*NQSotA Continuation Curriculum Learning with Question Answering on the SQuAD Dataset

Luke Johnston
Department of Computer Science
Stanford University
Stanford, CA 94305
lukej@stanford.edu

William Chen
Department of Computer Science
Stanford University
Stanford, CA 94305
wic006@stanford.edu

Rahul Palamuttam
Department of Computer Science
Stanford University
Stanford, CA 94305
rpalamut@stanford.edu

Codalab username: rpalamut

Abstract

We implement a slightly simplified Bi-Directional Attention Flow Model[4] and a slightly modified Multi-Perspective Context Matching[6] model for Question Answering on the SQuAD dataset. In the Multi-Perspective model, we add perspective matching between forward and backward contexts. We omit the character-level embeddings of both models, and make a few other small simplifications. We briefly looked at the effects of curriculum learning as a continuation method for our BiDAF network. Curriculum learning[1] is inspired by the education system, in which subjects are trained through an organized curriculum of varying tiers of difficulty. This differs from other machine learning approaches where training is done on random samples. We hoped this could improve performance on longer length questions.

1 Introduction and Related Work

In this paper, we train two models on a recent Machine Comprehension and Question Answering dataset, the Stanford Question Answering Dataset. This dataset consists of a number of question-paragraph pairs, where the answer to each question is contained somewhere in the paragraph. Hence, the task of the model is to predict the start and end index of the answer in the paragraph.

Our model structures are taken from previous work that has been done on the SQuAD[3] dataset, with a few modifications. Due to the inherent difficulty of end-to-end machine comprehension tasks, a variety of approaches have been suggested to predict answer spans from questions and contexts. These approaches range from how attention is computed to how answer spans are identified. In the Bidirectional Attention Flow model of Seo et al [4], a multi-stage hierarchical attention mechanism uses various attention computations to create a query-aware representation of the context which is then used to predict the answer location in the paragraph. On the other hand, the Multi-Perspective Context Matching model of Wang et al[6] uses a set of perspective matchings between the questions and contexts to predict the answer span. The details of these models are explained in their respective

*NQSotA stands for Not Quite State of the Art

sections below. We explore the efficacies of these models while additionally looking into how a curriculum learning strategy could affect their results.

2 Dataset and Pre-processing

The recently released Stanford Question Answering Dataset (SQuAD) consists of over over 100,000 questions, each matched with a paragraph that contains the answer. 10k of these pairs are reserved for testing, and the remaining 90k are split into 85k training and 5k validation datapoints. Given a question / paragraph pair, the model must predict the span of words in the paragraph that gives the answer to the question. As the first step of training the model, the 300-dimensional GLoVe vector encoding of each word in the question and paragraph is looked up. For our implementation of the Multi-Perspective model these vector representations of each word are made into variables (that are only initialized with the GLoVe embeddings), but for BiDAF the GLoVe embeddings are held constant. Additionally, we truncate all paragraph lengths at 200 words. If the answer falls after the truncation threshold, we set the answer span to be the last word in the paragraph. This truncation loses less than 1% of training and validation examples, but results in significant training speedup.

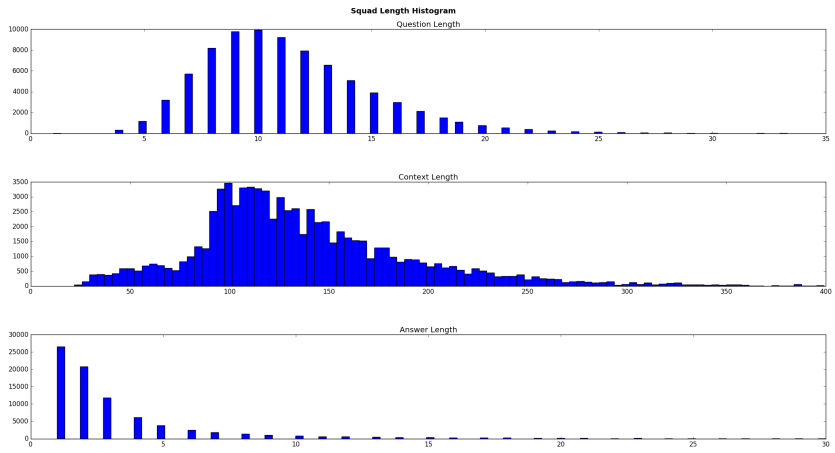


Figure 1: Distribution of question, paragraph, and answer lengths in the SQuAD dataset

3 Model 1: Slightly Modified Multi-Perspective Context Matching

3.1 Filtering

The first layer of the MPCM network filters the paragraph word embeddings by their similarity to the question word embeddings. The cosine similarity is taken between each pair of question and answer words

$$r_{ij} = \frac{q_i^T p_j}{||q_i||||p_j||}$$

and then we compute the maximum similarity each paragraph word has with any question word

$$r_j = \max_i r_{ij}$$

Then each paragraph word is filtered

$$p'_j = r_j p_j$$

The purpose of this layer is to allow the model to filter out irrelevant paragraph words. By identifying words in the paragraph that are similar to the words of the question, the model is one step closer to determining where the answer is.

Note that the word embeddings q_i and p_j are initialized with the GLoVE embeddings, but are trainable variables so can be trained for this specific task. Additionally, the original MPCM paper uses a character-level embedding in addition to the GLoVE embeddings, but we decided to omit this step since it only results in an increase of 3% in the F1 score (according to the original MPCM paper).

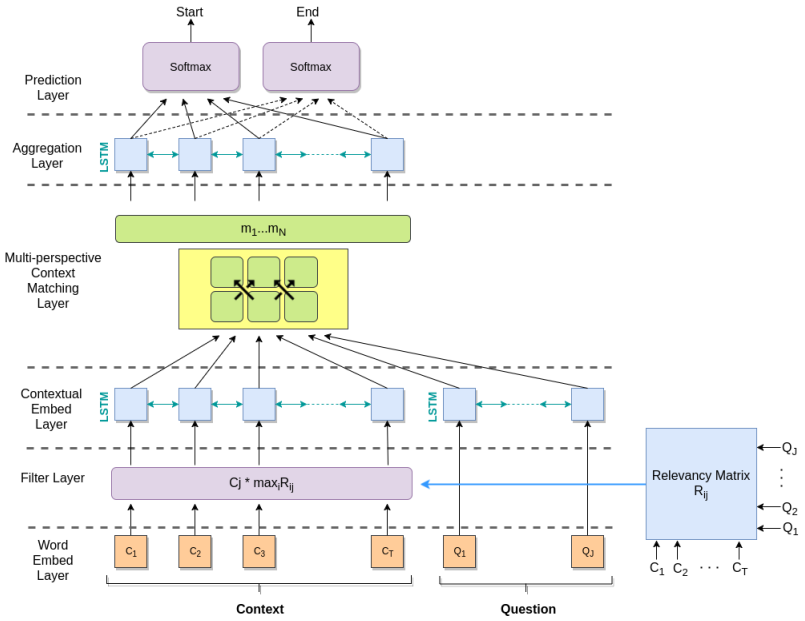


Figure 2: Multi-Perspective Matching Model Architecture

3.2 Context Representation

Let $Q \in \mathbb{R}^{100 \times n}$ be a matrix of all the embeddings of the question words, filtered with the previous layer, where n is the length of the question. Likewise, let $P \in \mathbb{R}^{100 \times m}$ be the embeddings of the context words, filtered with the previous layer, where m is the length of the paragraph. Note that $m \leq 200$, since we truncated long paragraphs, unlike the original MPCM paper. We apply a bidirectional LSTM with state size 100 to the question and paragraph separately to obtain context-aware representations of each word

$$U^f, U^b = LSTM(Q) \in \mathbb{R}^{200 \times n}$$

$$H^f, H^b = LSTM(P) \in \mathbb{R}^{200 \times n}$$

which are passed to the Multi-Perspective layer. We treat the forward and backward context-aware representations separately: U_f is the forward context-aware representation of the question, U_b is the backward context-aware representation of the question, etc.

3.3 Multi-Perspective Context Matching Layer, with Modification

The "perspectives" of the model name are a variable $W \in \mathbb{R}^{20 \times 100}$, where 20 is the number of perspectives we decided to use (the paper used up to 50 perspectives), and 100 is the size of the context embeddings. For single perspective W_i (a row of W), for each context-aware representation, we compute the perspectives on that representation

$$R^f = W_i \circ U^f$$

$$R^b = W_i \circ U^b$$

$$S^f = W_i \circ H^f$$

$$S^b = W_i \circ H^b$$

where $\circ$ represents broadcasted element-wise multiplication. Then, for each pair of question and paragraph indices (i,j) we compare the similarity between these perspectives

$$m^{ff} = \cos(R_i^f, S_j^b)$$

$$m^{fb} = \cos(R_i^f, S_j^b)$$

$$m^{bf} = \cos(R_i^b, S_j^f)$$
$$m^{bb} = \cos(R_i^b, S_j^b)$$

to get four "m-values" (matching values) for each perspective. This is the main modification to the MPCM paper - in the original paper, they only use m^{ff} and m^{bb}. We thought that including m^{fb} and m^{bf} should help the model, for example, match forward-contexts of the question with backward-contexts of the paragraph, which could be important.

If we now consider the matrix M^{ff} of all m^{ff} values, it is a matrix of one perspective matching between the forward contexts (of question and paragraph locations). From these matchings, we compute three final metrics for each paragraph location, following the MPCM paper. The following steps are done for M^{ff}, M^{fb}, M^{bf}, and M^{bb}:

Full-Matching: $m_i^{full} = M_{1i}^{ff}$

Max-Matching: $m_i^{max} = \max_j M_{ji}^{ff}$

Mean-Matching: $m_i^{mean} = \frac{1}{n} \sum_j M_{ji}^{ff}$

For the full matching step when we are considering the backwards question context embeddings (m^{fb} or m^{bb}), the 1 index is replaced with n (the length of the question words). This gives us three values for each context word: the perspective matching between the paragraph word and a full representation (either forward or backward) of the question, the max matching with any location in the question, and the mean matching with the entire question. Hence the output of this layer is $20 \times 4 \times 3 = 240$ matching values for each location in the context.

3.4 Aggregation Layer and Prediction

The output of the previous layer is sequence of vector representations of the paragraph. This sequence is fed through another bidirectional LSTM[2] of state size 100 to aggregate the matching information over time. The forward and backward outputs are concatenated to produce a final representation of each paragraph location, $o_j \in \mathbb{R}^{200}$. To obtain the predicted probabilities for the start index, each o_j is passed through a trainable affine transformation to be mapped to a single number, and then a softmax is taken over these numbers. The predicted end index probabilities are likewise obtained with another affine transformation and softmax layer.

3.5 Training Parameters

We used the Adam optimizer with a learning rate of 0.0001, following the paper, and a batch size of 32. Dropout is applied to the inputs of each LSTM with a keep probability of 0.2.

4 Model 2: Bi-Directional Attention Flow (BiDAF)

4.1 Contextual Representation Layer

First, we obtain a representation of the context of each word (in the paragraph and the question) using a Highway Network [5] + Bidirectional LSTM. Each GLoVe embeddings $a_i \in \mathbb{R}^{300}$ is forwarding through a two-layer highway network to get a new word embedding w_i. The first layer is a simple feed-forward layer with ReLU nonlinearity to map the GLoVe embedding $\in \mathbb{R}^{300}$ to a vector $b_i \in \mathbb{R}^{100}$, and the second layer performs the highway mixing:

$$n_i = ReLU(W_n b_i + b_n)$$
$$g_i = Sigmoid(W_g b_i + b_g)$$
$$w_i = n_i \circ g_i + b_i \circ (1 - g_i)$$

to get the new representation of each word $w_i \in \mathbb{R}^{100}$. n_i is the transformed word embedding, g_i is the gate of the highway network that specifies how much of this transformed word embedding to keep (and how much of the original embedding to keep), and w_i is the new embedding. The primary purpose of this highway network is to map the GLoVe vectors from size 300 down to the size of the rest of the model's hidden representations (100), but it also serves as an additional way for the

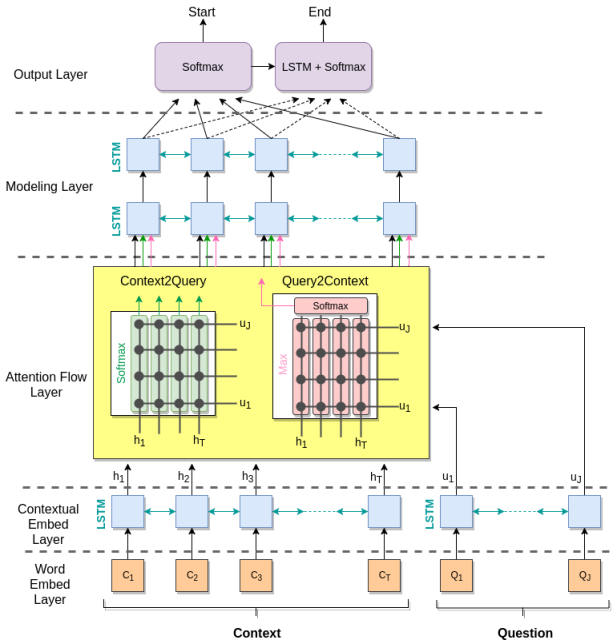


Figure 3: BiDAF Model Architecture

model to fine-tune the GLoVe encodings for the following task of context representation. The only difference in this step from the original BiDAF paper is that it uses word-embeddings created from a character-level RNN as input to the highway layer in addition to the GLoVe embeddings.

The next step uses a bidirectional LSTM RNN to obtain context representations of each location in the question and the answer. This step is identical to the context representation step in the MPCM model. Let $Q \in \mathbb{R}^{100 \times n}$ be a matrix of all the embeddings of the question words, processed with the previous layer, where n is the length of the question. Likewise, let $P \in \mathbb{R}^{100 \times m}$ be the embeddings of the context words, processed with the previous layer, where m is the length of the paragraph. Note that $m \leq 200$, since we truncated long paragraphs, unlike the original BiDAF paper. We apply a bidirectional LSTM with state size 100 to the question and paragraph separately to obtain context-aware representations of each word

$$U = LSTM(Q) \in \mathbb{R}^{200 \times n}$$

$$H = LSTM(P) \in \mathbb{R}^{200 \times n}$$

which are passed to the Attention Flow Layer. The size of each representation is 200 now because it is the concatenation of the forward and backward outputs of the bidirectional LSTM.

4.2 Attention Flow Layer

In the attention flow layer, first an attention value a_{ij} for each pair of paragraph and question words + context representations h_i, u_j is obtained

$$s_{ij} = w_{(a)}^T [h_i; u_j; h_i \circ u_j]$$

to form a matrix $S \in \mathbb{R}^{m \times n}$. $w_{(S)}^T \in \mathbb{R}^{600}$ is a trainable parameter vector, and $[;]$ notation denotes concatenation. The values of this matrix can be thought of as how much "attention" is given to the corresponding pair of question and paragraph locations. These attention values are then used to compute the following two steps:

Context-to-Query Attention For each location in the paragraph t, we compute the attention over each question location with

$$a_t = Softmax(S_{t:}) \in \mathbb{R}^n$$

where $S_{t:}$ is the tth row of S. These attention values are used to create a single vector, representing the attention-focused question, at this location in the paragraph:

$$\tilde{u_t} = \sum_{i=1}^{n} a_{ti}$$

Query-to-Context Attention In the above section, we compute an attention-focused representation of the question FOR EVERY location in the paragraph. In this section, we compute a single attention-focused representation of the paragraph (that does not correspond to any particular location in the question, but rather is focused based on the entire question). This is done by first taking the maximum along each column (second dimension) of S:

$$s' = \max_{col}(S) \in \mathbb{R}^m$$

which gives a value for each context word representing the max importance that context word has, with reference to any point in the question. Then we obtain the attention-focused representation of the context as

$$\tilde{h} = \sum_{i=1}^{m} Softmax(s')_i$$

Once we have obtained both the attention-focused representations of the question, and the attention-focused representation of the paragraph, we create a single vector g_t by combining the original context-aware paragraph representation with the attention-focused representations as follows:

$$g_t = [h_t; \tilde{u_t}; h_t \circ \tilde{u_t}; h_t \circ \tilde{h}] \in \mathbb{R}^{800}$$

4.3 Modeling + Prediction

The modeling layer of BiDAF is similar to the modeling layer of Multi-Perspective, except it uses a two layers bidirectional LSTM RNN to produce the final contextual embeddings of the context words. The inputs to this two layer bidirectional LSTM are the g_t vectors from the previous layer, and the outputs we denote m_t^1. These outputs are additionally passed through a third LSTM layer to produce m_t^2. Two final numbers for each location are computed

$$p1_t = w_{p1}^T[m_t^1; g_t]$$

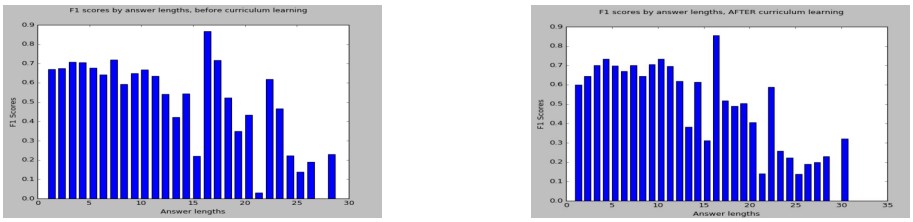
$$p2_t = w_{p2}^T[m_t^2; g_t]$$

and an application of softmax to the vectors $p1$ and $p2$ produces the final predicted probability distribution over the start and end index of the answer span, respectively.

4.4 Training, Regularization, Masking

Bidaf was trained using the Adam Optimizer, with a learning rate of 0.0003 and exponential decay every 32 batches by a factor of 0.999. Dropout for regularization was applied before each LSTM layer and before the last linear transformation before the output predictions. The dropout rate was 0.2 in all cases. We used a minibatch size of 32, and since not every example in a batch has the same question and paragraph length, we applied exponential masking before each softmax layer to ensure the probabilities for each word location were only nonzero if paragraph / question was long enough to contain that word location.

We decided to use the Adam optimizer instead of AdaDelta (what the original BiDAF paper used) because after a bit of research it seemed that Adam was a more recently developed and generally considered superior method, and more commonly used in the literature. However we later found out that this is not always the case, so one future thing to explore would be trying the original AdaDelta optimizer, or even others like RMSProp. Originally we were trying a higher learning rate of 0.1, but the model would get stuck in a local minima with training loss around 4.5, so we decreased the learning rate until that did not occur.



(a) F1 BiDAF (b) F1 curriculum learning

Figure 4: The F1 scores of both the BiDAF model to the right after 16 epochs is shown. On the left is the F1 score after 4 epochs of curriculum learning.

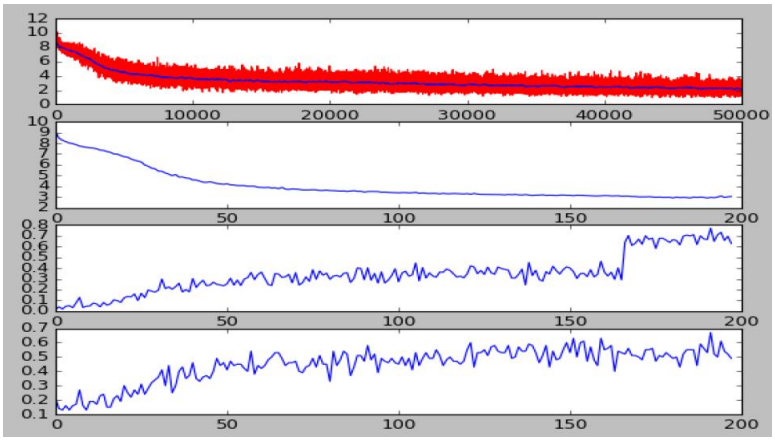


Figure 5: Graph of training loss, validation loss, validation F1, and validation exact match. Note that the F1 plot is slightly off. The large jump is due to a subtle calculation bug in F1 being fixed (our model actually had higher F1 than we thought for a long time).

5 Curriculum Learning as a Continuation Method

We suspected that it might be more difficult for the model to predict answers with a longer length, and decided to implement a continuation curriculum learning to train the model on more examples with longer answers lengths. To do this, during every validation step (every 1000 batches), the F1 scores are evaluated for answers of each particular length. Then, when forming a new batch during training, we sample from answer lengths with the following frequency:

$$f_l \propto (1 - F1[l])^2 N_l$$

where f_l is the sample frequency of datapoints with length l, $F1[l]$ is the $F1$ score of answers of length l on the validation dataset, and N is the number of answers of length l on the validation dataset. This causes the model to train on answer lengths it does poorly on more often. We took our best model parameters and trained them on this continuation curriculum learning strategy for 4 epochs.

6 Results

Our final F1 and EM scores on the test set for our implementation of BiDAF were

$$F1 = 60.54\%$$

$$EM = 45.64\%$$

after 16 epochs of training. Our modified multi-perspective model performed worse, only achieving an $F1$ of 50% on the validation dataset, and an EM of 38% after 20 epochs of training. When training our multi-perspective model, we at first were never able to get higher than a validation

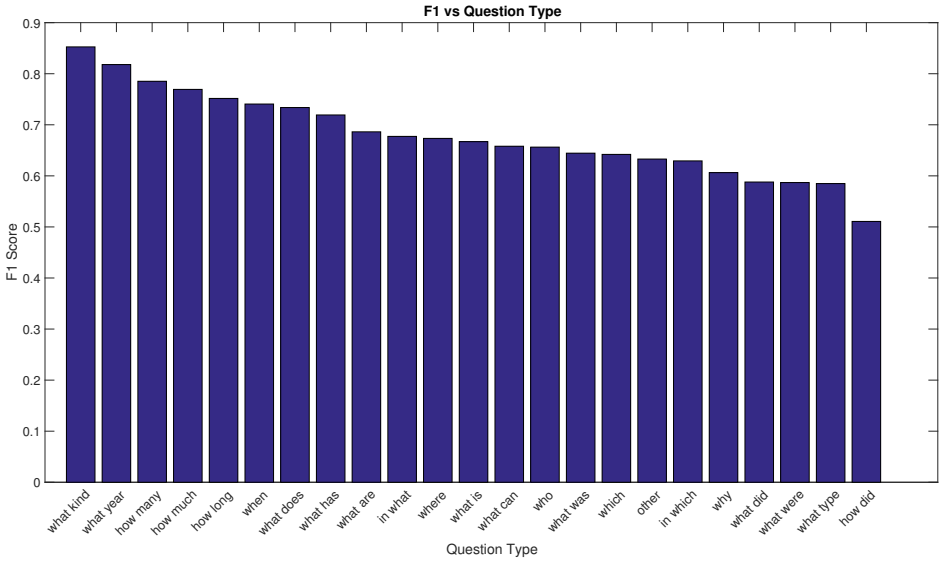


Figure 6: The validation F1 scores of the BiDAF model on various question types (types taken from Wang et al[6]) after 16 epochs.

F1 score of approximately 20%, and we could not determine our problem, which was our reason for deciding to implement BiDAF as well. We later realized that we had an off-by-one error when computing the word list from the span, and when we fixed that we realized our F1 was actually around 40% all along. The additional increase to our final MPCM model's performance came from decreasing the learning rate - initially, we were getting stuck at a local minima with a learning rate of 0.01. We suspect that further tuning the learning rate and the exponential decay rate could improve performance further, but did not have enough time to fully explore this.

Our curriculum learning results can be seen in figure 4. For high question lengths there are very few training examples, so the F1 scores do not mean much. The main consistent effect of our curriculum learning was a slightly lower F1 score for answers of lengths 1 and 2, and slightly higher F1 scores for some answer lengths in the range (3-8). This adjustment came at the cost of a lower $F1$ score, so it seems that encouraging the model to focus on harder examples requires it to perform worse on the easier examples, which lowers the total score in this case. Also, there is really not that much difference between the original F1 scores for answers of lengths $1-5$, which compose most of the dataset (see Figure 1), so perhaps a better focus for curriculum learning would have been the type of question, rather than length of answer. Towards this end, we graphed the F1 scores on different types of questions in figure 6, but did not have enough time to train with curriculum learning on this metric. The model seems to perform better on questions which can be answered with fewer words (shorter answer spans) and/or those which expect numerical responses: what year, how many, etc. These findings are consistent with the results from the answer length to F1 experiments mentioned above.

7 Conclusions and Future Work

We have implemented two Deep Learning architectures for Question Answering, namely Bidirectional Attention Flow and a modified Multi-Perspective Context Matching. Simplified versions of the architectures were shown to be effective and resulted in decent performance on the Stanford Question Answering Dataset. We then attempted leveraging Curriculum learning as a continuation method on our best model. However this was shown to decrease performance as the model began losing its ability to generalize the majority of examples and focused more on the outliers. This result may indicate that performing very well on the questions with longer answer spans could require a different type of model than the ones we experimented with which do well on shorter spans.

For future work, we also want to leverage both BiDAF and MPM models by concatenating their outputs and calculating loss. We believe the combined approach could result in improved performance. We also want to look at how utilizing character embeddings in the ensemble approach can potentially increase performance. And finally, we still believe that our model's hyperparameters can be fine-tuned to achieve accuracy more closely representing the reported results in the two papers.

8 Acknowledgments

We would like to thank our instructors Professors Chris Manning and Richard Socher as well as the TA's for putting together a great course on Deep Learning and Nautral Language Processing. We would also like to thank the incredibly helpful team of teaching assistants for this class. Finally, we would like to thank other students posting on Piazza and working late at night at Huang whose unwavering support and thirst for F1 scores kept us going.

9 References

References

[1] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning.

[2] Jason P. C. Chiu and Eric Nichols. Named entity recognition with bidirectional lstm-cnns. *CoRR*, abs/1511.08308, 2015.

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

[4] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.

[5] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

[6] Zhiguo Wang, Haitao Mi, Wael Hamza, and Radu Florian. Multi-perspective context matching for machine comprehension. *CoRR*, abs/1612.04211, 2016.