Reading Comprehension with SQuAD Dataset

Wei Kang
Codalab username: weikang
weikang@stanford.edu
CS224n Assignment #4, Winter 2018

Abstract

This paper summarizes the default final project for CS224n, Winter 2018. I implemented the Bi-Directional Attention Flow model[2] with some changes to solve the reading comprehension problem with Stanford SQuAD[1] dataset. With my implementation, a single model achieve **76.124%** in F1 score and **66.296%** in EM score with the test set.

1 Introduction

Reading comprehension by machine is one of the most challenging tasks in Machine Learning and Natural Language Understanding. In such tasks, a machine is given a context and a query, and is required to predict the answer from the context. This problem has gained a lot of popularity and attention in recently years because of the trend with neural network and deep learning. Deep learning network makes it possible to model the interaction between context and query without a lot of feature engineering work. **S**tanford **Qu**estion **A**nswering **D**ataset (SQuAD)[1] is a new reading comprehension dataset that features a public leaderboard for all the submitted models. More details about this dataset can be found here. In this paper, I present the model I built based on the BiDAF[2] paper, and its experimentation and result with SQuAD dataset. At the end of the paper, I also did the error analysis of my model and how future work can be done to improve it.

2 SQuAD Data Analysis

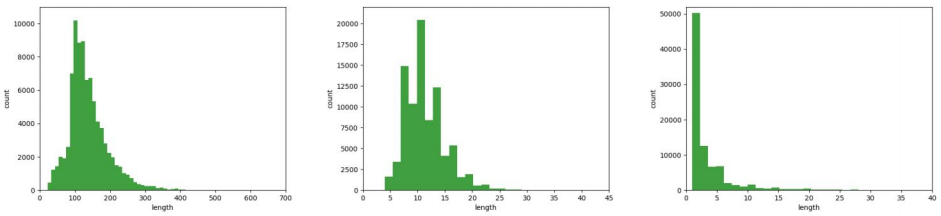
The SQuAD dataset was built from Wikipedia articles, and it contains more than 100,000 entries. Understanding the dataset used in this problem is very important before designing and optimizing models. Often times it can give us ideas regarding how to improve the model score based on the data pattern. Here, I plot the sequence length distribution for the contexts, questions and answers in the training data. As shown in the histograms in Figure 1, vast majority of the context length is less then 400, question length less than 30, and answer length less than 12. These are hyper-parameters we set in the modeling code to help the model performance and training efficiency.

3 Bi-Directional Attention Flow Model

In this section, I present the general architecture of the model I used in the final project, followed by details of different layers of this deep neural network model, and how they interact with each other.

3.1 Model Layers

The overall model architecture is shown in the diagram in Figure 2. There're mainly four layers in the model. The purpose of each layer is described in detail below.



(a) Context length (b) Question length (c) Answer length

Figure 1: Context/Question/Answer length distribution

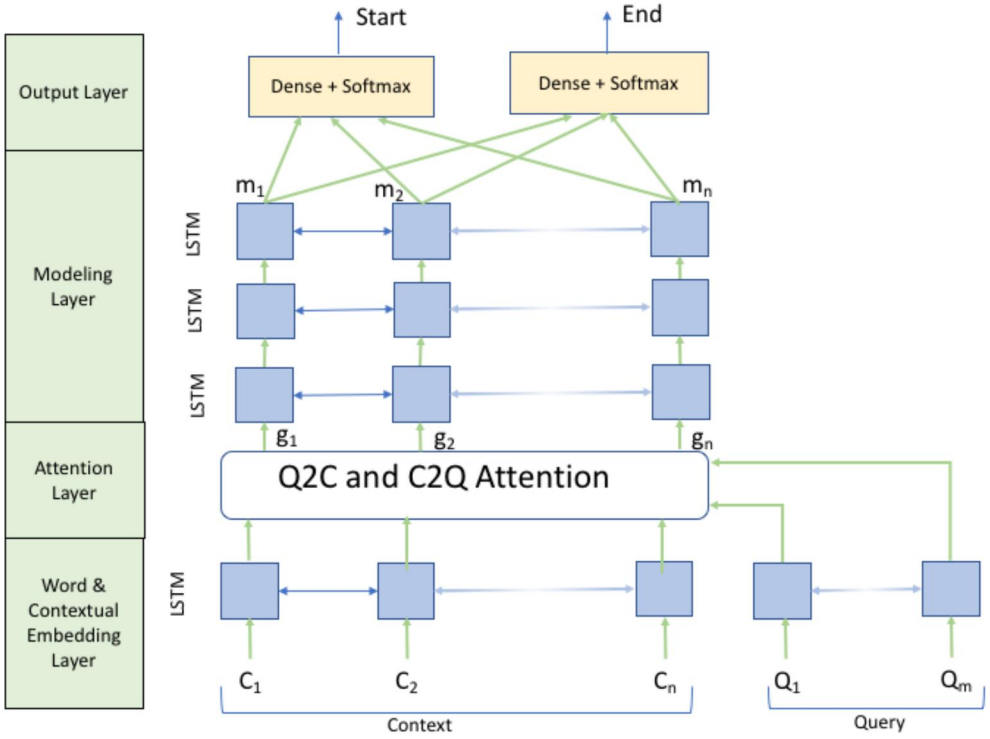


Figure 2: BiDAF model architecture

Embedding Layer: This is the layer where context and query words are converted into embeddings. We use GloVe[3] word embedding. Embeddings vectors with dimensionality of 100 are used in my system implementation. The size of the vocabulary used for the embeddings is around 400,000. From my experience, quite some words from the SQuAD dataset are still missing in this vocabulary. The word embedding for contexts and questions are passed to a LSTM network to form the contextual embedding. The output hidden states from the LSTM are the encoded representations for the contexts and questions.

In addition to the GloVe embedding vector, I also added an additional "exact match" feature in the context word embedding vector if the context word also appears in the query word. This is inspired by the features used in the Dr QA paper[4]. This small feature turns out to be quite useful.

Bi-directional Attention Layer: The mechanism of the Bi-directional attention layer is exactly same as the original BiDAF paper. A similarity matrix $\mathbf{S} \in \mathbb{R}^{(N \times M)}$ is used and the element of it is defined as

$$\mathbf{S_{ij}} = \mathbf{w_{(S)}^T}[\mathbf{h}; \mathbf{u}; \mathbf{h} \circ \mathbf{u}]$$

where $\mathbf{h}$ and $\mathbf{u}$ are context and question hidden states, $\mathbf{w}_{(\mathbf{S})} \in \mathbb{R}^6$ is a trainable weight vector, $\mathbf{h} \circ \mathbf{u}$ is element-wise multiplication, $[;]$ is vector concatenation.

Context-to-query attention is defined as weighted sum of question hidden states based on row-wise attention distribution of $\mathbf{S}$:

$$\alpha^i = softmax(\mathbf{S_i}, :) \in \mathbb{R}^M \quad \forall i \in \{1, ..., N\} \tag{1}$$

$$\boldsymbol{a_i} = \sum_{j=1}^{M} \alpha_j^i \mathbf{q_j} \in \mathbb{R}^{2h} \quad \forall i \in \{1, ..., N\} \tag{2}$$

Query-to-context attention is defined as weighted sum of context hidden states based on the attention distribution over context locations:

$$\boldsymbol{m}_i = \max_j \mathbf{S_{ij}} \in \mathbb{R} \quad \forall i \in \{1, ..., N\} \tag{3}$$

$$\beta = softmax(\boldsymbol{m}) \in \mathbb{R}^N \tag{4}$$

$$\boldsymbol{c'} = \sum_{i=1}^{N} \beta_i \mathbf{c_i} \in \mathbb{R}^{2h} \tag{5}$$

The final output from the modeling layer is denotated as $\boldsymbol{G}$ with $\boldsymbol{G_i} = [\boldsymbol{c_i}; \boldsymbol{a_i}; \boldsymbol{c_i} \circ \boldsymbol{a_i}; \boldsymbol{c_i} \circ \boldsymbol{c'}] \in \mathbb{R}^{8h} \quad \forall i \in \{1, ..., N\}$

Modeling Layer: Three layers of LSTM are used for the modeling layer, which is a bit different from the original BiDAF paper. The input of the modeling layer is $\boldsymbol{G}$. The output of the modeling layer is denotated as $\boldsymbol{M}$ with $\boldsymbol{M_i} \in \mathbb{R}^{2h} \quad \forall i \in \{1, ..., N\}$

Output Layer: The output layer is used to predict the start and end points of the answer span from the context. Here I simply use the a downprojecting linear layer to map the vector to a singular value for all context points. After that, a softmax is performed on these singular values to compute the probability values of all the context points.

4 Training and Experimentation

Tensorflow is used to implement the BiDAF model. After the model implementation, the majority of the time and efforts are spent on the model optimization, searching and tuning of various hyper-parameters. Adam optimizer is used in the final implementation. Due to the time limits and lengthy model training time, I didn't get the chance to try out other optimizers to see if that can change the training speed or results. This can be done in my future work. I also tried out GRU instead of LSTM in the model, it seems LSTM gives us a little bit better result. So I stick with LSTM in the final implementation.

4.1 Hyperparameters

During the model improvement process, I tried out the tuning of following hyper-parameters (the parameter values I eventually decided to use are also specified in the table).

Table 1: Hyper-parameters and optimal values

Hyper-Param	Purpose	Chosen value
Context length	Sequence length of the context paragraph	400
Learning rate	Learning rate of the optimizer	initial value 0.001
LR annealing	Learning rate annealing	decaying to 0.96 for every 1000 steps
Dropout	Dropout rate for the neural network	Set to 0.25
Embedding size	Word Embedding size	100
Hidden size	Hidden size of LSTM cell	100

Not all hyper-parameters tuning are successful as expected. In many cases, the model gives same performance and learning result after the parameter adjustment. Such parameters includes "embedding size", "hidden size", etc. Because of these, we set these hyper-parameters values as small as possible to speed up training process.

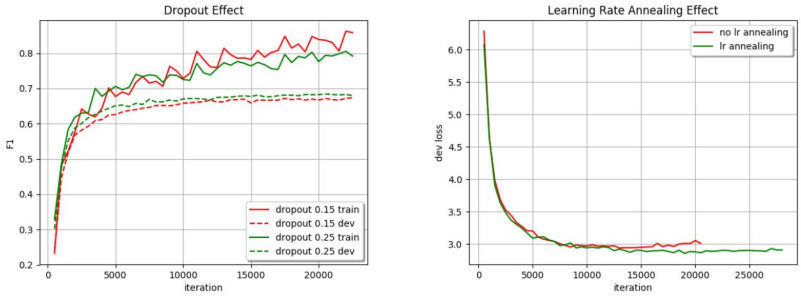
Hyper-parameter searching for the following did give me better performance results if proper values are sets.

Learning Rate Annealing: Learning rate annealing or decay applies exponential decay to the learning rate during the learning process. It allows us to have big learning rate at the very beginning so that we can have faster learning speed at the very beginning. What's more important, it also allows us to have a much smaller learning rate at late stage of the model training, as illustrated in Figure 3(b). This can help us to avoid the "ping-pong" problem at the end, and enable the model to still approach to the optima close to the end of valley.

In our case, our initial learning rate is 0.001, with the decay rate set at 0.96 with every 1000 steps,

Context Length: Context length tuning is also very important to the running time of our model training. Longer context length requires larger memory, and it slows down learning speed. By examining the data at the very beginning, it gives me a rough idea regarding the distribution of the context length and I can safely set a limit for context length without sacrificing the model quality. In this case, I set the context length to 400 based on the histogram distribution of the data set.

Dropout: Dropout rate play a critical to reduce overfitting in neural network. As we can see from the Figure 3(a), when the dropout is set at 0.15, the F1 difference between training and dev can be as big as 20 percent. After the dropout is raised to 0.25, the F1 difference between training and dev is reduced to around 12 percent.



(a) Dropout Effect

(b) Learning Rate Annealing Effect

Figure 3

4.2 Memory And Efficiency

In the provided baseline model, the batch size is set to 100. However, because of the huge memory requirements by the similarity matrix computation in BiDAF model, I ran into many OOM (Out Of Memory) issues. For example, for a batch size of 100, and context length 400, and question length 30, and embedding size 100, the matrix that need to be allocated from memory is in the size of $\mathbf{100 \times 400 \times 30 \times 100}$.

Initially, I had to reduce the batch size to run the model, the penalty is that the model learning speed is much slower with smaller batch size. The solution to this problem is that instead of using **tf.tile** to create huge matrix before the subsequent computation, we can take advantage of the Tensorflow **broadcast** functionality to do matrix operation. With **broadcast**, Tensorflow figures out how to apply smaller item to much bigger item one by one, without allocating huge amount of memory beforehand.

5 Model Results and Analysis

My model achieves F1 score 74.845, EM score 64.626 on dev set; and F1 score 76.124, EM score 66.296 on test set. As shown in Table 2, the dev set result is quite close to the original BiDAF paper implementation. The biggest difference between my model and the original BiDAF paper model is that I didn't use character level embedding data in the training. I also adjusted the original BiDAF model in several aspects, such as span optimization, more LSTM layers, more input features, etc. All these are done in an incremental way. Among them, "span optimization" gave me the biggest lift from the base BiDAF model. "Exact match" feature from Dr QA paper[4] also gave a nice lift considering this is actually a quite small input feature.

Table 2: Model results on dev set

Model	Dev F1 Score	Dev EM score
Provided Baseline	43.738	34.674
BiDAF (Ours, no lr annealing, single)	72.52	62.344
BiDAF (Ours, with lr annealing added, single)	73.145	62.857
BiDAF (Ours, with span optimization added, single)	74.42	64.333
BiDAF (Ours, final with "exact match" feature added, single)	**74.845**	**64.626**
BiDAF[2] (reference implementation, single)	77.3	67.7
BiDAF[2] (reference implementation, ensemble)	80.7	72.6

5.1 Attention Distribution

Attention is an important mechanism in NLP to improve accuracy and relevancy nowadays. To illustrate how attention is used to improve the result, I visualize the attention distribution in Figure 4 and Figure 5. As you can see in Figure 4 for the "Query to Context" attention value distribution, highlighted areas normally means query words also appears in context.

For the same two cases in Figure 4, Figure 5 gives another illustration on the probability distribution of attention, start position and end position after softmax is applied. Since the question contains "sir pindar's house", the same words in the context get high attention probability. And the model is able to locate "c1600" as the correct answer although "c1600" isn't part of the vocabulary.

Sometimes, however, the model is not clever enough to capture the correct attention. For example, in Figure 5(b), the model is doing a good job of capturing all the attentions for "steam" related terms, and its most attention was put on "steam turbine device was described by.." since the exact term "steam turbine" also appears in the question. However, the model fails to capture the attention for "1629" (which is very close to the correct answer, and also appears in the query). Because of this, the model made a wrong prediction.

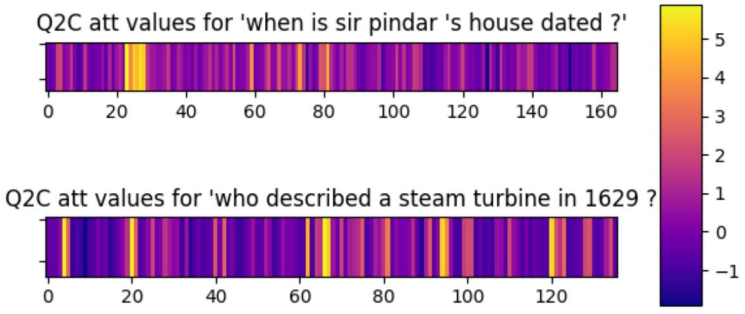
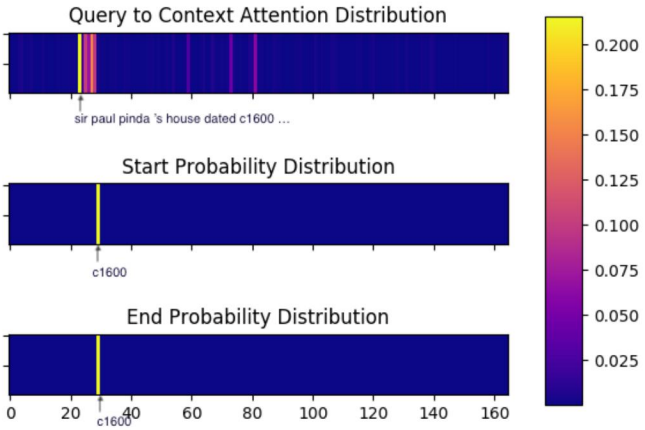
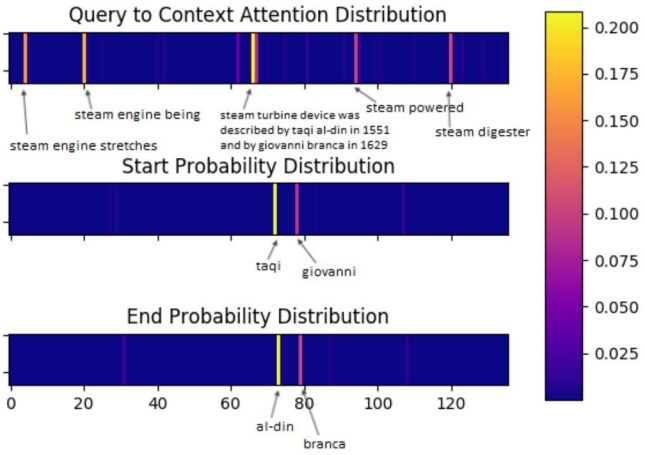


Figure 4: Attention values illustration



(a) Probability distribution for the question "when is sir pindar 's house dated ?"



(b) Probability distribution for the question "who described a steam turbine in 1629 ?" (Failed to capture attention for "1629")

Figure 5: Probability distribution after softmax

5.2 Error analysis

After the model is trained, I performed the error analysis with the prediction result on the dev set. There're many kinds of imperfect predictions made by the modle, and below are some typical prediction errors.

Semantic meaning ambuigity: Sometimes there's some subtleties in the semantic meaning between difference entities in the context. For the following example, some people would although think "northern mokotw" is the correct answer because "where" seems to be decorating it. However, actually both "where" and "northern mokotw" are decorating "pole _mokotowskie_". For this kind of situation, introduction of syntactic structure into the model can help alleviate the problem.

Context: besides , within the city borders , there are also : pole _mokotowskie_ (a big park in the northern mokotw , where was the first horse racetrack and then the airport) , park ujazdowski (close to the sejm and john lennon street)
Question: where was the first horse racetrack located ?
True Answer: pole mokotowskie
Predicted Answer: northern mokotw

Emphasize too much on close words: For the following example, the model seems to emphasize on "semptember 30, 1960" because it is very close to "abc", while "1960s" is several steps aways from "family-oriented series" and more faraway from "abc". If such situation happens quite often, it is worthwhile to add more weight when both critical terms (in this case "abc" and "family-oriented series") appears in the same sentence.

Context: the 1960s would be marked by the rise of family-oriented series in an attempt by abc to counterprogram its established competitors , but the decade was also marked by the network 's gradual transition to color . on september 30 , 1960 , abc premiered the flintstones , another example of counterprogramming ;
Question: when did abc begin making family-oriented series ?
True Answer: 1960s
Predicted Answer: september 30 , 1960

Attention on the wrong words: In this example, the model makes prediction for "continental edison company in france" because it is so close to term "begin working", which also appears in the question. So the attention is much bigger here. However, based on our human understanding, it is obviously wrong because this timeframe is not in 1984 at all. Model should be taught to honor the time sequence in terms of event happening.

Context: in 1882 , tesla began working for the continental edison company in france , designing and making improvements to electrical equipment . in june 1884 , he relocated to new york _city:57?60_ where he was hired by thomas edison to work at his edison machine works on manhattan 's lower east side
Question: where did tesla begin working in 1884 ?
True Answer: edison machine works
Predicted Answer: continental edison company in france

Unable to place importance on adjective: In this example, from the syntactic point of view, the model predicted answer is also correct. However, human being obviously care more on "negative", instead of "long-term", which is much more neutral an adjective. So during model building, we can also add some features regarding the sentiment or severity of the word.

Context: the report claimed that these noise levels would have a negative long-term impact on the health of the city 's residents .
Question: what type of impact can the residents of newcastle expect the city 's noise to have on them ?
True Answer: negative
Predicted Answer: long-term

5.3 Error in Dataset

In the dev set, there're also some labeling errors. For example in the following case, the model actually makes a correct prediction. However, due to the obvious error in the labeled data set, the correct prediction is marked as "wrong".

Context: the american automobile association reported that in the last week of february 1974 , 20 % of american gasoline stations had no fuel .
Question: according to the aaa , what is the percentage of the gas stations that ran out of gasoline ?
True Answer: last week of february 1974 ,
Predicted Answer: 20 %

6 Conclusion and Future Work

For this default final project assignment, I reimplemented the BiDAF algorithm in Tensorflow and made some adjustments from the original paper. The model implementation achieves good result on the class leaderboard. BiDAF is a very powerful attention mechanism in tackling the reading comprehension problem. By integrating BiDAF into the class-provided baseline model, I instantly achieved more than 20+% in F1 score. This shows the BiDAF is able to capture the complex word interaction between contexts and questions. However, as I show in the "Error Analysis" section, there're still some cases where proper attention isn't captured.

Due to the time limits of this default final project, not all of the ideas from the original BiDAF paper were implemented. For example, I didn't implement the character CNN mentioned in the original paper. Since quite some words are not covered by the GLoVE vocabulary, I expect character CNN can improve the final result if implemented. Some other hyper-parameter search, such as optimizer, can be done to see if it can affect the training and reuslt. Also the teaching staff has provided a lot of other advanced attention-related papers, and I didn't get a chance to read many of them. I'd expect an ensemble model integrating several attention mechanism can boost the final result.

Acknowlegements

I would like to thank Richard Socher and all the class TAs for a great class and project experience. And I appreciate Microsoft for providing Azure GPU environment for us to experiment on powerful GPU VMs.

References

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

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

[3] Jeffrey Pennington, Richard Socher, and Christopher D Manning. GLoVe: Global vectors for word representation.

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