SQuAD Challenge

Mackenzie Pearson
pearson3@stanford.edu

Matthew Creme
mcreme@stanford.edu

Raphael Lenain
rlenain@stanford.edu

Abstract

SQuAD is a reading comprehension dataset consisting of 100,000+ questions posed by crowdworkers on a set of wikipedia articles. The answers to these questions are substrings taken directly from the relevant wikipedia passage. The SQuAD challenge is to implement a model that given a passage and a question about that passage, can output the correct answer. In this paper we begin by exploring this SQuAD data set, looking at the structure of the passages and questions. Next we extend the baseline model [5] as described in the introduction to build an improved model for the SQuAD challenge. To perform this task we implement methods seen in two recent papers. First we implement Self Attention from the R-Net paper [1]. In addition to this we implement the BiDaf method [3].

1 Introduction

The baseline model introduced in [5] is quite a good starting point to approach this SQuAD challenge. Firstly, GloVe vector word embeddings are provided as representations for our context and question words. Next, a bi-directional GRU is implemented on both the context and question word embeddings as:

$$\{\overrightarrow{c_1}, \overleftarrow{c_1}, \cdots \overrightarrow{c_N}, \overleftarrow{c_N}\} = \text{biGRU}(\{x_1, \cdots x_N\})$$
$$\{\overrightarrow{q_1}, \overleftarrow{q_1}, \cdots \overrightarrow{q_N}, \overleftarrow{q_N}\} = \text{biGRU}(\{y_1, \cdots y_N\})$$

After this, a basic attention layer, which will be approved upon later in Section 4, is implemented.

$$\begin{aligned} e^i &= [{c_i}^T q_1, \cdots {c_i}^T q_N] \\ \alpha^i &= \text{softmax}(e^i) \quad \text{for } i = 1, ..., N \\ a_i &= \sum_{j=1}^{M} \alpha_j^i q_j \quad \text{for } i = 1, ..., N \end{aligned}$$

The outputs a_i of the attention layer are then concatenated with the outputs c_i of the biGRU. This concatenation is then fed into a final fully connected layer. From there, start and end logits are created through two seperate linear layers from the output of the fully connected layer. Lastly, we take a softmax over the start and end logits in order to create a probability distribution over start and end points. Our predictions are then generated as the argmax over these distributions.

In the upcoming sections, we detail information about the underlying data, our improvements to the baseline model as well as the results that we obtained.

2 Initial Data Exploration

We begin by exploring our train data set to see if there is any obvious bias in the data that could affect how we choose hyper parameters such as context length and question length.

2.1 Context Length

We analyze the context length below, where length is defined as the number of tokens in the context paragraph. The mean context length is 138, with median 127 and standard deviation of 57. We plot the distribution of context lengths in Figure 1.

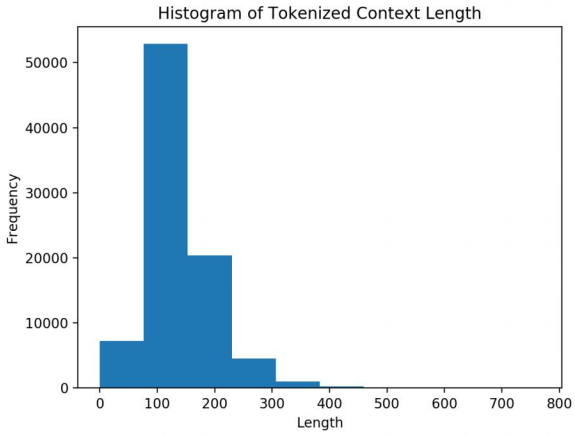


Figure 1: Context Length

2.2 Question Length

We analyze the question length below, where length is defined as the number of tokens in the question. The mean question length is 11.3, with median 11 and standard deviation of 3.71. Plotting the distribution of question length (Figure 2) we see that only a few outliers have question length greater than 30. Thus we keep the hyper parameter question length set to 30 as in the initial baseline model [5].

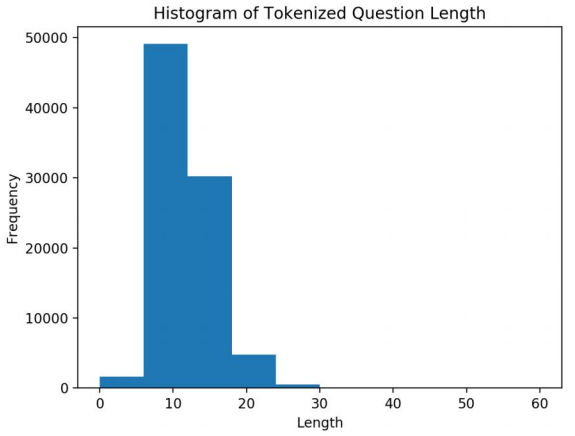


Figure 2: Question Length

2.3 Answer Placement in Context Length

Finding the answer for each question in the context we deduce where the answers to the questions are most likely to be. From Figure 3 we see that answers appear more frequently towards the start of the context. From this and the distribution of context lengths seen before we reduce the hyper parameter max context length to 450.

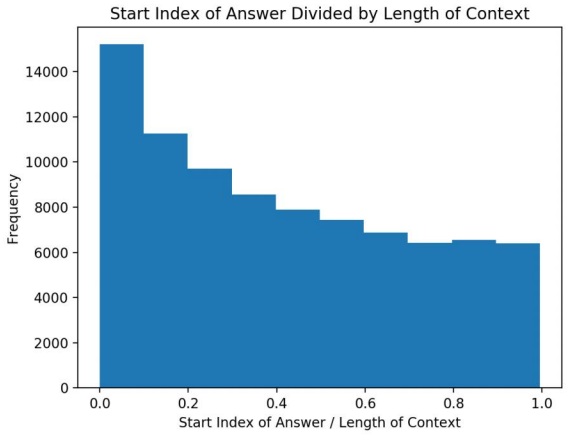


Figure 3: Answer Placement

3 Self-Attention

We turn to the R-Net[1] paper for our first additional layer. R-Net introduces self-attention as an additional layer following our more traditional attention layers (Basic Attention [5], or BiDaf [3]). Self-attention is interesting in itself because it only looks at the contexts, as opposed to our previous attention layer, and allows our full network to understand the relationships between the words in the contexts. However, because it is context to context as opposed to context to query, it is computationally expensive. We introduce the self attention model below.

3.1 The Self-Attention Layer

The self-attention layer takes as input a sequence of representation $v_1, \ldots, v_n \in \mathbb{R}^l$, which we take as our output from the BiDaf layer. Here $l = 8h$, where h is the hidden size. Then we introduce two weight matrices $\boldsymbol{W_1}, \boldsymbol{W_2} \in \mathbb{R}^{q \times l}$ and a weight matrix $\boldsymbol{w} \in \mathbb{R}^q$, where q is an additional hidden size. Then, we define:

$$\begin{aligned}
\boldsymbol{e}_j^i &= \boldsymbol{w}^T \tanh(\boldsymbol{W_1} v_j + \boldsymbol{W_2} v_i) \in \mathbb{R}, \text{ for } i, j = 1 \ldots N \\
\alpha_i &= \text{softmax}(\boldsymbol{e}^i) \in \mathbb{R}^N \\
\boldsymbol{a}^i &= \sum_{j=1}^{N} \alpha_j^i \boldsymbol{w}_j \in \mathbb{R}^l \\
\{h_1, \ldots, h_N\} &= \text{biGRU}(\{[v_1; \boldsymbol{a}_1], \ldots, [v_N; \boldsymbol{a}_N]\})
\end{aligned}$$

This layer outputs new representations $h_i, i = 1 \ldots N$ for our context words, which we feed into our following layer.

3.2 Self-Attention specific hyperparameters

Following directions from the R-net paper, we decided to pick $q = 75$ as a good hidden size for the self-attention layer. We chose a bi-directional GRU as opposed to a bi-direction vanilla RNN because they tend to perform better. We also picked the GRU over the LSTM for time complexity. While LSTM's tend to perform marginally better than GRU's, their additional time complexity did not seem worth it for our already slowly training model.

4 BiDaf

We now implement the Attention Flow Layer of the BiDaf modeled as described in [5] and [3], while leaving the rest of the structure of the baseline the same. The motivation for this new attention layer is to allow the attention to flow in two directions, from context to query and also from query to context.

4.1 The Attention Flow Layer

To compute the attention flow layer we begin by computing a similarity matrix S as described in the project handout [5]:

$$S_{ij} = w_{\text{sim}}^T[c_i; q_i; c_i \circ q_i]$$

where w is the weight vector we will train and c_i and q_i represent the question and context hidden states respectively. Next we perform the Context to Query attention step. This step develops a matrix of attended query vectors (the matrix of a_i's) which signify which query words are most important to each context word.

$$\alpha^i = \text{softmax}(S[i,:]) \quad \text{for } i = 1, ..., N$$

$$a_i = \sum_{j=1}^{M} \alpha_j^i q_j \quad \text{for } i = 1, ..., N$$

For the other direction, query to context, we develop which context words are the most similar to each query word which we output in c' below:

$$m_i = \max S_{ij} \quad \text{for } i = 1, ..., N$$

$$\beta = \text{softmax}(m)$$

$$c' = \sum_{i=1}^{N} \beta_i c_i$$

Finally, using the above, we output our bidirectional attention flow layer in b_i

$$b_i = [c_i; a_i; c_i \circ a_i; c_i \circ c'] \quad \text{for } i = 1, ..., N$$

4.2 BiDaf Specific Hyperparameters

To help avoid overfitting, we also used the regularization technique known as dropout [4]. For our BiDaf method we chose to set this hyperparameter to 0.2 as seen in the BiDaf paper [3].

5 Failed Exploits

5.1 Logits prediction modification

The model presented in the baseline creates start and end logits as

$$\text{logits}_i^{\,start} = w^T{}_{start} b_i' + u_{start}$$

$$\text{logits}_i^{\,end} = w^T{}_{end} b_i' + u_{end}$$

where b_i' are the oututs of the final fully connected layer. Now, this model seems to be an over simplification of what we actually want to predict, which is the start and end positions as a tuple. For this reason, we attempted to implement logits as:

$$\text{logits}_{i,j}^{\,start,end} = b_i'^T W_{start,end} b_j' + u_{start,end}$$

This would allow correlations between start and end positions to be captured. After these logits were calculated, a softmax can be taken over the logits and to create a probability ditribution over start and end tuples. The loss would then be updated to:

$$-\log p^{start,end}(i_{start,end})$$

And predictions would then be taken as argmax $p^{start,end}$. This would allow us to find the best start and end tuple. This idea proved difficult to implement and computationaly expensive so the idea was abandoned. Given more time, it would have been interesting to dive deeper into this method.

5.2 Deep contextualized word representations

The paper in question, [2], was pointed out to us as an additional layer after the vanilla GloVe vectors used in the baseline. We attempted but failed the implementation of this idea, mainly because of times constraints. This paper uses the idea of ElMo (Embeddings from Language Models) as an alternative to traditional embedding vectors. This implementation showed very promising results for SQuAD in [2], and would be a very satisfying next step to succeed at.

6 Final Model

6.1 Architecture

Our final architecture built on the original baseline and made the following changes:

- We changed the RNN encoder from a bidirectional-GRU to a bidirectional-LSTM for improved performance.
- We added an extra Self-Attention layer between the Basic layer and the output layer. See 3.
- We switched the basic attention layer for a BiDaf layer. See 4.

6.2 Hyperparameters

Additionally to the search mentioned in Sections 4.2 and 3.2, we performed hyperparameter search on a subset of the many hyperparameters contained in our final architecture. The notable changes to the baseline were:

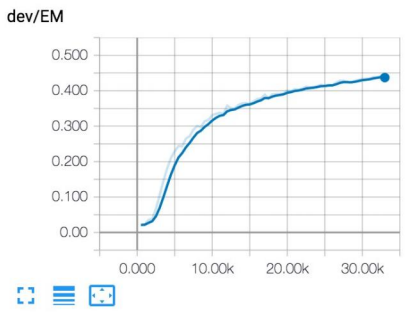
- We converged to a batch size to 20 to deal with "Out of Memory" problems we dealt with following our very memory hungry implementations of BiDaf and self-attention.
- We converged to a learning rate of $\alpha = 5 \times 10^{-4}$ to adjust to our decreasing batch size and slowly training model.
- Following our discussion in Section 2, we converged to a context length window of $N = 450$.

7 Results

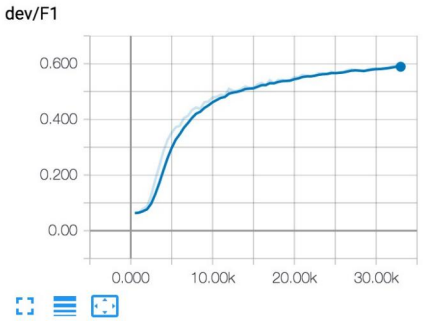
In this section, we present the results achieved by our final model described in 6 as well as our intermediate model that used self attention but did not have Bidaf.

7.1 Self Attention

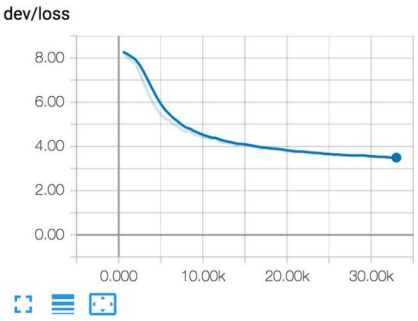
Presented below are the training plots from the self attention model:



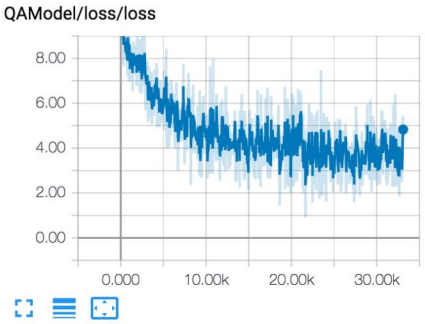
(a) Dev Em



(b) Dev F1



(c) Dev Loss



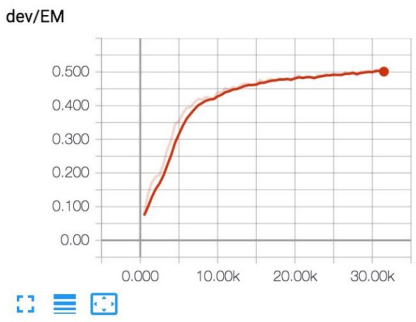
(d) Model Loss

The self attention model was able to perform significantly better than the baseline model and achieved the following results:

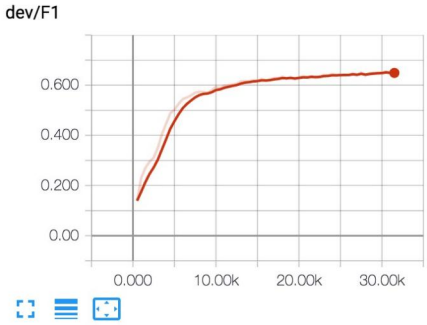
	Dev Set	Test Set
F-1	63.879	64.107
EM	52.062	52.733

7.2 Final Model

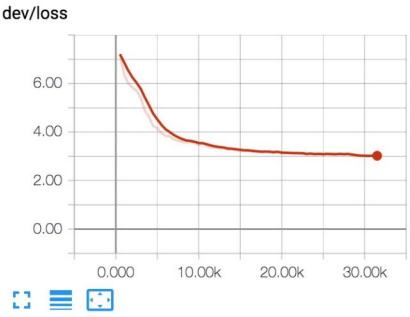
Presented below are the training plots our final model, as described in 6:



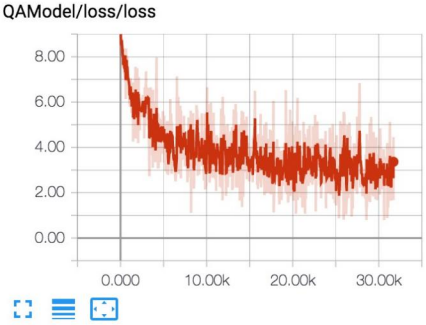
(a) Dev Em



(b) Dev F1



(c) Dev Loss



(d) Model Loss

The final model was able to achieve the best results as illustrated below:

	Test Set
F-1	70.83
EM	60.59

Clearly adding the additional BiDaf layer helped the model generate superior results.

8 References

[1] Microsoft Research Asia Natural Language Computing Group. R-NET: Machine Reading Comprehension With Self-Matching Networks. May 2017.

[2] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. *ArXiv e-prints*, February 2018.

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

[4] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[5] CS224N Course Staff. CS 224N Default Final Project: Question Answering. 2018.