QANet implementation with multiple embedding methods for SQuAD

Stanford CS224N {Default} Project / option 3

Jiachen Xu, Ying Guo
jx2318@stanford.edu, adaguo@stanford.edu

Abstract

Question answering is one of popular research topics in NLP area and plenty of deep learning architectures have been proposed to solve it. This project start from BiDAF and QANet, aiming to experiments the influence and effectiveness of different encoder architectures and embedding methods. The baseline model achieved 68.25 AvNA, 57.95 EM and 61.25 F1 in the dev set. Based on this, we experimented 1. word embedding concatenate character embedding 2. QANet architecture. 3. fast-text embedding as pre-trained word embedding for QANet. 4. ELMo embedding as pre-trained word embedding for QANet. 5. Combined the BERT embedding with QANet. The best performance is from fast-text embedding as pre-trained word embedding for QANet. Ultimately, we got 65.636 EM and 68.853 F1 for dev set and 61.809 EM and 65.1 F1 for test set.

1 Introduction

The tasks of machine comprehension (MC) and question answering (QA) are becoming an essential benchmark for model evaluation for past years in NLP area. In order to achieve promising results, a lot of new deep learning architectures and methods have sprouted onto the market over the last few years. As known, BiDAF and QANet were two state-of-art models before BERT comes out to dominate this area with impressive F1/EM score. In this project, we implemented QANet as in [1], and compared it with our based model BiDAF[2] without char-embedding. Unlike RNN based model BiDAF, QANet only use convolution and self-attention mechanism, which gives a big reduction of training and inference time.

In addition, as using the pre-trained word representation has become basic building blocks in NLP tasks, we introduced multiple embedding methods: character-level embedding[3], Glove[4], FastText[5], ELMo[6] and BERT[7] onto QANet for the purpose of learning impact of different embeddings. Both Glove and FastText are word-embedding methods based on word2vec[8], however FastText extends concept of word2vec which treats each word as composed of character ngrams. Compared to Glove, FastText gives better performance on rare words and “out of vocabulary” words. With inspiration of original BiDAF model, we added character embedding block into the model, which consists of loading pre-trained character-level embedding and passing to the convolution layer, then concatenat the output with the word embedding(Glove and FastText). Compared with word-embedding-only methods, the new char-word embedding combination methods have improvements on EM and F1 score.

Furthermore, we also applied deep contextualized word embeddings: Elmo and BERT onto QANet. Unlike Glove and FastText word embeddings, which are context independent, Elmo and BERT can generate different word embeddings for a word depends on its context and position in a sentence. Due to the complexity of these embedding models, they indeed could bring a decent F1/EM scores for our task, but the trade off is high requirement of computational power, long embedding time and high memory consumption.
2 Related Work

Question answering is one of the most popular research topics in NLP area among these years. Different deep learning architectures have been researched and applied. Among all these models, transformers, consist of the encoder and decoder block, are one of the most important factors for the model.

Bi-Directional Attention Flow (BiDAF) network is one previous state-of-the-art model. Largest innovation of this architecture is that it applied bidirectional-LSTM architecture in the encoder block to capture the contextual information of the word. Except BiDAF, another state-of-art model is QANet. Unlike BiDAF, QANet applied the self-attention accompanying with position embedding to learn the contextual meaning of words. Since it get rid of the RNN architecture, this architecture kind of released the problem of long training and inference time problems.

Inspired from these two architecture and also based on some other researches, ELMo and BERT have been proposed. ELMo embedding is also called 'Deep Contextualised Word Representations'. The encoder block from ELMo is similar to BiDAF, which used deep bi-LSTM blocks to learn the contextual information for the word within the context. And then concatenate the hidden states each outputs bi-LSTM outputs based on the weighted summation.

As for BERT, Bidirectional Encoder Representations from Transformer, is more similar to the QANet architecture, which applied attention instead. The combine embedding from word and position embedding pass to multi-head attentions blocks to learn the contextual information of words. Also BERT pre-trained with the next sentence prediction and masked language model task, which are two genius methods to train the model.

3 Approach

3.1 BiDAF

BiDirectional Attention Flow Model (BiDAF) consists of five layers, which are Embedding layer, Encoder layer, Bidirection Attention layer, modeling layer and Output layer.

To be more specific, in the embedding layer, it will look up the pre-trained word embedding for both context and questions based on the word/char index, and following with a dropout layer and highway layer, which aimed to increase the robustness of the model and refine the embedding representation. At the encoder layer, the output from embedding blocks will be padded to a fixed length vector and then passed through a bi-directional LSTM layer to learn word representation based on both left and right side of text. The bidirection context-query attention blocks target to learn the attention from context to question and vice versa. The modeling layer as second encoder have the same
architecture as the encoding layer, which refine the outputs from the bidirectional attention layer. The final output blocks will generate the probabilities vector with the same length as the context to indicate the start and end index of the predicted answer.

3.1.1 Baseline: without character embedding

The baseline model is based on BiDAF network, and the only difference is without character embedding layer. The word embedding used here is the pre-trained Glove embedding with 300 dimensions. All implementation is given in default project(https://github.com/minggg/squad).

3.1.2 BiDAF with character embedding

As to match the original BiDAF model, we added character-level embedding block back into the Embedding layer. In the embedding block, it will firstly embed with the pre-trained 300 dimension Glove word-embedding for both context and questions and then concatenate with 64 dimension character-level embedding. The output of embedding feed to a dropout layer and highway layer, which aimed to increase the robustness of the model and refine the embedding representation.

3.2 QANet

In contrast to BiDAF, which applies the bidirectional-LSTM to capture the contextual information, QANet applies the self-attention accompany with convolution to reach the goal. The architecture has been shown below.

![QANet Model](https://github.com/minggg/squad)

Figure 2: QANet Model

More specifically, at the embedding block, the pre-trained character embedding will be passed to convolution layers in order to find numeric representation of words by looking at their character-level compositions. And then concatenate with the word embedding passing to the convolution layer and highway layer.

Encoder blocks consists of a positional encoding, layer normalization, depthwise separable 1d-convolution, self-attention and feed-forward layers. The positional encoding is added for the self-attention layer and the self-attention layer is for learning the dependencies between the words in the text and use that information to capture the internal structure of the text. The next block is the Context-Query Attention block. In this block, a similarity matrix \( S \in R^{n \times m} \) between each pair of context and query words will be calculated. The similarity function is the trilinear function: 

\[
f(q, c) = W_0 [q, c, q \cdot c],
\]

where \( \cdot \) is the element-wise multiplication and \( W_0 \) is a trainable variable. Then normalize the similarity matrix by row with softmax function to get \( S' \). The context-to-query attention (indicate as \( A \)) is computed as the matrix multiplication between \( S \) and query (indicate as \( Q \)).
Also normalize the similarity matrix by column to get $S''$. The query-to-context attention is calculated as $B = S' \cdot S'' \cdot C^T$, where $C$ represent query matrix. The rest part of the model is just the output block. The implementation is based on bangliu’s code (https://github.com/BangLiu/QANet-PyTorch) and hengruo’s code (https://github.com/hengruo/QANet-pytorch). To be more specific, most parts of the code are from bangliu’s implementation, except the Attention block, which is from hengruo’s implementation. Since we believe the implementation from hengruo is more consistent with the original paper.

### 3.2.1 with FastText embedding

Similar as Glove embedding, FastText is also a word based embedding method. However, it treats each word as composed of character ngrams. So the vector for a word is made of the sum of this character n-grams. With this improvement, it becomes to recent state-of-the-art English word vectors.

![Figure 3: FastText-Char embedding](image)

In this model, we applied FastText in QANet by replacing glove word embedding by using pre-trained fasttext embedding with 2 million word vectors trained on Common Crawl. The model architecture is same as the previous model.

### 3.2.2 with ELMo embedding

![Figure 4: ELMo-QANet embedding](image)

ELMo embedding is also called 'Deep Contextualised Word Representations'. This embedding/encoder block is more like BiDAF, which uses deep bi-LSTM blocks to learn the contextual information for the word within the context. ELMo representations are purely character based, which allows the network to use morphological clues to form robust representations for out-of-vocabulary tokens unseen in training. Considering ELMo is character based representation, we discarded the char-embedding in this model.
and applied ELMo in 2 different ways, 1) ELMo model only, 2) concatenate with Glove output. The
pre-trained model we are using is from AllenNLP. Considering time consumption part of LSTM,
we only use pre-trained small model with 13.6M parameters. With using ELMo model API from
AllenNLP, it generates the batch contextual word embedding lively with frozen weights and options,
and then output to Conv and highway models for further use.
Unfortunately, we don’t have results for this part yet. With ELMo only embedding, for 1000 context
takes about 4mins for embedding. Compares to other word based embedding methods, contextual
based embedding is indeed a time killer.

3.2.3 with BERT Embedding
BERT is another state-of-the-art model to generate the contextual work embedding. Its encoder
architecture is similar to QANet architecture. After tokenization, each word piece (some word will be
split to different pieces) will be embedd to a vector and then concatenate with the learnable positional
embedding. The combine embedding will be passed to multi-head attention, which captured different
contextual meaning of the the words, followed by the dropout layer, add & norm layer and feed
forward layers. Except the architecture, BERT also pre-trained with the next sentence prediction
and masked language model task, which are two genius methods to train the model. Considering
the model efficiency and limited memory resource, rather than BERT, we applied ALBERT [9],
which is a lite BERT. ALBERT has 18x fewer parameters and could train 1.7x faster comparing with
BERT. ALBERT mainly improve the BERT efficiency with two techniques, factorized embedding
parameterization and cross-layer parameter sharing.

The overall logic we applied to incorporate contextual embedding is shown in figure 3. We used the
‘transformer’ from huggingface for ALBERT with ‘albert-base-v2’ weights and freeze all weights,
because of CUDA limitation, to get the contextual embedding for context and questions. And then
feed the generated contextual embedding to a convectional block and a highway block. This two
blocks are designed by us and totally coded by orselves. The reason for us to add these two blocks is:
1. compact the information from the contextual embedding. 2. decrease the learnable parameters for
the overall model to increase the training and inference efficiency. In order to comparing with all
previous experiments, the following architecture of the model are same as before. The output from
highway block are fed into the QANet attention block, which calculate the Context-to-query attention
and query-to-context attention, and the output blocks.

4 Experiments

4.1 Data
We applied modified SQuAD 2.0 dataset. This data set consist of questions posed by crowdworkers on
a set of Wikipedia articles and the answer if either a segment of text, or span from the corresponding
4.2 Evaluation method

There are 3 metrics we have applied:

**F1**: it measures the portion of overlap tokens between the predicted answer and the truth answer.

**Extract Match (EM)**: scores will return 1 if the prediction is exactly same as the truth answer and 0 otherwise. This is useful in evaluating the predictions on negative sample, since the correct answer for the negative sample is None, if the model predicts something, then it will get 0 for this prediction.

**AvNA**: this measures the classification accuracy of the model when only considering its answer (any span predicted) vs. no-answer predictions.

We also take the training time into our consideration when evaluating the model, since we believe the train efficiency and inference time are important factors for model performance as well. We train different model with same GPU setting, and tried to comparing the time that different models achieved same accuracy.

4.3 Experimental details and Error Analysis

4.3.1 Baseline: BiDAF, word embedding only

Baseline model is the provided, which is similar to BiDAF, but without the character embedding layers. We did not change any hyper-parameters. The training time is around 7-8 hours.

The Baseline model achieved 68.25 AvNA, 57.95 EM and 61.25 F1 scores. From the error analysis, we found that the model tend to answer non-answer question incorrectly, like the example given at figure 6. The question asked about the 'rarest cause', however, the model answered the 'most common cause' instead. From our perspective, this may because the baseline model only applied the word embedding, which tend to generate similar embedding for words with similar context words, like 'rarest' and 'common'.

![Figure 6: Error Analysis: Baseline model](image)

4.3.2 BiDAF, with character embedding

When adding the character embedding, the training time is a bit longer than the baseline model, which is around 8-9 hours. AvNet achieved 68.19 AvNA, 58.04 EM and 61.25 F1 scores. Except EM scores, the other two metrics did not have large improvements. The model have better performance when the question has no answers but tends to give longer answer, like example in Figure 7. Hence, the EM score did not increase that much.

4.3.3 QANet

The hyper-parameter for QANet model were pretty similar to the previous setting. The Dropout rate is 0.2, and kernel size and channels for the convolution layer are same as well. Considering the training times issue and GPU capacity, we set the number of heads of multi-attention equals to 1 and 6.
decrease the batch size to 16. Also because of the resource issue, we decrease the sample size to 64. The training time is around 12 hours.

For the QANet model, AvNet is 68.92, EM is 59.67 and F1 score is 63.01. All metrics increased a bit comparing with the previous two models. But one largest issue we found is that the model sometimes could not capture the correct answer of the problem, even if it appears in the paragraph, like the example given at figure 8.

We also tried to increase the number of multi-head attentions to 4 and maintain all other settings the same as previous model. The training time increase to 27 hours. But the performance of the model increased a lot as well. For dev set, AvNet increased to 72.96, EM increased to 63.42 and F1 increased to 66.76.

We believe this is increase the number of multi-attention did helps in increasing the model performance. According to the paper, “multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.”

4.3.4 QANet + BERT

When combineing BERT and QANet, the model took quite a long time when generating the BERT embedding, although the following model training is pretty soon. Although we tried to use whole, \( \frac{1}{3} \), \( \frac{1}{4} \) and \( \frac{1}{16} \) of the data, but the CUDA always out of memory. So at the end, we only slice \( \frac{1}{64} \) data from the training set. And the metrics changed is shown below at figure 9. The performance is pretty bad. This may because the training data is too few and we freeze all parameters in ALBERT, hence the generated contextual embedding is not good enough for this data.

4.3.5 QANet-FastText

We applied pre-trained FastText model with 2 million word vectors trained on Common Crawl (600B tokens) and then concatenate with char embeddings. The total run time for this model takes about 26 hours with same training and hyper parameter settings as previous model. The performance details show as below at figure 10:
Both train loss and dev loss continues to improve throughout. Although the dev NLL improves throughout the training period, the dev EM and F1 scores initially get worse at the start of training, before then improving. This is due to no-answer. However, unlike in based model, it improves quickly starting at around 300K iterations, and AvNA increase to 74.3 and Loss reduces to 2.5. Ultimately, we got 65.636 EM and 68.853 F1 for dev set and 61.809 EM and 65.1 F1 for test set. Moreover, the trending of EM/F1 score still arise, I think if we increase number of epoch will also improve the score as well.

5 Conclusion

In this project, we applied different embedding methods and encoder architectures based on BiDAF and QANet to solve question answering problems. We experimented model whose encoder block based on Bi-directional LSTM and attention separately. For the Bidirectional LSTM based encoder, we compared BiDAF model without character embedding and with character embedding. For the attention based encoder, we experimented QANet model with original setting, fast-text embedding, ELMo embedding and BERT Embedding. General speaking, we observed that attention based encoder could achieve similar performance quickly than the LSTM based encoder. Also based on the error analysis and experiments, character embedding could improve the model performance if concatenating with word embedding. For the pre-trained embedding, comparing with Glove embedding, fast-text embedding could achieve better performance if hole all other parts of the model same. Although we expected to observe further improvement when introducing contextual embedding, the limitation of the virtual machine resource made the apple-to-apple comparison not available.

In the next step, we want to run the full data set for the QANet+BERT embedding model with advanced virtual machine setting and comparing its performance with the other experiments.
References


