Machine Comprehension using Deep Learning

Henry Lin[1] and Sharman Tan[2]

[1]*Department of Symbolic Systems, Stanford University*
[2]*Department of Computer Science, Stanford University*
{henryln1, sharmant}@stanford.edu

March 25, 2019

Abstract

Machine comprehension is one of the most interesting tasks for Natural Language Processing (NLP) and the field of machine learning overall. Many of the recently successful methods involve the use of word embeddings and different variations of Recurrent Neural Networks (RNNs). In this paper, we explore the performance of a Bi-Directional Attention Flow (BiDAF) network alongside a bi-directional LSTM encoder and character-level embeddings. We then evaluate our approach on the Stanford Question Answering Dataset (SQuAD), and we achieve a F1 score of 74.862 and an EM score of 64.230 on the test set.

1 Introduction

Machine comprehension is one of the many subfields of NLP that has gained substantial traction in recent years as deep learning continues to grow in popularity. The problem involves a machine or model that is given a paragraph of information, called the context, and a query as input, and the machine works to extract the correct answer to the question from the context paragraph. The most recent and successful deep learning approaches have involved leveraging advanced forms of RNNs as well as pretrained word embeddings to achieve high accuracy on SQuAD. Often times, models use an attention mechanism that allows the program to hone in on a small, relevant part of the context to focus on the interaction between the context and query.

In this paper, we detail the bi-directional attention flow (BiDAF) implementation alongside several other improvements and modifications that we implemented to improve our model's performance. While they varied in impact, some of our modifications include learning character-level embeddings, conditioning the end location probability distribution on the start location probability distribution, and implementing bi-directional GRU vs. LSTM layers.

2 Related Work

Since the release of SQuAD with a publicly available leaderboard, there have been significant advancements in the field of machine comprehension in recent years [1]. The most successful models have approached human level performance on the dataset, and there are several common themes in the approaches spearheaded by different teams. Many of the top performing programs utilize some type of attention mechanism within their machine to help model the interactions between the queries and contexts [2].

A traditional encoder-decoder RNN model faces a bottleneck in the last layer of the encoder, since that single layer is responsible for holding the entire input's information. Attention layers bypass that bottleneck by placing emphasis on different parts of the inputs to better capture information. The success of attention layers in many NLP tasks has led to it becoming a key component of many state-of-the-art models.

One of the most successful attention models is Bi-Directional Attention Flow (BiDAF) [2], and it is the cornerstone to the machine comprehension model that we implement in this paper.

3 Task Definition

Given a context paragraph and a question, our model's task is to output the answer to the question by signifying the start and end positions of the answer, which are contained within the context. Formally, the context is represented as $C = c_1, .., c_N$ and the question as $Q = q_1, ..., q_M$. The model then outputs p_{start}, $p_{end} \in C$, representing the probability distributions of the pos-

sible start and end locations of the answer to the question.

4 Model

4.1 Bidirectional GRU (Baseline)

The baseline model is a bidirectional GRU containing an RNN encoder layer that encodes the context and the question into hidden states. An attention layer combines the context and question representations, and the output layer applies a fully connected layer and two separate softmax layers to compute the start and end locations of the answer span.

4.2 Simple BiDAF

Our first improvement to the baseline implementation is what we call a Simple BiDAF layer, a simplified version of the BiDAF layer described in the paper [2]. Rather than computing the described similarity matrix, which is computationally expensive to compute, we perform a simple matrix multiplication between the context and question embedding matrices and use the resulting matrix $S' \in R^{N \times M}$ as our similarity matrix (where N is context length and M is question length). This computation is as follows.

$$S = CQ^T$$

$C \in R^{N \times b}$ is the matrix that holds a context hidden state in each of its columns, $Q \in R^{M \times b}$ is the matrix that holds a question hidden state in each of its columns, and b is the batch size. Once we have computed S, we follow the remainder of the implementation details for the more complex BiDAF layer (described in the following section). Intuitively, although we compute S through a simple matrix multiplication unlike the more complex BiDAF implementation [2] , we are still encoding information about each c_i and q_j in S_{ij}.

In our final implementation, however, we replace our Simple BiDAF layer with the more complex BiDAF layer detailed in the paper [2] to maximize our F1 and EM scores.

4.3 Bidirectional LSTM with BiDAF

Figure 1 displays the general architecture of our final model, which consists of character, word, and context embedding layers as well as a BiDAF layer, a modeling layer, and an output layer.

4.3.1 Word and Character Embedding Layers

To better account for the morphology of words and to better handle words outside of our vocabulary, we implement a character-level embedding layer much like the layer described in the paper [2]. We implement a single-layer character-level CNN that trains the character-level embeddings of all 100 printable string characters in Python, which includes all uppercase and lowercase characters of the alphabet as well as all digits and commonly used punctuation. We map each input word to a vector space using the CNN. More specifically, we take our trainable character-level encodings $e_1, ..., e_{100} \in R^{d_c}$ and we compute hidden representations $h_1, ..., h_{100} \in R^f$ from windows of characters $[c_{i-k}, ..., c_i, ..., c_{i+k}]$ centered at position i and with window size k. Then, we apply element-wise max pooling on the hidden states to obtain the character-level encoding of each word. Figure 2 visualizes the character-level embeddings we learn for each of the 100 characters.

The word embeddings are fixed, pretrained 100-dimensional GloVE vectors, and we concatenate the character-level and word-level embeddings of each word to obtain the inputs to our contextual embedding layer.

4.3.2 Contextual Embedding Layer

The contextual embedding layer is a 1-layer bidirectional LSTM that accounts for the temporal relationships between words. The LSTM takes our concatenated embeddings as input and outputs our context hidden states $c_1, .., c_N \in R^{2h}$ and our question hidden states $q_1, ..., q_M \in R^{2h}$ respectively. N is the number of words in each context, M is the number of words in each question, and h is the hidden vector size. We pad or truncate the contexts and questions to be N and M words long, respectively.

4.3.3 Bidirectional Attention Flow Layer

For the attention flow layer, we implement a the BiDAF layer described in the paper [2], which allows attention to flow from contexts to questions as well as from questions to contexts. The context to question (C2Q) attention informs us of the query words that are most strongly related to the context words, while the question to context (Q2C) attention informs us of the context words that are most similar to a query word. This implementation of BiDAF differs from our Simple BiDAF implementation in that we define the similarity matrix $S \in R^{N \times M}$ by finding the

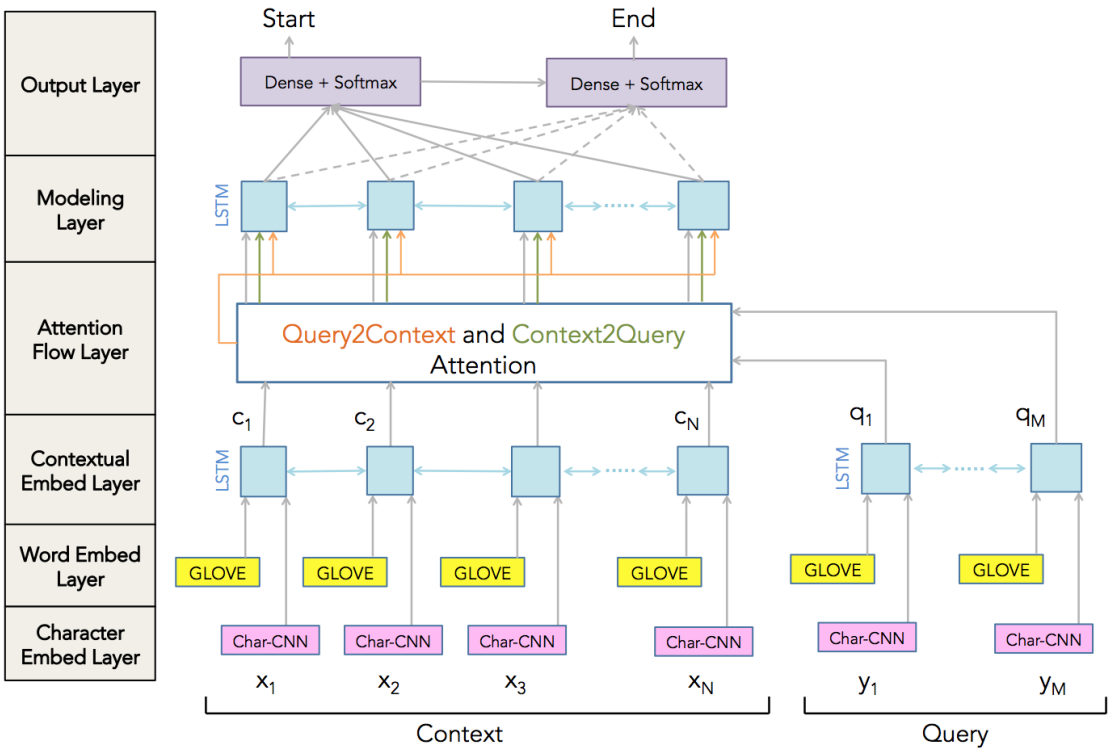


Figure 1: Visualization of Model Architecture [2]

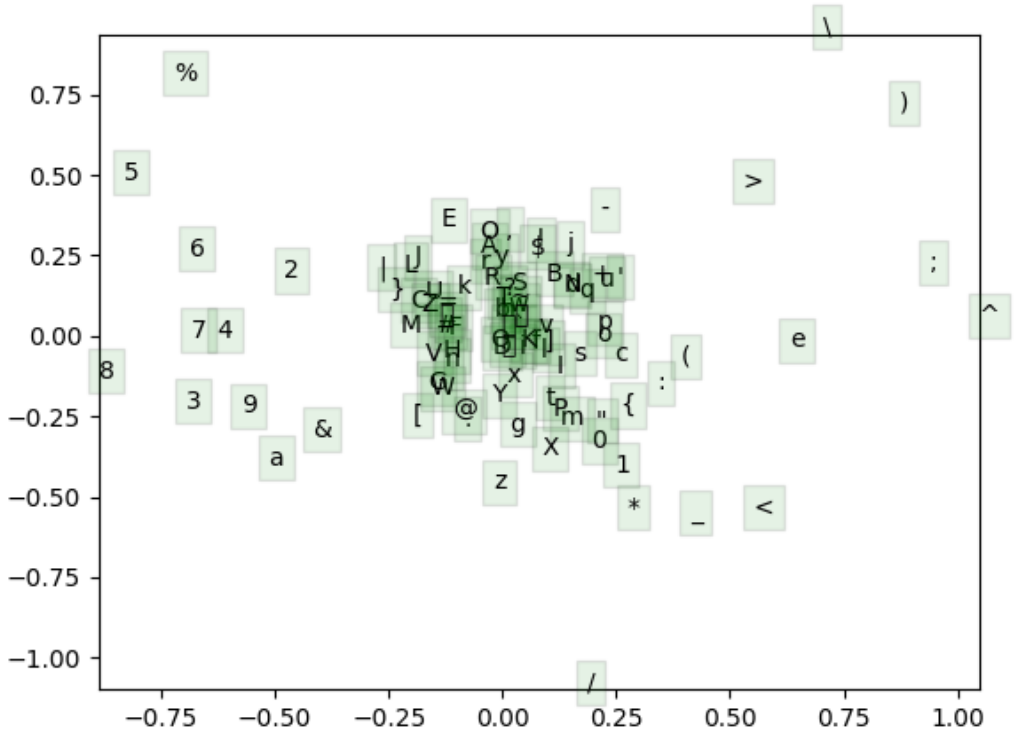


Figure 2: Visualization of Trained Character Embeddings

similarity score S_{ij} for each pair (c_i, q_j) of context and question hidden states [2] as follows.

$$S_{ij} = w_{sim}^T[c_i; q_j; c_i \circ q_j] \in R$$

In the equation for S_{ij} above, $w_{sim} \in R^{6h}$ is a weight vector. We then use S to compute C2Q attention by taking a row-wise softmax of S to obtain attention distributions that we use to take weighted sums of the question hidden states q_j to get the C2Q attention outputs a_i.

We compute Q2C attention outputs by taking the max of each row of S corresponding to each of the N context locations to get a vector $m \in R^N$, which is the attention distribution β over the context hidden states. Then, use β to take a weighted sum over the context hidden states to get the Q2C attention output c'. Overall, the BiDAF layer outputs a vector $b_i \in R^{8h}$ that is the concatenation of the context hidden state c_i and the attention outputs a_i and c' from the C2Q and Q2C attention layers respectively for all $i \in 1, ..., N$.

4.3.4 Modeling Layer (M)

The modeling layer is a 1-layer bi-directional LSTM that takes the outputs of the BiDAF layer as input and outputs a matrix $B \in R^{N \times 2h}$ which is then used in the output layer. The purpose of the modeling layer is to help capture information about each context word in relationship to not only the rest of the context paragraph, but also the query.

4.3.5 Output Layer - Conditioning End on Start (CES)

Our implementation of CES is inspired by the paper [3], but we do not implement the 2-layer RNN model suggested in the paper.

First, we concatenate the matrix B from the modeling layer to the hidden states of the context words and subsequently pass it through a fully connected layer with ReLU activation. The resulting matrix $B' \in R^{N \times 2h}$ is then used to calculate the probability distributions for the answer's start and end positions. We do this by first calculating the start position's probability distribution through a simple softmax layer, as follows.

$$p_{start} = softmax(W_{start}B' + b_{start}) \in R^N$$

Then, we condition the probability distribution for the end location on the start location probability distribution. In theory, this may lead to some improvement when calculating the answer, since the start location distribution actually does have a direct impact on the end distribution (e.g. high probability answer starts later in context means high probability answer also ends very late in the context).

To condition the end location's probability distribution on that of the start location, we first find a_{start}, the attention output for the start position. To do this, we compute the weighted sum of the entries in B' corresponding to each of the N context locations, using p_{start} to define the weights, as follows.

$$a_{start} = p_{start_1}B'_1 + ... + p_{start_N}B'_N$$

Then, we append a_{start} to each of the entries in B' corresponding to each context location, and we pass this modified B' (call it B'') through a simple softmax layer to get the end distribution p_{end}, as follows.

$$B'' = \{[b'_1; a_{start}], ..., [b'_N; a_{start}]\}$$
$$p_{end} = softmax(W_{end}B'' + b_{end}) \in R^N$$

Because a_{start} incorporates the start probability distribution and B'' that we use to predict the end position incorporates a_{start}, we are conditioning the probability distribution for the end location on the start location probability distribution.

4.3.6 Determining Start and End with Smart Span Selection

Using start and end distributions p_{start} and p_{end} over the context paragraph, we calculate a joint probability matrix to determine the most likely start and end for the answer. Both probability distributions have dimension N, so we compute the joint probability matrix by multiplying the two together to form a matrix with dimensions $N \times N$. We then zero out the triangle of values below the main diagonal, because the end's word position cannot be before the start word's position. With the resulting matrix, Then, using a simple $argmax$ function on the resulting matrix, we find the most probable start word and end word locations.

5 Experiments

5.1 Dataset

We use the Stanford Question Answering Dataset (SQuAD) to train, validate, and test our model. SQuAD is a machine comprehension dataset drawn from a large number of Wikipedia articles. For each example in the dataset, there is a context paragraph, a query, and possibly multiple human responses to the query drawn from the context. The dataset has over 100,000 examples and is a very robust dataset to evaluate our model's performance on.

5.2 Simple BiDAF

From implementing the Simple BiDAF layer, we observe an increase of 13.81 in our F1 score and an increase of 11.949 in our EM score on the dev set. Although our Simple BiDAF implementation is much simpler than the more complex BiDAF layer implementation [2], the Simple BiDAF layer still significantly increases our F1 and EM scores from the baseline implementation. This may be because even the simple matrix multiplication of the context and question embeddings computes dot products that encode information from both the contexts and the questions.

5.3 Character CNN and Bi-directional LSTM

Following the Simple BiDAF layer, we implement a single-layer character-level CNN to learn character embeddings, and we also convert the baseline's bi-directional GRU contextual embedding layer to a bi-directional LSTM layer. This results in an increase of 3.162 in our F1 score and an increase of 3.482 in our EM score. This improvement may be due to the additional morphological information that our learned character embeddings encode, as well as the higher level of complexity of the bi-directional LSTM model compared to the bi-directional GRU model.

5.4 Multilayer Character CNN

We also attempt to learn the character embeddings using a multilayer CNN instead of a single-layer CNN. The multilayer CNN did not perform any better than the single layer and was significantly more computationally expensive, so we use the single-layer CNN for our final model. We tried sigmoid and ReLU activation functions in addition to a shrinking hidden size for each layer of the multilayer CNN, but in the end, the single-layer CNN more efficient and produced more accurate results.

5.5 Conditioning End Distribution on Start Distribution (CES)

Next, we condition the end location distribution on the start location distribution, replacing the baseline's implementation that determines each of the distributions independently. This addition did not noticeably change our F1 and EM scores on the dev set. In fact, it slightly decreased our F1 score by 0.952 and our EM score by 1.012. This slight decrease may be attributed to randomness. Because our implementation of CES involves a simple fully connected layer and may not significantly change the end location distribution, we are not very surprised about these results.

5.6 BiDAF

The computational efficiency of our Simple BiDAF layer allowed us to quickly outperform the baseline implementation, making way for our conversion from a bi-directional GRU to a bi-directional LSTM and our addition of character-level embeddings. These additions improved our F1 and EM scores on the dev set from 43.169 and 34.134 (baseline) to 60.141 and 49.565, respectively. At this point, we decided to replace our Simple BiDAF implementation with the more complex BiDAF implementation detailed in the paper [2].

5.7 Smart Span Selection

The final step that further improved our model was smart span selection using a joint probability matrix. This addition did not require any more training since the model had already been trained – it simply worked with the calculated probabilities differently. By having the output start and end depend on the joint probability, we raised both the F1 and EM score by a noticeable amount. Our F1 and EM scores on the dev set increased by 2.698 and 2.204, respectively, and our F1 and EM scores on the test set increased by 2.556 and 2.203, respectively. These results confirm our hypothesis that utilizing a joint probability matrix instead of independent start and end probabilities results in more accurate answers in many cases.

5.8 Simple BiDAF with Modeling Layer (M)

Our final model (BiDAF/Bi-LSTM/Char CNN/CES/Modeling/Smart Span) produced our highest F1 and EM dev scores of 74.649 and 63.765, but we did notice that our BiDAF model without an LSTM modeling layer resulted in lower scores than our Simple BiDAF model without the modeling layer. Therefore, we added the LSTM modeling layer to our Simple BiDAF model (which included the Bi-LSTM contextual embedding layer, character CNN, CES, and smart span selection) to see how the Simple BiDAF with the modeling layer would compare to the more complex BiDAF with the modeling layer.

The learning curves for dev loss, F1 score, and EM score for the Simple BiDAF and the

more complex BiDAF models are displayed in Figure 3. Surprisingly, for the first several thousand iterations of training, Simple BiDAF's F1 and EM scores were much higher than the more complex BiDAF model's, and Simple BiDAF's loss was also much lower than the more complex BiDAF model's. However, after approximately 8,000 iterations, Simple BiDAF's F1 and EM scores and loss converged to around those of the more complex model. Ultimately, Simple BiDAF produced dev F1 and EM scores of 73.979 and 63.160, respectively, just 0.670 and 0.605 points below the dev scores of the more complex model. This indicates that the Simple BiDAF model performs very well on SQuAD, despite our notable simplification of the similarity matrix in our Simple BiDAF layer implementation.

5.9 Hyperparameters

We conducted a hyperparameter search over learning rate, embedding size, and batch size to find the optimal hyperparameter settings for our model. Ultimately, we settled upon 100-dimensional GloVe vectors, a learning rate of 0.0007, and a batch size of 35. We found that other GloVe embedding sizes did not increase the performance of our model, and increasing learning rate by too much often resulted in the model diverging. Using a grid-style search, we may have discovered different hyperparameters that optimize our model's performance.

6 Error Analysis

6.1 Deciding Precise Boundaries

The text in blue is the predicted answer, the text in red is the true answer, and the text in purple is the overlap between the predicted and true answers.

Context: *"... von _lettow_ conducted an effective guerrilla warfare campaign, living off the land , capturing british supplies , and remaining undefeated ..."*
Question: *how did von lettow conduct his group ?*
Predicted Answer: *an effective guerrilla warfare campaign*
True Answer: *effective guerrilla warfare campaign , living off the land , capturing british supplies , and remaining undefeated*

As the above example exemplifies, our model has some difficulty determining precise start and end locations for the answer – when an answer is long or divided by commas, our model may not predict the exact section that constitutes the answer. Decisions involving answer start and end locations are especially difficult when longer answers are expected. This is especially true of "Why" questions, as their answers usually involve providing reasons and explanations of actions. Therefore, our model has more difficulty correctly answering "Why" questions and questions that expect long answers in general.

6.2 Multiple Correct Answers

The text in blue is the predicted answer and the text in red is the true answer.

Context: *"the immune system is a system of many biological structures and processes within an organism that protects against disease . to function properly , an immune system must detect a wide variety of agents , known as pathogens , from viruses to parasitic worms ..."*
Question: *what does the immune system protect against ?*
Predicted Answer: *disease*
True Answer: *a wide variety of agents , known as pathogens , from viruses to parasitic worms*

From the context and the wording of the question, "disease" seems to be an obvious answer; therefore, it is not surprising that our model predicts "disease" as its answer. However, the true answer comes from a more descriptive section of the context given. In such situations in which multiple answers are possible, our model's answer may be considered incorrect even though it is actually a valid answer to the question.

6.3 Character Embeddings

Shown in Figure 2 is a visualization of the trained character embeddings used in the model. We used an embedding size of 20 that was then trained using a single layer CNN with a hidden size of 100. As the visualization shows, the character embeddings do cluster together with similar characters after training the model. The clearest evidence of this is when looking at the center where all the letters have congregated. The next clustering occurs with the numbers that seem to dot the left side of the graph. There is not as much differentiation between the letters as we thought there would be, and this may be a contributing factor to why the character CNN feature did not improve performance significantly. This may be due to the size of the training data or the embedding size; in a future model, it would be interesting to test pre-

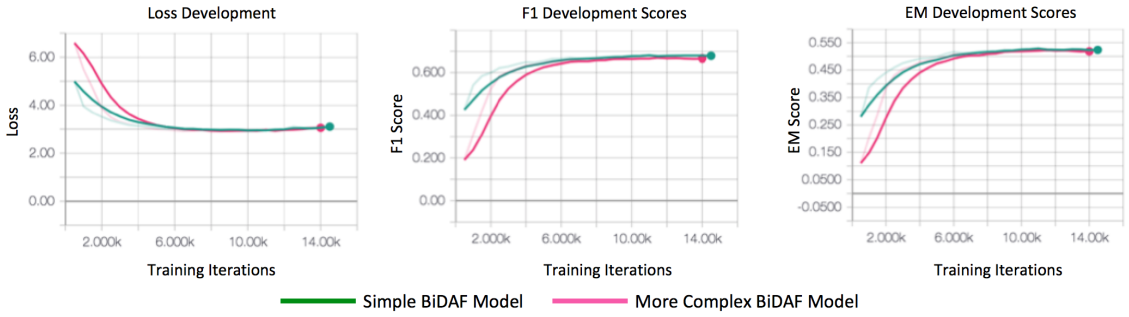
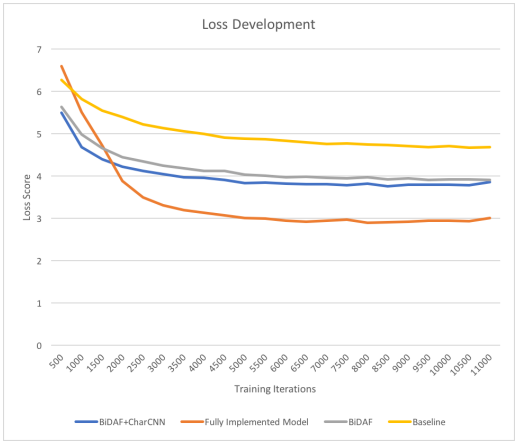


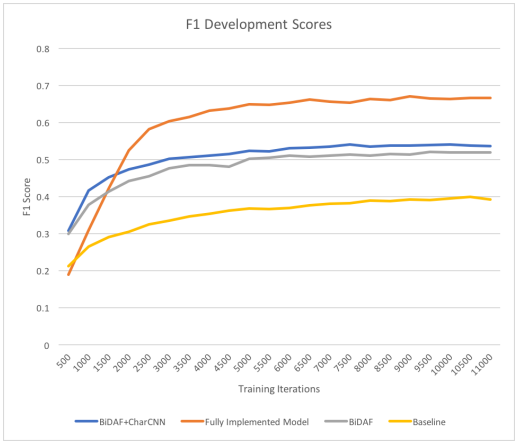
Figure 3: Simple BiDAF with Modeling Layer v. More Complex BiDAF with Modeling Layer

Experiment	Dev F1	Dev EM	Test F1	Test EM
Baseline	43.169	34.134	-	-
Simple BiDAF/Bi-GRU	56.979	46.083	-	-
Simple BiDAF/Bi-LSTM/Char CNN	60.141	49.565	-	-
Simple BiDAF/Bi-LSTM/Char CNN/CES	59.189	48.553	-	-
BiDAF/Bi-LSTM/Char CNN/CES/M	71.951	61.561	72.306	62.027
BiDAF/Bi-LSTM/Char CNN/CES/M/Smart Span	**74.649**	**63.765**	**74.862**	**64.230**
Simple BiDAF/Bi-LSTM/Char CNN/CES/M/Smart Span	73.979	63.160	-	-

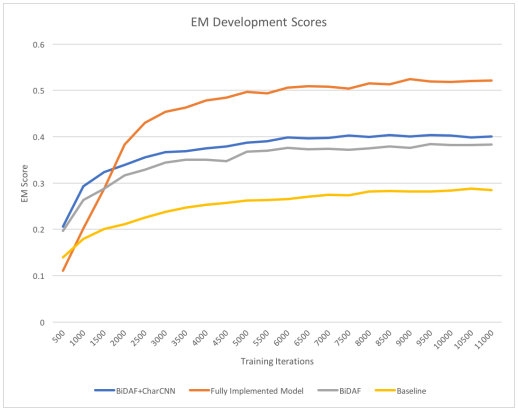
Figure 4: F1 and EM Scores by Experiment



(a) Loss



(b) F1 Scores



(c) EM Scores

Figure 5: Performance on Development Set

trained character embeddings or a larger embedding size.

7 Conclusion and Future Work

In this paper we explored the capabilities of the BiDAF model applied to the SQuAD dataset and we were able to achieve strong results using our implementation. The outcomes of the experiments highlight different improvements made by multiple model adjustments and modifications, and this can be be continued in the future. The biggest jump in our model's performance came from the BiDAF model and the subsequent modeling layer, so we are interested in investigating the performance of a model integrating the BiDAF layer along with other attention layers.

Our model containing our Simple BiDAF layer and the following modeling layer produced final scores just under those of our final implementation despite our significant simplification of the BiDAF implementation. Therefore, we are also interested in investigating the performance of the Simple BiDAF model further to find alternative implementations of attention layers that may be less computationally expensive but produce results equal or superior to the more complex BiDAF model. Furthermore, modifications to the final stages of predicting the start and end locations clearly resulted in noticeable improvements in our model's F1 and EM scores, so we would like to explore these areas more as well.

8 Acknowledgements

We would like to thank Abigail See for providing the original Github repository with the baseline implementation and evaluation metrics and the handout that contained concise and useful summaries of relevant papers of interest. We would also like to thank the TAs for providing support in debugging code and suggestions for different modifications to attempt.

References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016.

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

[3] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*, 2016.