Start and End Interactions in Bidirectional Attention Flow for Reading Comprehension

Sean Rafferty
Department of Computer Science
Stanford University
Stanford, CA 94040
seanraff@stanford.edu
codalab: seanraff

Ted Li
Department of Computer Science
Stanford University
Stanford, CA 94040
tedli@stanford.edu
codalab: tedli

Abstract

The reading comprehension machine learning task involves reading in a question and returning an answer from an associated context paragraph. This task has proven to be difficult, as the performance of state-of-the-art models still do not compare with human performance. The difficulty of the tasks comes from understanding two separate pieces of information as well as the relationship between them before picking a response from the context. Recently, machine learning models have used attention to extend other natural language processing tools to reading comprehension, and among these methods is the Bidirectional Attention Flow (BiDAF) model, which was first introduced by Seo et al. [1]. In this paper, we present experimental results from variations of the default BiDAF model.

1 Introduction

Applying machine learning to the reading comprehension task, also knowns as machine comprehension (MC) has become more popular recently. As a task, it has proven to be challenging but also valuable for many other tasks. The challenge mainly arises from the depth of reading comprehension. Models designed for end to end machine comprehension typically must generate an understanding of both the question and the context, and afterwards, must determine the relationship between the two, picking they key word(s) in the context that answer the question.

2 Background

In recent years, much progress has been made in the MC task, many of which use variations of attention as their method of relating the question to the context. However, even with these improvements, machine learning models have not been able to match human performance. In this paper, we attempt to improve on the tools that are used for MC.

2.1 Datasets

In order to implement and test variations on the BiDAF model, we had to use several datasets for embeddings as well as questions and answers.

2.1.1 Stanford Question Answering Dataset

In order to evaluate our model, we use the same dataset that Seo et al. used - the Stanford Question Answering Dataset (SQuAD), which was released in 2016 by Rajpurkar et al. [2]. SQuAD consists

of around 100 thousand pairs of questions and context paragraphs, as well as their associated answers. The contexts were taken from Wikipedia articles, while the answers were labeled by humans using a span of the context.

2.1.2 Global Vectors for Word Representation

In order to speed up our training process, we used pre-trained word embeddings known as Global Vectors for Word Representation (GloVe), which were generated by Pennington et al. using co-occurrence statistics [3]. We used the GloVe vectors training on Wikipedia 2014 and Gigaword 5, which has 6 billion tokens with a vocabulary of 400 thousand words.

2.2 Related work

Most of the newer models that have been applied to the reading comprehension task have used variations of generating attention vectors using vector representations generated from the question and answers.

2.2.1 Bi-directional Attention Flow for Machine Comprehension

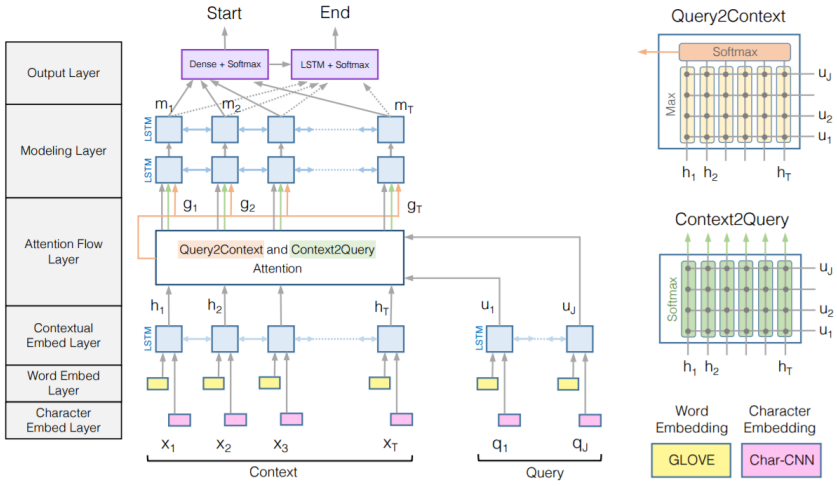

Figure 1: **BiDAF Model** Overall structure of bi-directional attention flow presented by Seo et al. in their 2017 paper [1]

In their paper, Seo et al. present a new model for answering the MC task. Their BiDAF model starts off by creating contextual embeddings for both the question and the context. These contextual embeddings are generated using word embeddings and character embeddings, and combined using a highway network and fed into a bi-directional LSTM. Afterwards, the contextual embeddings are combined into an attention using a novel method described later. The attention is then fed into the modeling layer which consists of two bi-directional LSTMs. The output of the modeling layer is used for the output layer, which predicts the start word and the end word. The start word is predicted by applying a linear transformation to the output of the final LSTM. The end word is predicted by applying yet another bi-directional LSTM to the final modeling layer and then applying a linear transformation. They then take the softmax of these predictions and use the maximum value as the predicted start and end token indices.

The attention vectors that are generated by the BiDAF offer many improvements over previous methods. One reason is that their attention flow generation does not involve summarizing the context into a fixed-length vector. Instead, an attention is generated for each word in the context, allowing the context as well as its attention to freely flow to the modeling layer. This structure avoids early

summarization and loss of information. In addition, the attention vector is not computed recursively, and while this means that the attention vectors are not context dependent, it may allow for attention vectors to focus on the relationship between individual words in the context and the entire query. Another major difference in their model is that they compute attention in two directions - from context to query, as well as from query to context. By combining both of these attentions, they are able to better represent the complex relationships between the question and the query.

3 Approach

Our approach to testing and improving the BiDAF model involved implementing incrementally more complicated versions of the model. Using this method, we were able to test out various parameters and tools for optimizing our performance at each step and make small changes at each step, allowing us to safely explore the model.

3.1 Simple model

We began by implementing a simpler version of the model where we only kept the layers that we felt were vital for the task. As a result, we did not use character level embeddings, had only one bi-directional LSTM layer in the modeling layer, and did not use a bi-directional LSTM before the linear transformation for predicting the end word index. These layers were picked in particular because without them, we still maintain the major layer groups that Seo et al. describe in their paper: contextual embedding, attention, modeling, and output [1]. By simplifying these layers, we were able to reduce the runtime and the complexity of our model so that we could diagnose and fix problems in a more efficient manner. Other design choices included using $\alpha(h, u) = w_{(S)}^T[h; u; h \circ u]$ to generate our similarity matrix, using $\beta(h, \tilde{u}, \tilde{h}) = [h; \tilde{u}; h \circ \tilde{u}; h \circ \tilde{h}]$ for our attention vector, and to use linear weight vectors and the dot product operation for our final predictions.

After we were able to successfully overfit our model on a smaller training dataset, we began to train and tune it on the entire train dataset and evaluate it on the development set.

3.2 BiDAF without character embeddings

The next model that we implemented was very similar to the default model as detailed in the paper by Seo et al. Them only difference was that we did not use character embeddings in the construction of our contextual embeddings. Again, after we were able to successfully overfit our data, we began to train and tune our model on the entire dataset.

3.3 Introducing a new output layer

In their paper, Seo et al. make note that many different output layers are possible depending on the task that is at hand. In particular, they state that for their task, the output is created by taking the dot product of a modeling vector with a weight vector. For the start word prediction, the modeling vector is a concatenation of the input to the modeling layer and the output of the second modeling bi-directional LSTM, while the start weight vector is a trainable weight vector. For the end word prediction, the input is a concatenation of the input to the modeling layer and the output of a third modeling bi-directional LSTM (whose input was the output of the second modeling bi-directional LSTM), and the end weight vector is a different trainable weight vector.

While the two predictions are directly or indirectly made using the same data (output of the second modeling LSTM), the final predictions are done independently of each other. Because of this, it's possible that the highest scores for the start and end positions may have the end position before the start. To fix this, when making their final prediction, the authors pick their start and end index by finding the pair of start and end probabilities that have the greatest product with the condition that the start index is less than the end index. While this may help improve their test and validation scores, this is done outsize of the model and involves use of outside information - knowledge that the start of a phrase should be before the end of a phrase. As a result, the cross entropy loss function that they use is not as useful for optimizing the model. Take for example a case where the start index has the greatest probability at index a. In this case, if the end index has the largest probability at index b

and second largest probability at index c, under the order conditions that $b < a < c$, then the model will correctly predict the answer as (a, c), but there will still be a relatively larger gradient since c is not the highest predicted end word. While this is usually fine, it may have negative implications on other interactions down the line, since it is possible that b is the better choice, and that c should only be picked because of its positional relationship with a, and our model (with respect to what is trained and where we back-propagate) does not understand or make predictions about this.

To help address this issue, we propose generating a new input to the third bi-directional LSTM - the one whose output is linearly transformed into the prediction for the end word. This new input needs to account for the interaction between the start and end indices. To do this, we take advantage of tools that are used elsewhere in the BiDAF model. When calculating the similarity matrix and attention vectors, the model has to consider the interactions between two separate matrices. Seo et al. primarily accomplish this by concatenating three groups of vectors. The first two are the vectors that are interacting, and the third is the Hadamard product of the two. Thus, we propose that given a start token prediction vector, $p_s \in \mathbb{R}^{c \times 1}$, where c is our context length, and the output of our second modeling bi-directional LSTM, $l2 \in \mathbb{R}^{c \times 2d}$, where d is our LSTM output size, that the input to our third bi-directional LSTM for end-token prediction be $[p_s; l2; p_s \circ l2]$, where $\circ$ is a broadcasted Hadamard product. After adding this layer to our model, we again trained and tuned our model on the entire dataset.

4 Experiments

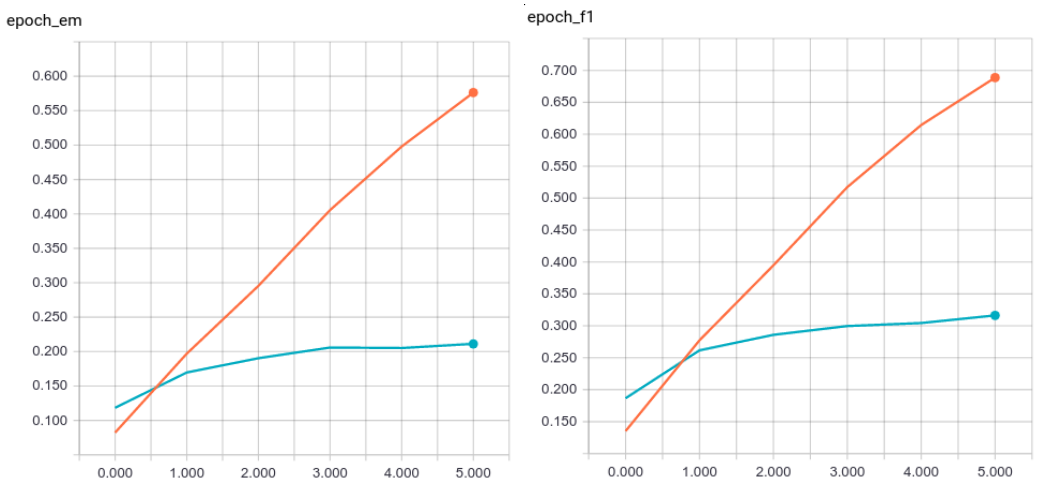

Figure 2: **Overfitting** The exact match score (EM) is plotted on the left and the F1 score is plotted on the right. Results on the training dataset are in orange and results on the validation dataset are in blue. In this example we did not apply any dropout or regularization. We quickly fit the data, reaching 70% F1 and 60% EM on the train set in just 6 epochs. However, we are quickly overfitting on the validation set as indicated by EM plateauing at around 20% and F1 plateauing at around 30%. This suggests we should regularize our model by penalizing large weights or adding dropout.

We now analyze the results tabulated in Table 2. Comparing rows two and four shows that adding the extra complexity of two more LSTM layers yields drastically more overfitting. This reinforced the importance of dropout in our more complex models. Comparing rows three and four show that while dropout can help with overfitting, it will also cause our models to take longer to train. Row one is an initial promising result combining a more complex model with dropout. In row five we used our new layer that feeds into the end word predictions which incorporates information about the start word. We hypothesize that this would achieve better EM and F1 scores if we had used less dropout and allowed it more time to train. Although we were overfitting, it was only slightly and reducing dropout may allow our model to achieve even higher EM and F1 scores on the validation set.

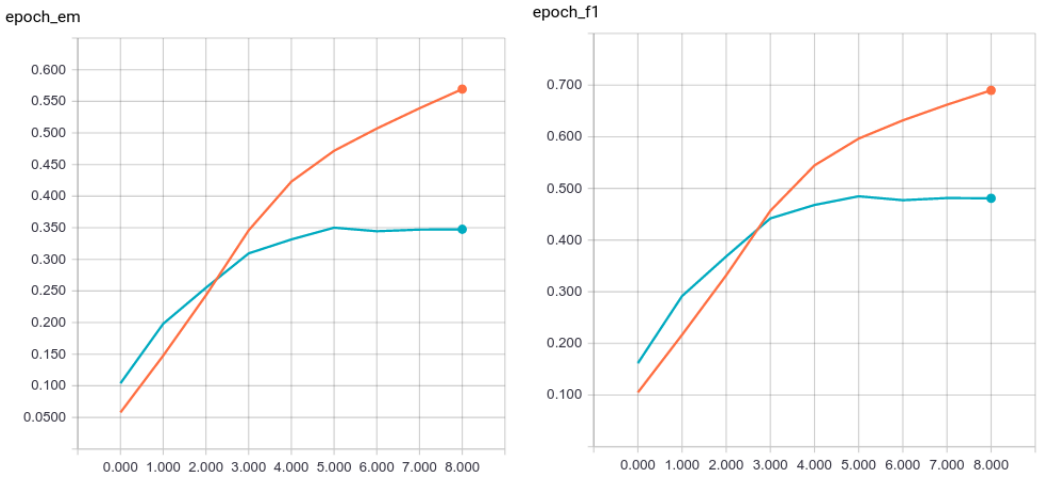

Figure 3: **Convergence** The exact match score (EM) is plotted on the left and the F1 score is plotted on the right. Results on the training dataset are in orange and results on the validation dataset are in blue. Once again we have issues with overfitting. We can see a clear point where we start overfitting and stop learning general features. These graphs are for the fifth row in Table 2 which happens to be our bet performing model.

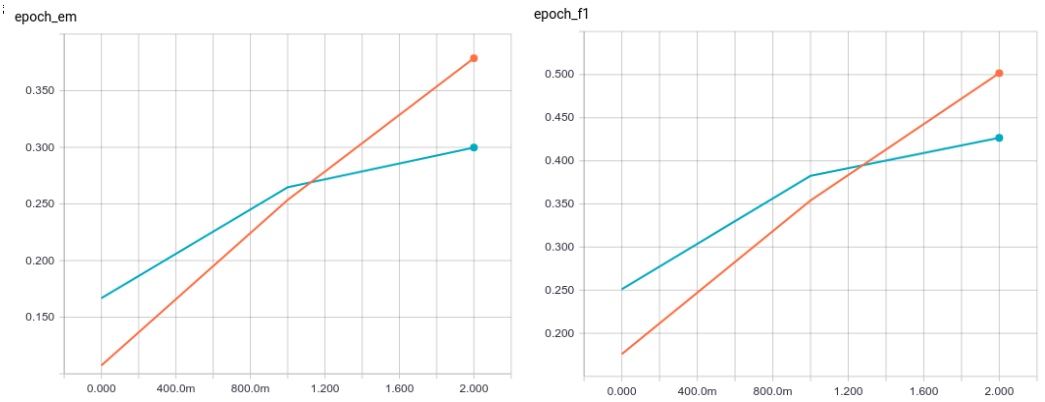

Figure 4: **Without Start and End Interactions** The exact match score (EM) is plotted on the left and the F1 score is plotted on the right. Results on the training dataset are in orange and results on the validation dataset are in blue. This model does not use the start and end interaction layer.

After achieving decent results with the simple model we tested how much adding the start and end interaction layer benefits the model. We ran two models with 0.001 learning rate, 0.96 exponential learning rate decay, 0.10 dropout, 5 LSTM layers, and gradient clipping. In one of these models we omitted the start and end word interaction layer and in the other model we included it. The model with the interaction layer performed significantly better, achieving 42% EM and 57% F1 on the dev set while the other model only achieved 30% EM and 43% F1 on the dev set. We conclude that modeling the interaction between the start word on the end is potentially valuable.

5 Conclusions

Overall, we were able to get a working implementation of three versions of the Bi-directional attention flow model. With these models, even though the performance does not match that of the original papers, we were able to gain insights about various hyperparameters, as well as the benefits and drawbacks of increased complexity on the BiDAF model. Furthermore, we proposed a new

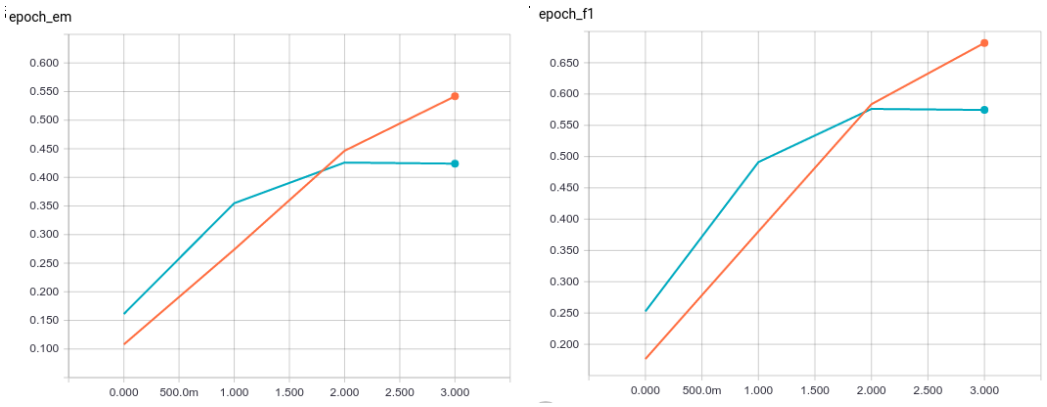

Figure 5: **With Start and End Interactions** The exact match score (EM) is plotted on the left and the F1 score is plotted on the right. Results on the training dataset are in orange and results on the validation dataset are in blue. This model is the same as Figure 4 except with the start and end interaction layer added on top.

Table 1: Tabulated Experiments

LR	Decay	Drop	# LSTM	Clip	EM@2	EM@4	F1@2	F1@4	Test EM/F1
0.001	1.0	0.5	5	True	16%, 14%		23%, 22%		
0.001	1.0	0.0	5	False	20%, 17%	58%, 21%	28%, 26%	69%, 32%	
0.001	1.0	0.5	3	False	3%, 8%	10%, 4%	10%, 13%	20%, 12%	
0.001	1.0	0.0	3	False	24%, 18%	38%, 24%	38%, 24%	54%, 38%	
0.001	0.96	0.2	5	True	20%, 15%	47%, 35%	29%, 22%	54%, 47%	
0.001	0.96	0.35	5*	True	16%, 18%	35%, 28%	24%, 27%	47%, 41%	
0.001	0.96	0.1	5*	True	10%, 16%	54%, 42%	18%, 25%	68%, 57%	44.467%, 56.784%
0.001	0.96	0.1	5	True	17%, 11%	38%, 30%	18%, 25%	50%, 43%	

Table 2: We tabulate the results from some of our experiments above. Learning rate (LR) and learning rate decay are the first two columns. Next is the probability of dropping out outputs from all LSTM layers and dropping out the inputs to the first LSTM layers. Next is the number of LSTM layers we used in our model. The fifth column indicates whether we used gradient clipping. The final four columns are the exact match (EM) and F1 scores for the training and validation datasets after the second and fourth epochs. The left percentage in each column are averages over an entire training epoch while the right percentage in each column are computed over the entire validation dataset at the end of each training epoch. An asterisk in the "# LSTM" column indicates we used the start and end word interaction layer.

output layer that leverages tools that are used earlier in the BiDAF model in order to consider the interactions between the start and end token positions. While this does not completely solve the issue, since it is still possible to predict a start token after an end token, our early results show that it may be helpful for increasing performance. We believe that this is because it puts more emphasis on the positional relationship between start and end, decreasing the necessity of the post-model selection and thus increasing the expressive power of our model, leading to a more relevant loss function. Given more time, we would like to pursue this idea further with further hyperparameter and overfitting tuning on both the original model and our proposed new model.

Acknowledgments

We would like to thank the cs224n staff for all of their help throughout this project with a data, preprocessing, ideas, debugging, and GPU instances to allow us to really explore our project. In addition, we would like to thank Microsoft Azure for providing us with Azure GPU computing time.

Contributions

Sean: Set up most of the initial system. Handled submissions.

Ted: Set up most of the initial model.

Approximately equal: Contributed to model tuning/improvements, new ideas. Fixes to the code (model, loss, masking, etc). Tracking model performances. Writing the paper/filling out the poster.

References

[1] Seo, M., Kembhavi, A., Farhadi, A., & Hajishirzi, H. (2017) Bidirectional attention flow for machine comprehension. In ICLR.

[2] Pranav R., Jian Z., Konstantin Z., & Percy L. (2016) SQuAD: 100,000+ questions for machine comprehension of text. In Proceedings of the Conference on Empirical Methods in Natural Language Processing.

[3] Pennington, J. & Manning, C. (2014) Glove: Global vectors for word representation.