements can be made and that unnecessary trade-offs and model complexities can be avoided. Finally, while some errors are unavoidable, it is still important to analyze the mistakes the models make to see if certain classes of error could be more easily fixed than others, in essence concentrating more productive time towards fixing mistakes that give significantly greater improvements to our model

In the future, to resolve the errors involving the model focusing on the wrong context areas for answering the question, we would seek to implement the Answer Pointer system in our Output Layer to help condition the ending pointer prediction more accurately based on the starting pointer prediction, rather than just simply taking an argmax. This could give us a more useful probability distribution of locations over which these boundary pointers could point in the context, increasing the range of complexity that our model could express and possibly answering even more advanced questions with long-term dependencies [2].

Acknowledgments

We would like to thank the CS 224N professor and TAs for creating such an interesting, exciting class and wonderful baseline structures to help get us started on the default final project. We would also like to thank all our peers with who with discussed ideas to help point out flaws and possible improvements that could be made. Our project would not exist in the state it is here today without the support and kindness of the staff and students.

References

[1] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, Bidirectional Attention Flow for Machine Comprehension, ArXiv e-prints, Nov. 2016.

[2] http://web.stanford.edu/class/cs224n/default_project/default_project_v2.pdf