The SQuAD Challenge - results of several experiments

Rajeeva Gaur
SUNet ID: rajeevag

Satyam Kotikalapudi
SUNet ID: satya592

March 22, 2018

Abstract

Given SQuAD (Stanford Quesiton Answering Dataset) The goal of the project is to provide improved F_1 and EM results over the baseline result. In this paper we report the results of several experiments we conducted by turning features on/off. At the end we combined some features to get a better result compared to the baseline result.

1 Introduction

In a question-answering system given a paragraph of texts and a question about the paragraph as inputs question-answering system returns the correct answer. In general, question answering systems are very complex and are considered an improvements over the traditional search engines because they provide precise occurrences of answers in documents.

In this project we are given SQuAD (Stanford Question Answering Dataset). We are provided with a baseline implementation of RNN encoder layer, attention layer, and an output layer. The focus of our project is to replace model in the attention layer, and tune hyper-parameters, use different optimization algorithms, improve on answer spans i.e. smarter spans contributing towards improving upon the baseline result.

The organization of this report is as follows: In *Section2* we present related work. In *Section 3* we describe the criteria for selecting the model and hyper-parameters. We describe the model's implementation, and additional implementations, and tuning in *Section 4*. *Section 4* has been divided into several *Sub-sections*. In *Section 5* we present our results. In *Section 6* we conclude and highlight our result. In *Section 7* we discuss future work.

2 Related work

In recent years, natural language processing in general and a question-answer system in particular has become a hot topic for research. Researchers at Stanford [3], Facebook [4], and IBM [5] and many other universities and industries are actively working on it. Our project is based on SQuAD. SQuAD is less than two years old and already has many researchers working on improving the accuracy of system.

3 Criteria for models' and parameters' selection

We have two criteria for evaluation: F_1 score and "Exact Match(EM)" score. F_1 score is defined as follows:

$$F_1 = \frac{2pr}{p+r}, \quad \text{where } p \text{ is precision and } r \text{ is recall} \tag{1}$$

EM is a binary measure, i.e. if the system's output matches the ground truth exactly then the score is 1, otherwise the score is 0. The leader board prioritizes on improving the F_1 score over EM score. Note that EM score is a stricter measure.

4 Models' selection and implementation details

4.1 Determining maximum context and question lengths

We created graphs(Figure 1) for contexts and answers and analyzed the graphs we determined that the default length for contexts and questions provided in the code is reasonable.

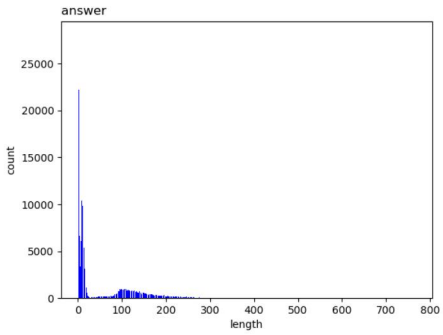
4.2 *Attention* models

Frist we chose to implement *BiDAF*(Seo et al. [1] Attention Model. In our implementation we precisely followed the steps given in the project document. However, due to small amount of available memory, the code ran out of memory. Therefore, we split the *Similarity matrix* in *BiDAF* into three components: w_{sim_c} for context hidden states, w_{sim_q} for question hidden states, and $w_{sim_{coq}}$ for element-wise product. Furthermore, we reduced the batch size to 50 to resolve the memory issues.

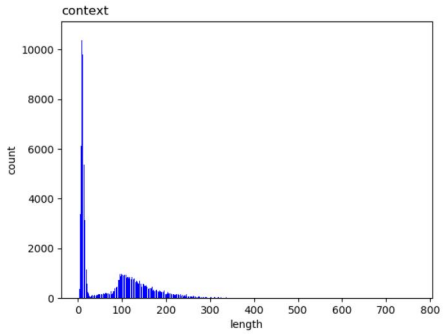
Later we implemented *Dynamic Coattention Network* Attention Model and tried adding additional layers to both *BiDAF Dynamic Coattention*

Figure 1: Data analysis.

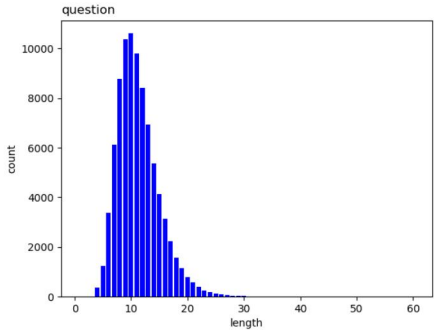
(a) Answers



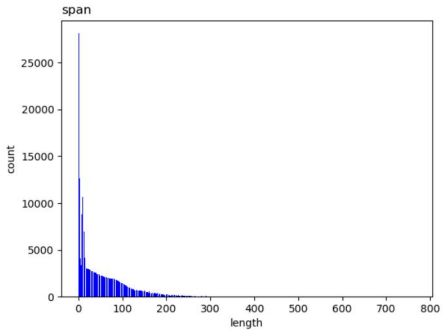
(b) Contexts.



(c) Questions.



(d) Spans frequency.



Network. Also tried to combine them using weighted average, but could not complete the training due to long iteration periods.

4.3 Hyper-parameter tuning

We experimented with a few learning rates. We found that starting learning rate of 0.001 is reasonable. However, we made the learning rate decay exponentially with increasing epoch. Our experiment suggests that the following exponentially decaying learning rate is a good choice for this project:

$$\text{learning rate} = 0.001 * e^{-0.2*i}, \text{where i is epoch number} \tag{2}$$

4.4 Regularization

We considered the following dropouts 0.25, 0.2, 0.18, and 0.15. Based on our experimental results, we chose dropout to be 0.2 as a reasonable choice. We also used L2 regularization; however, the results were below par.

4.5 Smarter span selection at test time

We first implemented and experimented with span selection as given Chen et al. [1]. However, the runtime was significantly slower. Furthermore the hyper-parameter "offset" ($i <= j <= i+$ offset) required significant tuning time. As a result we implemented a greedy method as follows: For batches where end-position is smaller than start-position, we compute temp-start-position = np.argmax(start-dist[:end-position]) and temp-end-position = np.argmax(end-dist[start-position:]). Between pairs <temp-start-position, end-position> and <start-position, temp-end-position> we chose the pair that gives larger product-probability (some corner cases such as end-position being 0 and start-position being the last index needs to be taken care of).

Due to time to run, We first determined the best model and hyper-parameters and then applied smarter span selection to it (it is possible that other model and/or hyper-parameters may have performed better, but due to time constraint we did not experiment).

4.6 Model size and number of layers

We increased the number of hidden layers by one and in a separate experiment increased the hidden size to 300 we used different dropout rates for *BiDAf*. However, it took significantly longer to run without improvements in the result.

4.7 Optimization algorithms

The baseline code has included Adam Optimizer. In addition to Adam Optimizer, we also used Stochastic/Batch Gradient Descent in combination with exponentially decaying learning rate. The intuition is that as the algorithm approaches the optimal point a smaller learning rate is preferred to prevent oscillations. However, The runtime was extremely large and the results for up to 5000 iterations and improvements were very slow. So, we did not proceed further.

5 Results

5.1 Regularization

We present the results in the table form. First we present the result of regularizaton using the *Attention* model in the base line code.

Table 1: Selecting Regularization (*Dev* Results)

	L2	dropout 0.25	dropout 0.2	dropout 0.18	dropout 0.15
F_1	0.45	0.39	0.455	0.38	0.39
EM	0.33	0.28	0.336	0.28	0.28
At iterations	16000	17000	20000	26000	16000

The result presented in Table 1 is the best result we achieved after observing the trend for 3000 additional iterations before stopping the program. Due to the nature of the problem, it is likely that the results would have improved if allowed for more iterations. However, due to time constraints, we had to stop the experiments. For the results look into Table 1 Adam's Optimizer with a fixed learning rate of 0.001 were used.

5.2 Model size and number of layers

By adding one more hidden layer and by increasing the hidden size to 300 along with dropout rate set to 0.25, the run time was significantly higher and results were not at par. So, we discontinued the experiments after 5000 iterations. The number of iterations are small; however, we had limitations on time so we looked at the trend to make the decision.

5.3 Optimization algorithm selection

The convergence with *SGD* were very slow and appeared to tapper off at 10000 iterations. So, we didn't proceed after 10000 iterations. We used *SGD* with exponentially decaying learning rate to prevent any oscillations as number of *epochs* increase. The formula we used for exponentially decaying learning rate is given in *Section* 4.3.

5.4 *Attention* model selection

We started with *BiDAF* and compared it with the base *attention* model. Table 2 summarizes the result.

Table 2: Selecting *Attention* model (*Dev* Results)

	Base Model	*BiDAF* model	*DCN model*	*DCN* + addl. Layer
F_1	0.442	0.333	0.309	0.236
EM	0.348	0.29	0.22	0.164
At iterations	15000	18000	18000	4500

Later we implemented *Dynamic Coattention Network* and compared it with the base *attention* model. We implemented weighted *BiDAF* and *Dynamic Coattention Network* model, training took 40sec per iteration and could not complete the training.

Our currently submitted code to the leadership board has *Smart span selection at test time* and produces *Dev* scores: $F_1 = 0.494$ and $EM = 0.397$ taken at iteration 19000 with a batch size 50. We are working on improving the result.

6 Conclusion

We tried several features with on/off and the best result currently we have from *BiDAF* with *Smart Span Selection At Test Time*, using *Adam's Optimizer* with decaying learning rate. Our best result on Leader Board is: $F_1 = 0.494$ and $EM = 0.397$ at 19000 iterations with batch size 50. We are working on improving the result.

7 Limitations and Future work

Due to limited time and compute resources, we were able to run some experiments with features on/off. Furthermore, if the per iteration run time of an experiment was high, we terminated the experiment. We also terminated experiments if we noticed that there were not enough progress in three consecutive reporting of the F_1 and EM scores on Dev. However, we have noticed that F_1 and EM were not monotonically decreasing as a result it is likely that we may have stopped some experiments too soon.

With more compute resources and time we would like to study into the experiments which we may have terminated too soon. We also want to implement more *Attention* methods. In addition to feature on/off study we would like to combine more features together to see if the results improve. We could use feature selection algorithm which is a greedy algorithm, we give the pseudo-code for feature selection algorithm as follows:
selected-feature-list = ϕ
All-features
For each iteration do:

if All-feature is ϕ break the for-loop

Evaluate each feature $\in$ All-features, choose feature f with the best result

If the improvement from f is not significant break the for-loop;

selected-feature-list += f

All-feature -= f
End for-loop

References

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

[2] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. arXiv preprint arXiv:1704.00051, 2017.

[3] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. arXiv preprint arXiv:1611.01604, 2016.

[4] A. M. Rush, S. Chopra, and J. Weston, A neural attention model for abstractive sentence summarization, CoRR, vol. abs/1509.00685, 2015. [Online]. Available: http://arxiv.org/abs/1509.00685

[5] R. Nallapati, B. Xiang, and B. Zhou, Sequence-to-sequence rnns for text summarization, CoRR, vol. abs/1602.06023, 2016. [Online]. Available: http://arxiv.org/abs/1602.06023