Invertigation of BiDAF and implementation of QANet for Question Answering

Stanford CS224N Default Project

Jiefu Deng
Department of Computer Science
Stanford University
`harryd1@stanford.edu`

Abstract

In this project, I build two question answering system that have relatively good performance on SQuAD 2.0 dataset. The baseline model is Bi-Directional Attention Flow (BiDAF), which achieved 59.21 F1, 55.92 EM and 65.85 AvNA on Dev dataset. Firstly I implement a CNN-based character embedding to it which achieved 60.192 EM, 63.480 F1 and 69.89 AvNA on Dev dataset. Then I re-implement QANet with Pytorch which is basically the same as the original paper proposed one. It achieved 59.973 EM, 63.403 F1 and 68.12 AvNA on Dev dataset, which is less than the first one. Ultimately, I got 59.307 EM and 62.761 F1 on test set.

1 Key Information to include

- Mentor: N/A
- External Collaborators (if you have any): N/A
- Sharing project: N/A

2 Introduction

Machine Comprehension, given a context and question, aims to decide the answer location in context or no answer added in SQuAD 2.0[1]. The existing models have achieved great score in SQuAD 1.1[2], while more challengable thing is not only correctly answer the questions, but determine whether there is an answer or not in the given context. The BiDAF model[3] achieved EM 67.7 and F1 77.3 on SQuAD 1.1. However, used as basline model, BiDAF only achieved EM 55.9 and F1 59.2 after 30 epoch in my VM. It's about 25% percent of reduction. One obvious weakness of BiDAF is that its recurrent structure can not be parallel computed, making training process quite slow. There had been proposed methods to avoid using RNNs[4]. But it sacrifice performance with around 1.1 lower than BiDAF.

The QANet[5] model solved this problem by replacing RNN with convolutions layer and self-attentions layer, showed in Figure 5 This architect has the ability to learn relationship of each word in context and have a look at surranding words at the same time, rather than learning sequtial represetation using RNN model. And then QANet introduces convolutional network which can take advantage of GPU parallel computation. It is reported to achieve SOTA in 2017 with 84.6 F1 score.

In this project, I further investigate the character embedding using baseline model BiDAF. Then I re-implement QANet using Pytorch and test it on SQuAD 2.0. But the speed of my QANet is not as expected as the original paper reported "13x faster in training and 4x to 9x faster in inference". I can not resolve this issue, but I did some experiments about it which will be discussed in later section.

Machine comprehension and question answering (QA) have became important tasks in NLP area in the past few years. My project is mainly focus on QA on SquAD 2.0[1]. As I learned in lectures and Assignment 5, character-level embeddings perform better than word embedding on out-of-vocabulary words. Inspired by [6], I implement a convolution neural network which can convert inputs character to their CNN-based output. CNN is designed to automatically and adaptively learn spatial hierarchies of features through backpropagation by using multiple building blocks, such as convolution layers, pooling layers, and fully connected layers. I can use it to extract character-level representation of a given word and concatenate it with traditional word embedding for better represetation.

Since the QANet model performs best in 2017 by replacing RNN with convolutions layer and self-attentions layer. Another important part of my project is to re-implement QANet and have a better understanding of it.

3 Related Work

Question answering is one of acitve research area in NLP in the past few years. There have been various RNN based deep learning architectures researched and applied. Most of them consist of an encoder-decoder architecture, which is one of the most popular factors for recent success.

Nowadays, the most successful model is transformer based model. Such as BERT[7] and its variants. These pre-trained model shows great performance on a wide range of NLP tasks, including question answering. Due to handout limitation, I'm not going to use it. Before BERT appears, BiDAF is one of SOTA model on SQuAD dataset. It uses bi-direction LSTM to capture forward and backward information from context word vectors. Different from BiDAF, QANet uses self-attention with position embedding to capture context information. It get rid of RNN architectrue in order to be computed in parallel. The author reported QANet is faster and achieved higher scores on SQuAD dataset. As their training time is less, they can adjust more hyperparameters and use data augmentation to improve variety of training data. The author also said by introducing convolution layer and self-attention layer improved F1 by +2.7/1.3 respectively. And data augmentation gives another +1.1 F1 at most.

4 Approach

4.1 CNN Based Character-Embedding Overview

As proposed in BiDAF original parper[3], I add character-level embedding into embedding layer. I referd to Pytorch official tutorial([8]) to accomplish a CNN which can convert input character to CNN-based embeddings. The CNN model just contains one convolutional layer and one max-pool layer. And I concatenate the original 300 dimention word embedding with 200 dimension character embedding to form the combined embedding. After this operation, the embedding layer uses a lookup method to convert context and questions into embedding.

4.2 QANet Model Overview

Specifically speaking, it has 5 layers.
Input Embedding layer: It has pre-trained, 300 dimentinal word embedding from GloVe [9]. The character is represented as 200 dimensinal trainable vector. The output of a given word vector x is the concatenation of $[x_w : x_c] \in R^{500}$. Then I pass this vector to a highway network.

Embedding Encoder Layer: The encoder layer uses convolution-layer and self-attention layer to replace RNN-based layer. As illustrated in the right of 5. Here positional encoding formula I used in the layer $PE_{(pos,2i)} = \sin \frac{pos}{1000^{\frac{2i}{d}}}$. Where pos is the position of the word in the context, i refers to each dimention of the vector, d refers to the dimention of vector. Then I refered the [10] to implement the depthwise separable convolution layer. The kernal size is 7, the number of conv layers is 2, the number of multi-head attention is 4. The self-attention layer uses multi-head attention to calculate the interaction between query and key. I referred to (https://github.com/BangLiu/QANet-PyTorch) to implement this layer.

Context-query Attention Layer: This layer is a common layer in question answering model. It uses a similarity martix $S \in R^{n*m}$ computed from context C and query Q. Then apply softmax function to each row of S to get $\bar{S}$. The similarity between context word c and query word q is $f(q, c) = W_0[q, c, q \odot c]$, where W_0 is a trainable parameter. The context-to-query attention A is $A = \bar{S} \cdot Q^T \in R^{n*d}$. And the query-to-context attention B is $B = \bar{S} \cdot \bar{\bar{S}}^T \cdot C^T$.

Model Encoder Layer: The model encoder layer consist of a attention layer whose head number is 4. And 2 depth seperable conv layer[10] connecting with a feedforward layer. Each model encoder layer takes previous output attention A and B and then output three shared weight matrics M_0, M_1, M_2.

Output Layer: The three matrices are fed into output layer as 5. This layer is to predict the probability of start position and end position in the context. The probability is given by $p^1 = softmax(W_1[M_0; M1])$ and $p^2 = softmax(W_2[M0; M2])$ respectively. The loss function is defined as the average of cross-entropy between the predicted start/end position with the true start/end position.

I was able to carry out the embedding layer with previous character embedding described in section 4.1. The embedding encoder layer is almost the same as the original paper, but only gives 4 heads in attention layer. Because number of 8 heads consumes more time to train and get similar result as 4. For the model encoder layer, I chose the same hyperparameter with the paper. The mask is a little different, I uses True while the starter code uses False.

4.3 Baseline

For the baseline, I take the BiDAF model proposed in [3]. The only difference is that provied start code doesn't have a character embedding layer. A full implement of baseline model can be found here: https://github.com/minggg/squad.

4.4 Implementation

For CNN Based Character-Embedding, I referred to the paper[6]. The character split data is provided by starter code. For QANet I referred to the official tensorflow implimentation (https://github.com/NLPLearn/QANet). As to encoder block I mainly referred to Bangliu (https://github.com/BangLiu/QANet-PyTorch). I tried my best to align with starter code's style.
At first, I would like to do multiple parameter experiments, since it is said QANet is quite faster. However, the implemented model is almost as slow as the baseline. So it's hard to do more experiments on hyperparameters and data augmentation proposed by the original paper. And I tried to scale batch size to see the reason of the model slowness. But it seems that smaller batch size runs as slow as the original size of 64. Other parameters I used is descried in section 4.

5 Experiments

5.1 Data

This project uses modified SQuAD 2.0 dataset.[1] The SQuAD 2.0 dataset posed by crowdworkers on a set of Wikipedia articles and the answer is either a segment of context, or None. The data has been split into three parts train, dev and test. Training set has 129,941 examples, dev data set has 6078 examples and test set has 5915 examples which is added some special hand-labeled examples.

5.2 Evaluation method

First of all, There are three answers provied for each SQuAD question from different crowd worker. The evaluation method is briefly listed below:
EM : Extract Match. If the prediction answered exactly the same as golden answer, the result is 1. Otherwise it returns 0.
F1: It measures the portion of overlap tokens between the predicted answer and the truth answer.
AvNA: it represent the model's prediction exists – a span of text, vs None answer. Though the loss score(NLL) may be low, AvNA equals to 0 tells us the prediction is unreliable.

5.3 Experimental details

I firstly added character embedding concatenate with word embedding into BiDAF. I trained the model for 30 epoch with a learning rate of 0.5, dropout rate 0.2, batch size 64. It takes about 31 hours to complete training process on Azure NC6_Promo.

Then I implement the QANet. In order not to run out of memory, I have to set batch size to 32. I choose the Adam optimizer as original paper proposed during 1000 warm-up steps. During warm-up scheme with an inverse exponential steps, the learning rate will increase from 0.0 to 0.001. And then maintain a constant learning rate for the rest of training process. Exponential moving average is applied on all trainable variables with a decay rate 0.9999. For regularization, I use L2 weight deday where $\lambda = 3 * 10^{-7}$ on all weight parameters.

5.4 Results

Table1 shows the overall result.

	F1	**EM**	AvNA
Dev set: Baseline	59.21	55.92	65.85
Dev set: QANet	59.97	63.40	68.12
Dev set: BiDAF with Char Embedding	**60.19**	**63.48**	**69.89**
Test set: BiDAF with Char Embedding	**59.31**	**62.76**	N/A

Table 1: Result of implemented models.

My best model is BiDAF with CNN-based character embedding, which achieved 60.19 F1, 63.48 EM, 69.89 AvNA on dev set and 59.31 F1, 62.76 EM. My model ranked 23th on IID Squad Track (test leaderboard) at writing time. This model takes 300 dimentional word embedding concatenate with 200 dimentional character embedding, batch size 64 and 4 attention heads. It takes around 40 minutes per epoch and 31 hours to train in total. The evaluation scores are showed in Figure [**?**]. As the figure shows, EM, F1 and AvNA keep relatively at the same level as steps get larger. So 30 epoch is enough for the training process. As comparison, baseline model takes about 30 minutes

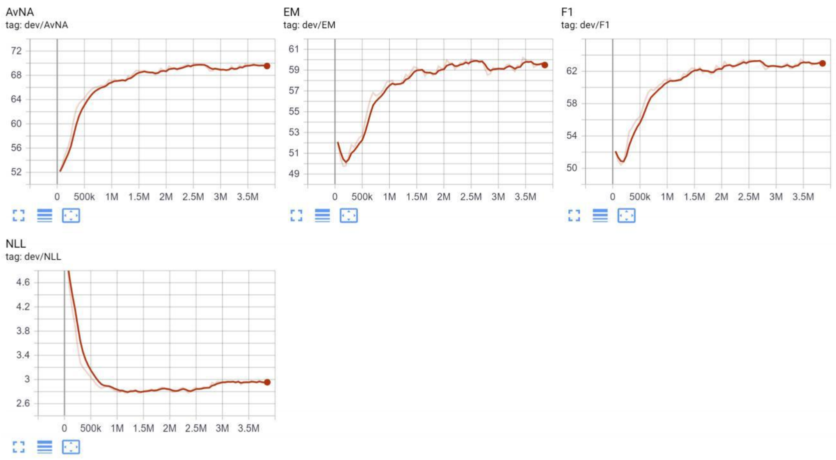


Figure 1: BiDAF with Character Embedding performance on dev set

per epoch to train and get 59.21 F1, 55.92 EM and 65.85 AvNA. It takes about 24 hours to train the whole model on Azure.
Then, my implimented QANet model takes about 65 minutes per epoch to train, which is unexpected. It takes about 48 hours to complete the training process. It is far from the original paper reported "...our model is 3x to 13x faster in training and 4x to 9x faster in inference"[5]. I was almostly follow every setting from original paper to accomplish the model. It is strange to see that the model speed differentiates so much. So I did an experiment to investigate the slowness of self-implemented QANet. It shows that if I scale the batch size of the model, like 32 to 16, it will keep training speed

for about 65 minutes per epoch. Due to GPU memory limitation, I can not enlarge the barch size with 128. I think that the slowness of self-implemented QANet model is mainly due to resource allocation limitation. In other word, if we use muptiple GPU, it could perform better on speed due to the ability of parallel computation. It is interesting to arrange QANet on multi-GPU to verify the hypothesis. I'm not able to do that, since there is hardward limitation.
As for the performance of my implemented QANet model, it achieved 59.97 F1, 63.40 EM and 68.12 AvNA which is as expectation. In [1], the auther reported "...a strong neural system that gets 86% F1 on SQuAD 1.1 achieves only 66% F1 on SQuAD 2.0." It can be seen from Figure 2, the QANet

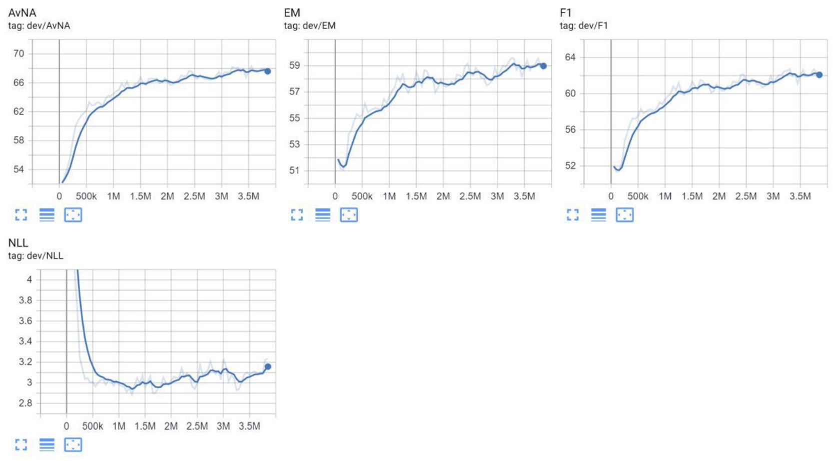
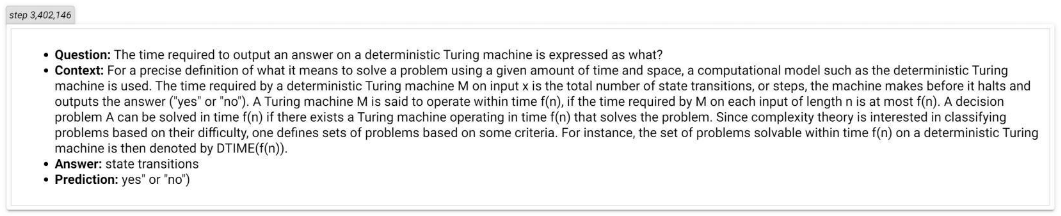


Figure 2: Self-implemented QANet performance on dev set.

model reaches it peak at around 3.3M steps, which is 26 epoch. Then all of the EM, F1, training loss head down until the end of training. I think it is mainly due to model overfitting. It means that even if increase the number of epoch will NOT impove the scores at all. However, it is possible to adjust dropout rate to make model less overfitting. This is also worth investigating in the future.

6 Qualitative Analysis

I randomly pick some questions which my model gave wrong prediction.

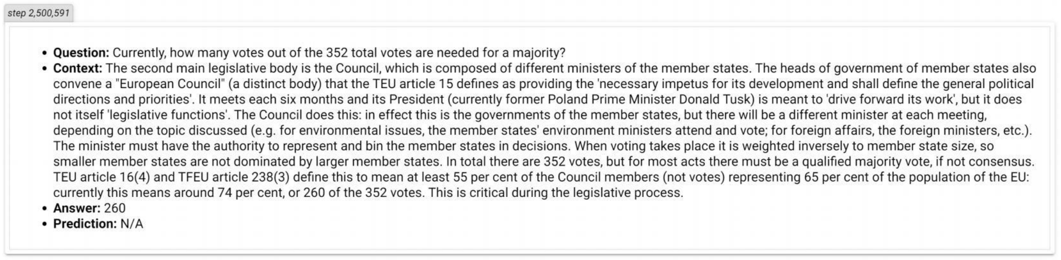


step 3,402,146

- **Question:** The time required to output an answer on a deterministic Turing machine is expressed as what?
- **Context:** For a precise definition of what it means to solve a problem using a given amount of time and space, a computational model such as the deterministic Turing machine is used. The time required by a deterministic Turing machine M on input x is the total number of state transitions, or steps, the machine makes before it halts and outputs the answer ("yes" or "no"). A Turing machine M is said to operate within time f(n), if the time required by M on each input of length n is at most f(n). A decision problem A can be solved in time f(n) if there exists a Turing machine operating in time f(n) that solves the problem. Since complexity theory is interested in classifying problems based on their difficulty, one defines sets of problems based on some criteria. For instance, the set of problems solvable within time f(n) on a deterministic Turing machine is then denoted by DTIME(f(n)).
- **Answer:** state transitions
- **Prediction:** yes" or "no")

Figure 3: Wrong prediction of BiDAF with Character Embedding model.

Comment: This is an error occurred in model BiDAF with character embedding. Perhaps the model made this mistake showed in Figure 3 because it misunderstand the question. The subject of question is "The time ... is expressed as what?". While the model thought it is asking about "output an answer on a deterministic Turing machine". I observed the same error in the baseline model. One possible method to avoid this kind of mistake is to reconstruct the question, like translate the quesion into another language and put it back proposed in [5].



step 2,500,591

- **Question:** Currently, how many votes out of the 352 total votes are needed for a majority?
- **Context:** The second main legislative body is the Council, which is composed of different ministers of the member states. The heads of government of member states also convene a "European Council" (a distinct body) that the TEU article 15 defines as providing the 'necessary impetus for its development and shall define the general political directions and priorities'. It meets each six months and its President (currently former Poland Prime Minister Donald Tusk) is meant to 'drive forward its work', but it does not itself 'legislative functions'. The Council does this: in effect this is the governments of the member states, but there will be a different minister at each meeting, depending on the topic discussed (e.g. for environmental issues, the member states' environment ministers attend and vote; for foreign affairs, the foreign ministers, etc.). The minister must have the authority to represent and bin the member states in decisions. When voting takes place it is weighted inversely to member state size, so smaller member states are not dominated by larger member states. In total there are 352 votes, but for most acts there must be a qualified majority vote, if not consensus. TEU article 16(4) and TFEU article 238(3) define this to mean at least 55 per cent of the Council members (not votes) representing 65 per cent of the population of the EU: currently this means around 74 per cent, or 260 of the 352 votes. This is critical during the legislative process.
- **Answer:** 260
- **Prediction:** N/A

Figure 4: Wrong prediction of QANet model.

Comment: Here is another error occured in QANet model. As showed in Figure 4, my QANet model gave no answer for this question. Because it matches wrong span of context, "In total there are 352 votes...". Apparently there is no answer in that sentence. The correct answer is right after a long, complicated sentence. It means this model is lack of ability to seize relationships between long sequential context with question.

7 Conclusion

Summarize the main findings of your project, and what you have learnt. Highlight your achievements, and note the primary limitations of your work. If you like, you can describe avenues for future work. In this project, 1) I add CNN-based character embedding to BiDAF and the result shows the performance on SQuAD 2.0 dataset increased by +7.56 EM, +0.98 F1 and +4.04 AvNA score. The idea is to capture the most important feature, the one most highest value, from a given vector. And then concatenate the information with word level embedding. The result ranks 23-th on class test leaderboard on March 16th. 2) I re-implement the QANet model as [5]. Its performance on SQuAD 2.0 dataset is NOT as fast and good as expected. I noticed that even decrease the batch size, the model spend roughly the same time per epoch. Because of GPU memory limitation, I'm unable to try larger batch size. I noticed that increase the number of heads in self-attention can increase the model performance. But for slowness of the model, I don't have time to try all kinds of parameters combination as I wanted. For future work, I would like to implement an ensemble of models to get better performance and try to speed up training process by employing serveral GPUs.

References

[1] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018.

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

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

[4] Jonathan Raiman and John Miller. Globally normalized reader. *CoRR*, abs/1709.02828, 2017.

[5] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *CoRR*, abs/1804.09541, 2018.

[6] Xuezhe Ma and Eduard H. Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *CoRR*, abs/1603.01354, 2016.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[8] Define a convolutional neural network. `https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#define-a-convolutional-neural-network`. Accessed: 2021-03-02.

[9] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[10] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

A Appendix

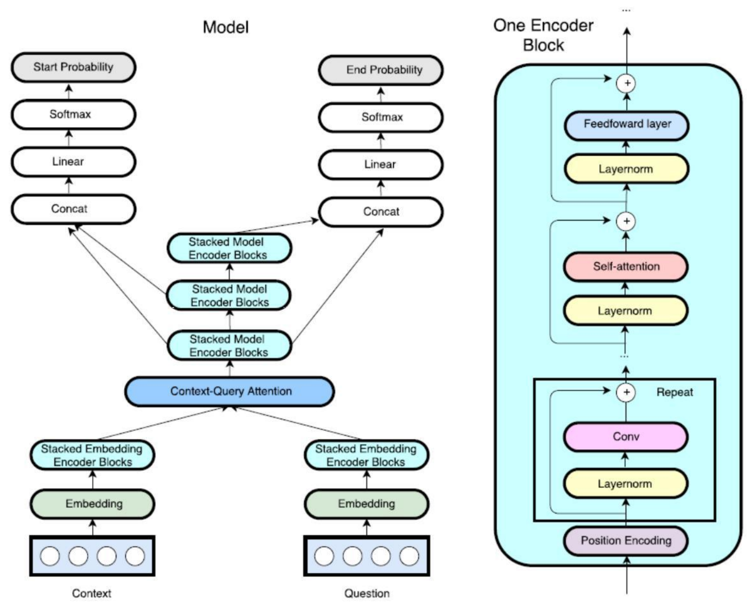


Figure 5: The architecture of QANet (left) with its encoder blocks inner structure(right). Figure copied from [5]