CS224N Project Final Report
Bidirectional Attention Flow Model for Reading Comprehension

Bardia Beigi {bardia}, Soroosh Hemmati {shemmati}, Michael Painter {mp703}

Codalab submission username: bardia

Contributions of all members were equal in the project

Mar. 22, 2017

1 Introduction

Machine Reading Comprehension (MRC), which entails answering a query based on a context paragraph requires complex modeling and implementation. The field is growing to be an important one, as machines are becoming more responsive to humans' contextualized query either through speech as is the case with Amazon Alexa, Google Home, etc, or text with smart answers in Google Inbox or smart replies of chatbots. With the recent advances in deep learning frameworks, processing power, and Natural Language Processing research, the field has been ready for innovation in the area of MRC.

To tackle such a complex problem, there has been new modeling breakthroughs such as attention mechanisms. Attention mechanisms focus the model in looking for answers in a targeted area of the context paragraph.

This project employs the latest techniques in attention modeling. Even though a baseline with a bilinear attention model was implemented first, it was soon realized that the performance of the model would benefit from more complex attention modeling. Thus, a bidirectional attention flow model was implemented[1] in addition to more thorough and complex modeling of the word representations followed by a more careful output representation. The result of the implemented models will be discussed later in the paper.

2 Notation and Building Blocks

We begin by defining the notation and building blocks that we use to define the different models that we have built over the project. This mirrors the code in structure, where pieces were built modularly and pieced together as needed. For conciseness, we will let the diagrams do most of the talking.

2.1 Notation

Throughout this report we will commonly use matrices such as $X \in \mathbb{R}^{d\times T}$ to represent sequences of words. Where each each column vector $X_{:t} \in \mathbb{R}^d$ is a dense vector representation for a word.

$$X = \begin{bmatrix} | & | & & | \\ X_{:1} & X_{:2} & \cdots & X_{:T} \\ | & | & & | \end{bmatrix}.$$

2.2 Basic Blocks

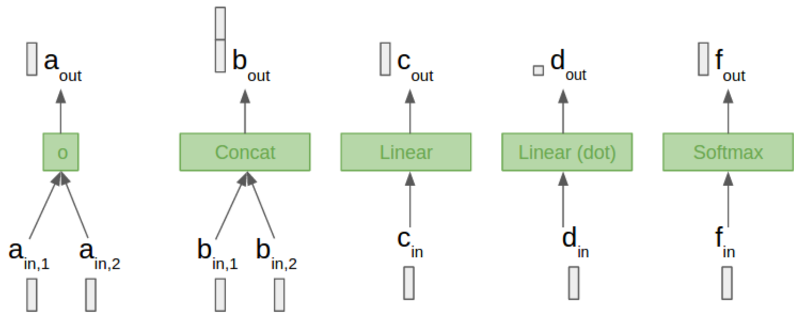


Figure 1: Some of the basic building blocks that we will use. Respectively, Element-Wise Multiplication, Concatenation, Linear, Linear (Dot), Softmax.

These blocks are used to implicitly define the following equations of the following respective forms later in the report,

$$a_{out} = a_{in,1} \circ a_{in,2}, \qquad b_{out} = [b_{in,1}; b_{in,2}], \qquad c_{out} = W^{(c)} c_{in} + b^{(c)},$$
$$d_{out} = {w^{(d)}}^T d_{in}, \qquad f_{out} = \text{softmax}(f_{in}).$$

where the inputs and outputs are to be replaced by the appropriate variables in the model. If the dimensions of the inputs are $a_{in,1}, a_{in,2}, b_{in,1}, ..., f_{in} \in \mathbb{R}^d$, and $W^{(c)} \in \mathbb{R}^{d' \times d}$, $b^{(c)} \in \mathbb{R}^{d'}$, $w^{(e)} \in \mathbb{R}^d$ are trainable variables. The outputs would then be

$$a_{out} \in \mathbb{R}^{2d}, \qquad b_{out} \in \mathbb{R}^d \qquad c_{out} \in \mathbb{R}^{d'} \qquad e_{out} \in \mathbb{R} \qquad f_{out} \in \mathbb{R}^d.$$

2.3 BiLinear attention

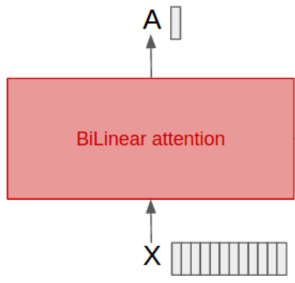


Figure 2: The BiLinear attention block, taking in a vector time sequence and a vector to attend to. A 'context' vector is output c (note this should not be confused with 'context paragraphs' refereed to elsewhere in this report).T hopefully encapsulating which of the vectors in the time sequence X are relevant to the attending vector v.

We define the time sequence $A \in \mathbb{R}^d$ in terms of the inputs $X \in \mathbb{R}^{d \times T}$ and $v \in \mathbb{R}^{d'}$,

$$a_i = X_{:i}^T W v, \qquad s = \text{softmax}(a), \qquad c = \sum_{t=1}^{T} a_t X_{:t}$$

where $W \in \mathbb{R}^{d \times d'}$ is again a trainable matrix.

2.4 LSTM networks

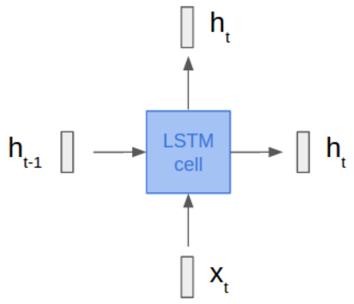


Figure 3: Our outside view of the LSTM cell, which we will use as our fundamental unit for more complex recurrent structures.

For the purpose of this report we will build up from the LSTM cell, considering only the equations such as

$$h_t = \mathrm{LSTM}(x_t, h_{t-1}),$$

where a more thorough treatment of LSTMs can be found on Christopher Olah's blog [2]. Further, for the entirety of this section we will us $d \times T$ as the size of the input to an LSTM network, and we will take d' to be the state size of the LSTM(s), thus the output will be of size $d' \times T$.

2.5 LSTM networks

To use LSTMs we create a rolled out network, where the (internal) weights are shared between each of the cells.

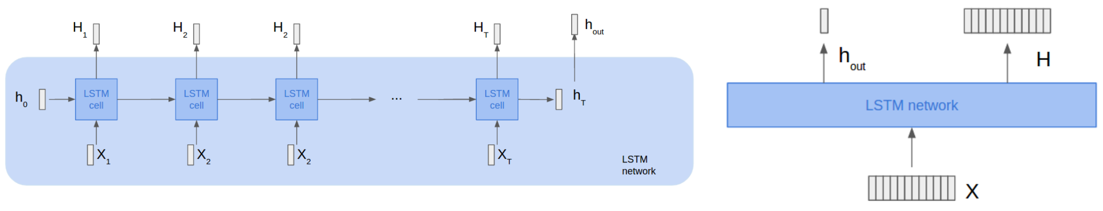


Figure 4: Left, a rolled out LSTM network, one layer deep, and right, a block used to represent this network.

In the network from figure 4 we have one input and two outputs, which are $X \in \mathbb{R}^{d \times T}$, $H \in \mathbb{R}^{d' \times T}$ and $h_{out} \in \mathbb{R}^{d'}$ respectively.

The equations for the network are

$$H_{:0} = 0, \qquad H_{:t} = \mathrm{LSTM}(X_{:t}, H_{:(t-1)}), \qquad h_{out} = H_{:T},$$

noting that $H_{:0}$ is separate from H. We could set $H_{:0}$ to something non-zero, however in all of the models we consider in this report every LSTM has it's state initially set to 0.

2.6 BiDirectional LSTM networks

In a similar fashion to section 2.4 we can define a recurrent network that can at each step utilize information from the future and the past, not only the past.

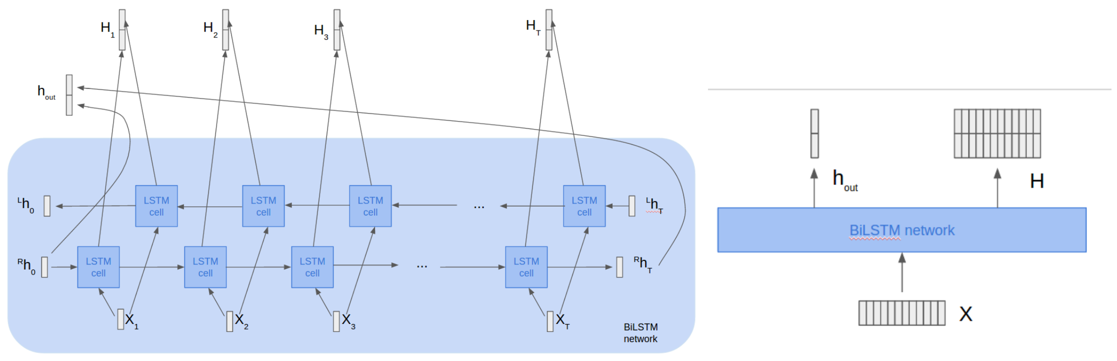


Figure 5: Left, a rolled out BiLSTM network, one layer deep, and right, a block used to represent this network.

Again, we consider a rolled out network in figure 5, however, noticeably we now have that $H \in \mathbb{R}^{2d' \times T}$ and $h_{out} \in \mathbb{R}^{2d'}$, as we concatenate the outputs from the forward and backward networks. In figure 5 we again reduce it to a single block.

The equations for this network are a similar to those in section 2.4.

$$h_0^R = 0, \qquad h_t^R = \text{LSTM}(X_{:t}, h_{t-1}^R), \qquad h_T^L = 0,$$
$$h_t^L = \text{LSTM}(X_{:t}, h_{t+1}^L), \qquad H_{:t} = [h_t^R; h_t^L], \qquad h_{out} = [h_T^R; h_1^L]$$

2.7 Deep (Bi)LSTM networks

Finally, we can chain the rolled out LSTM networks defined in sections 2.4 and 2.6 to create deep recurrent networks. Writing $h, H = \text{LSTM}(X)$ as shorthand for the network in figure 4 and it's corresponding equations. We can then define a deep network of depth D with the following equations

$$H_0 = X, \qquad h_i, H_i = \text{LSTM}(H_{i-1}), \qquad H_{out} = H_D, \qquad h_{out} = [h_1; ...; h_d].$$

In exactly the same fashion we construct a deep BiLSTM network, simply replacing the 'LSTM' by a 'BiLSTM'.

3 Baseline Model

In its simplest form, the baseline model we used in this project can be broken down as follows:

- Word Embedding Layer: maps each word to a vector space using pretrained GloVe vectors.
- Question Encoder: BiLSTM over the question summarizes the meaning in the concatenation of the final states.
- Conditional Paragraph encoder: BiLSTM over the context paragraph conditioned on the question representation. This was implemented by feeding in a concatenation of the question summary vector and the current word vector into the LSTM at the current time step.
- Attention Vector: calculated over the context paragraph, based on the question representation. To be more specific, we have implemented a bilinear attention model in which we create an attention vector based on the context vectors, x_t, and question vector q, as $a_t = \text{softmax}(x_t^T W q)$ (see section 2.3, figure 6).
- Model Layer: New context vectors for each word in the paragraph based on attention,
- Output Layer: Final LSTM to create two probability vectors, representing the likelihood of each word being the start or end of an answer.

We then take the argmax of the final probability vectors to find the start and end indices of the answer. The loss is calculated over each minibatch and minimized using an Adam optimizer.
We further enhanced the baseline model using the following extensions.

- Enabling backpropagation through embeddings,
- Deep encoders: creating multiple layers of BiLSTM networks each of which acts as input to the next layer for encoding question and context paragraphs representations,
- Deep decoders: creating multiple layers of LSTM's to further enhance the attention-combined representations of context words used to find answer probabilities.

3.1 Implementation Details

These details were implemented for all models described in this section and section 4:
To avoid overfitting, gradient clipping was enabled and the model was trained using a non-zero dropout rate. In addition, all inputs (questions or context paragraphs) with lengths lower than their corresponding maximum were zero-padded and those with longer lengths were clipped at the maximum allowed length. In addition, in cases where the true answer resided in the clipped section of a paragraph, we would feed the model -1 to imply that an answer does not exist.

4 BiDirectional Attention Flow (BiDAF)

To overcome the limitations of the baseline model, a BiDAF architecture was implemented. Here is a high level description of the BiDAF model [1]:

- Word Embedding Layer: maps each word to a vector space using pretrained GloVe vectors, similar to the baseline model,
- Contextual Embedding Layer: creates new vector representations using BiLSTM's, quite similar to the baseline model without final summarization,
- Attention Flow Layer: couples contextual embeddings found in the previous step to produce attention vectors for each word in the context,
- Modeling Layer: employs an RNN to scan the document, given the attention vectors,
- Output Layer: provides answers to the question by creating two probability vectors, representing the probability of a context word being the start or end words in the answer.

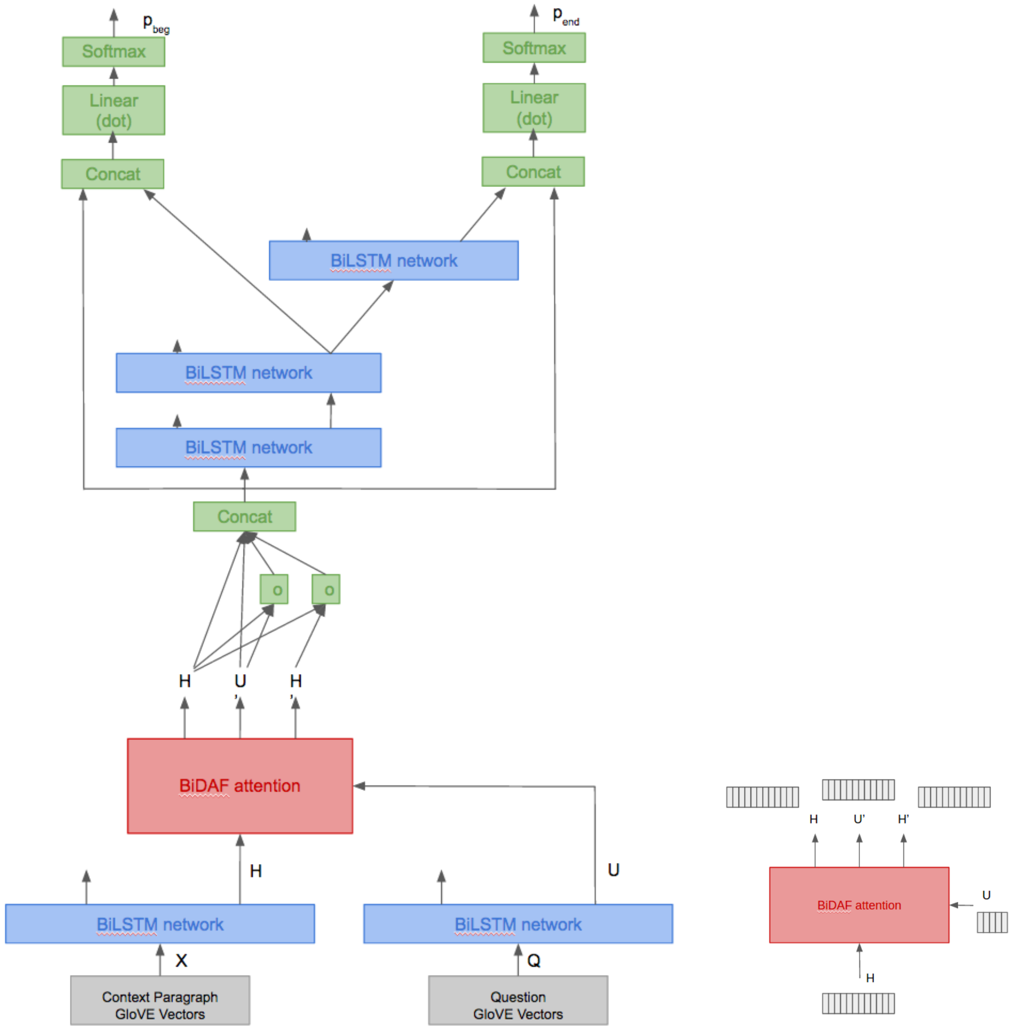


Figure 6: Overview of the BiDAF model, using the blocks defined in section 2. On the right is a visualization of the BiDAF unit, which is defined more precisely below. Unused outputs from any blocks are indicated with 'arrow stumps'.

Here is the BiDAF model in more mathematical detail:

- Word Embedding Layer: matrices P and Q are created, where $P \in \mathbb{R}^{d \times T}$ and $Q \in \mathbb{R}^{d \times J}$, where T is the length of the paragraph vector (cut off at a certain point to equalize the lengths of paragraphs), J is the length of the question (also cut off), and d is the embedding size.
- Contextual Embedding Layer: Here, we obtain the context word vectors H and U by running P and Q through BiLSTMs. Therefore, we have $H \in \mathbb{R}^{2d \times T}$ and $U \in \mathbb{R}^{2d \times J}$.
- Attention Flow Layer: it is due to this layer that the name BiDirectional appears in the description of this model. Once context and question vectors are properly created, a similarity matrix is created which models the similarity of the context and question words: $S_{tj} = \alpha(H_{:t}, U_{:j}) = w_{(S)}^T[h; u; h\, o\, u]$, where $w_{(S)} \in \mathbb{R}^{6d}$ and o is the elementwise multiplication operation. Now, we will proceed to define the following attention matrices:
  - Context-to-query Attention: signifies which query words have the closest similarity to each context word. To establish this quantity, we will use S defined above to get $a_t = \text{softmax}(S_{t:}) \in \mathbb{R}^J$. Subsequently, we can define our context-to-query attention matrix, $U'_{:t} = \sum_j a_{tj} U_{:j} \in \mathbb{R}^{2d \times T}$.
  - Query-to-context Attention: signifies which context words are closest to each query word. Similar to context-to-query, $b = \text{softmax}(\max_{col}(S) \in \mathbb{R}^T$. Then, $h' \sum_t b_t H_{:t} \in \mathbb{R}^{2d}$. This vector is a weighted sum of the words in the context based on the query. H' is h' tiled T times across the column.

  The final result of this layer is G where $G_{:t} = \beta(H_{:t}, U'_{:t}, H'_{:t}) \in \mathbb{R}^{d_G}$ represents the query aware representation of the t^{th} word of the context paragraph and $\beta(h, u', h') = [h; u'; h\, o\, u'; h\, o\, h'] \in \mathbb{R}^{8d \times T}$.
- Modeling Layer: The input to the modeling layer is the matrix G which goes through a BiLSTM to create the matrix $M \in \mathbb{R}^{2d \times T}$. M captures the interactions of the context words with each other.
- Output Layer: To create the index of the final start index, we use a softmax layer on G and M as follows:

$$p^1 = \text{softmax}(w_{(p^1)}^T[G; M])$$

  where $w_{(p^1)}$ is a matrix to be learned. Similarly, to find p^2, M is passed through another layer of BiLSTM's to create M^2 and $p^2 = \text{softmax}(w_{(p^2)}^T[G; M^2])$.

Training: the loss function used is the cross entropy loss for the beginning and end predictions:

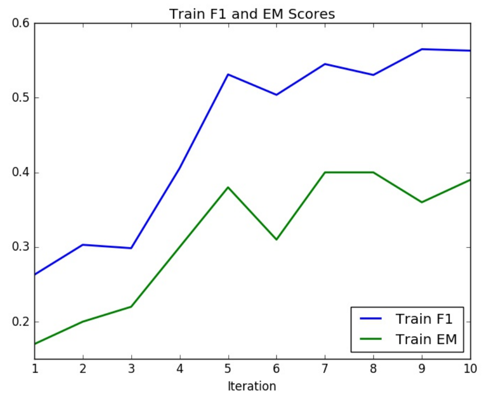
$$L(\theta) = -\frac{1}{N}\sum_i^N \log(p^1_{y^1_i}) + \log(p^2_{y^2_i}).$$

5 Results

Due to the challenges and errors we encountered (explained in the next section), our results from the baseline model and its extensions turned out to be obsolete. As a result, the results given here are all due to the BiDAF model.

5.1 F1 and EM Scores

Our highest F1 and EM scores on the train set are 56.4% and 40.0% respectively. These were achieved by assuming a max paragraph length of 300, max question length of 100, a hidden state size, d, of 50, a learning rate of 0.01, gradients clipped at 10, and using an Adam optimizer. The same model also gave the best results on the dev set, 58.6% F1 score and 46% EM score. It should be noted that these scores were all found on the GPU. Our highest submission F1 and EM scores, however, are 43.936% and 30.965% on Codalab's dev set as well as 45.054% and 32.739% on Codalab's test set, respectively. A lower submission score was



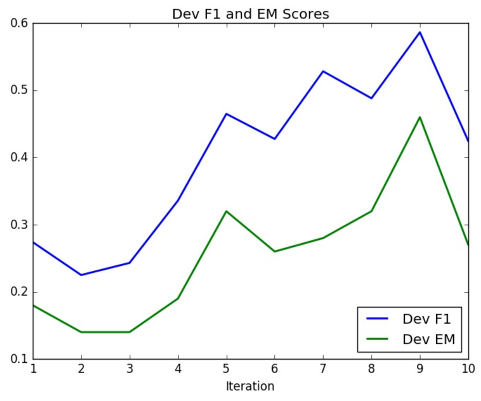


Figure 7: F1 and EM scores during training

expected since the scores calculated locally are all based on a sample size of 100.
We have tried tweaking the model parameters slightly to better the results with little to no tangible difference in outcome. Keeping all the dimensions the same, we changed the hidden vector size, d, to 100 hoping that the added complexity might enhance the results but it did not, with F1 scores of below 55% and EM scores of at most 42%.
Figure 7 shows the train and dev scores during training for the best results we got. Currently, we are running a model using AdaDelta optimizer with a learning rate of 0.02 which has not fully converged yet.

5.2 Loss

For the best model reported above, loss dropped rapidly and fairly consistently during the first 7 epochs. However, upon reaching epochs 8 - 10, train error began to plateau and validation error began to increase, pointing out the fact that the model had begun to overfit (see figure 8). One should not put too much weight on the behavior of the validation error since it is based only on 100 samples.

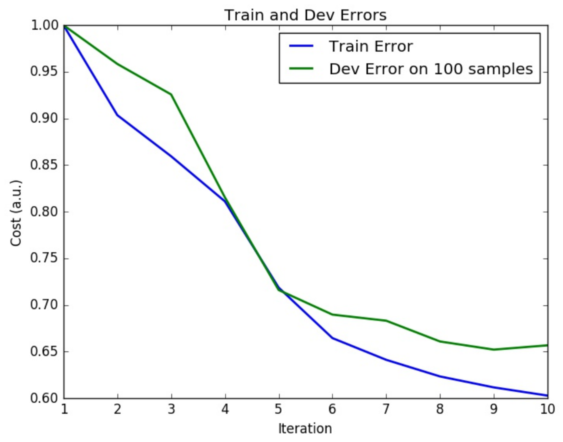


Figure 8: Training and Validation Error During Training

5.3 Analysis

The BiDAF model was a lot more successful in correctly answering questions, however, there exist certain pitfalls:

- The model works really well on getting fairly accurate results on shorter questions which do not have a grammatically complex structure. As the model does not perform dependency parsing to properly identify the meaning of the question and direct attention, having simple sentences really helps enhance prediction.
- Examples in which question and paragraph share a few key words seem to get much better answers. For instance, for context paragraph "Michael studies at Stanford" and questions "Where does Michael study?" or "Which university does Michael go to?", "Stanford" would have a much higher probability of being identified as the correct answer given the first question.
- There are examples in the data where the answer indices are given as -1, indicating that an answer does not exist in the paragraph. In cases like this, as the model always picks two indices within the paragraph as the start and end positions of the answer, it will definitely perform poorly.

6 Challenges

We faced numerous challenges on the path to finishing this project, which was a humbling experience. Beside the initial challenges of understanding the organization of the starter code, upon implementing the first baseline model, we had a major setback.

The initial baseline model repeatedly crashed due to a memory error on Azure. After some investigation, it turned out that our model simply had a few variables which took up more than 2GB of memory space. The solution was to decrease the state size of the LSTM cells used as well as to use a smaller mini-batch size for training. Additionally, by looking at the histograms of the quantity of context paragraphs as well questions vs. their respective length, we realized that almost all context paragraphs in the training set have a maximum length of 300 tokens, and almost all questions in the training set have a maximum length of 60 tokens. This finding enabled us to save us a lot of space limiting the size of the LSTM layers processing the context paragraphs and questions.

However, even after fixing that error, the baseline continued to perform poorly. Our effort to debug the problem was not effective until recently when we paid more attention to the variable scoping used in the project. The problem was that our variable scoping was not correctly used, in other words we would set up the Tensorflow nodes under a set of scope variables, but would not instantiate the model using the same variable scopes resulting in disjoint model training from evaluation. As a result, we took a careful look at the variable scopes and made sure we never left the scope while instantiating the model.

Other minor setbacks included inputting the remove probability instead of keep probability in Tensorflow's dropout function, as well as problems with the dimensions of the model throughout implementation.

7 Conclusion & Future Work

Using complex attention mechanisms to achieve a higher score seemed inevitable since the start of the project. Therefore, after implementing the baseline model with bilinear attention, we opted to use the more complex bidirectional attention flow model - BiDAF - which uses context-to-query and query-to-context attention modeling to emphasize the connection between certain query words and certain words in the context paragraph. The complete BiDAF model was relatively fast to train at slightly over an hour per epoch. The results from the BiDAF model largely improved on those of the baseline, and we were able to get a max test set F1 and EM accuracy of 45.054% and 32.739%, respectively.

These results could be improved upon by performing more parameter tuning such as learning rate, dropout rate, etc. It would also be really interesting to introduce Quasi Recurrent Neural Networks (QRNNs) to speed up the training time due to their parallel nature, as well as possibly improve the obtained F1 and EM scores. Finally, combining a dependency parser with the output of the model layer and running additional BiLSTM layers to train using that data would likely increase the accuracy of the model.

References

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

[2] Christopher Olah. Understanding lstm networks, 2015.