SQuAD Question Answering with Transformers

Chitradip Mandal
Department of Computer Science
Stanford University
cmandal@stanford.edu

Abstract

In this project we use us the QANet architecture to answer questions on the SQuAD v2.0 dataset. In addition to the reference architecture, we add a parts of speed encoding to each word. Also we add a penalty to long answers which gives slightly better F1/EM scores. We show the potential speed advantage of a non-recurrent based architecture like QA net over traditional RNN based architectures.

1 Introduction

Question answering is a valuable NLP task. It is an effective indicator for machine comprehension. We use the SQuAD (7) version 2.0 dataset to evaluate our Question Answering model.

Our model is based on QANet (10) with a few modifications. We add Part of Speech Tagging using the NLTK library (2) and add it to the input vector. In addition, we discount the probability of longer answers. As in QANet, the model does not contain any recurrent layers.

Our model is composed of four high level layers: 1) Input embedding layer consisting of 300-dimensional GloVe vectors concatenated with learnable vectors for character level and parts of speech tagging, 2) Bidirectional question-context attention layer 3) Model Encoding layer and 4) Output encoding layer. The model achieves a F1 of 63.94 on the DEV set.

2 Related Work

There has been lot of work on SQuAD question answering. The majority of recent work is with Pretrained Contextual Embeddings like BERT (5). Our model does not use pre-trained embeddings. Yu u. a. (10) proposed the RNN-free QANet architecture that our model is based on. Seo u. a. (8) proposed the Bidirectional attention model used in our BiDAF attention layer. The inspiration for Part of Speech tagging is taken from Chen u. a. (4)

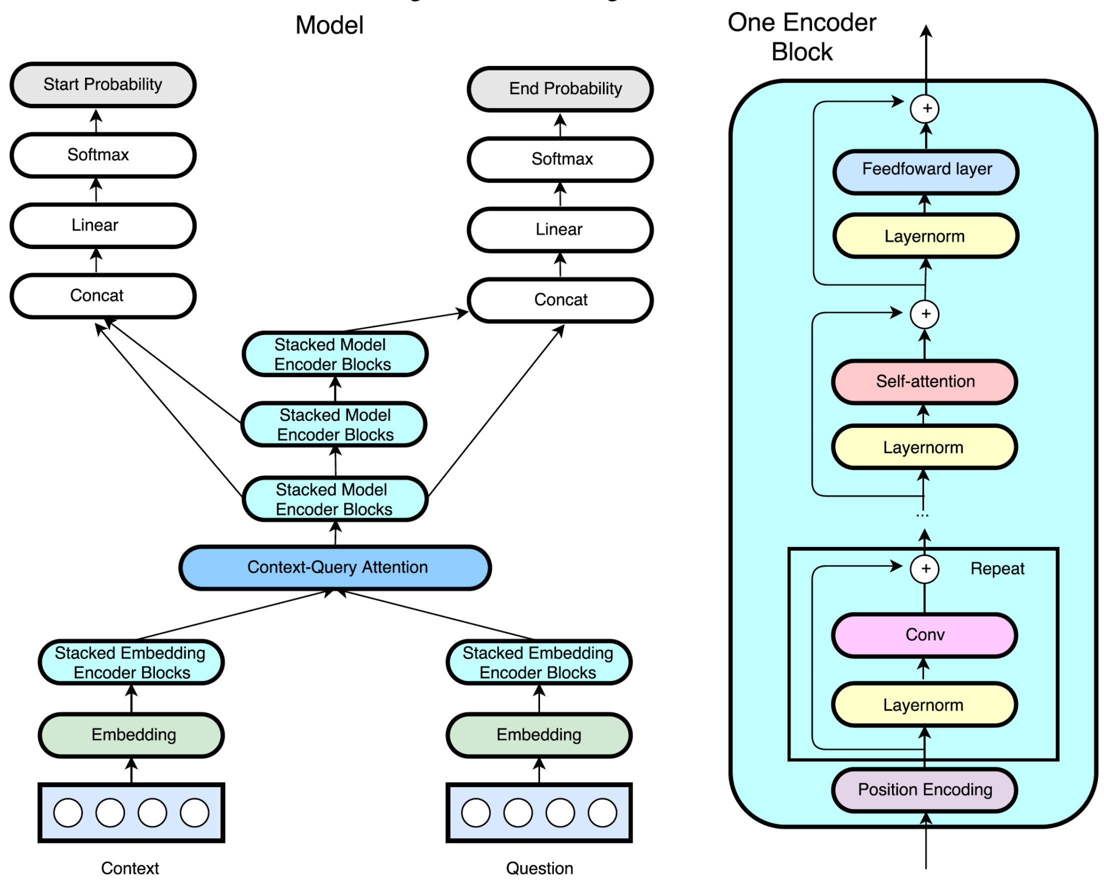
3 Approach

Figure 1 from the paper shows the architecture of the model. The code is structured in 4 layers - Input Encoding, Context-Query (BiDAF) attention, Model Encoding and Output. The workhorse for both the Input Encoding and Model Encoding is the EncoderBlock.

3.1 Encoder Block

The general **encoder block** (shown on the right side of Figure 1) has 3 layers. The first layer consists of a number of convolution layers. The second layer is a multi headed self-attention layer (Vaswani u. a. (9)). Lastly, there is a feed forward layer. All of these individual sub-blocks (i.e. each convolution layer, attention layer, and feed forward layer) is placed in a residual block with layer

Figure 1: Model Diagram



normalization. The residual block has a full identity from input to output, so for any operation, output $y = f(layernorm(x)) + x$

We have coded most components of Encoder Block ourselves using pytorch (1). This includes the Convolution layers, Layer Normalizations, Feed forward network and Residual connections. We use standard modules like Linear, LayerNorm, Dropout, and Conv1D. For the Multiheaded Attention layer and the Positional encoding layer, we used code from OpenNMT (6), also known as "The Annotated Transformer" (`http://nlp.seas.harvard.edu/2018/04/03/attention.html`). We also explored the Attention function from `https://github.com/jadore801120/attention-is-all-you-need-pytorch`, but decided to use the OpenNMT code.

In this code, we can vary the model size, the kernel size for convolutions, the number of heads for the MultiHead Attention parameters and the dropout rate.

3.2 Input Encoding Layer

The **Input Encoding Layer** is composed of an encoding of the word embeddings, character embeddings and embeddings based on parts of speech tagging.

The word embeddings used are the 300 dimension GloVe vectors. The character encoding consists of learnable embeddings. Each word is composed of a maximum 16 characters. We use convolution with kernel size 5 and max pooling to convert the representation to a 200 dimensional vector. This was also put through a highway layer with dropout.

$$char\_embedding = highway(maxpool(ReLU(conv1d(c_e))))$$

We tagged each sentence with parts of speech using the NLTK library. These tags are encoded using learnable embeddings.

$$pos\_embedding = highway(ReLU(linear(pos\_tag\_embedding)))$$

The final input encoding is the concatenation of word, character and pos embeddings. These are transformed to the hidden or "model" size by 1X1 convolution. These are then encoded using 1 instance of the Encoder block described above.

3.3 Context-Query Attention

For the **Context-Query Attention** layer, we use the BiDAF attention layer provided by in the starter code for the Default final project. We calculate a similarity matrix between the encoded context and query as per BiDAF (8). We then normalize row wise and matrix multiply with the encoded query to get the context to query attention. $c2q = S1 \cdot Q^T$ where $S1$ is the row wise normalized similarity matrix. Also query to context is calculated by $q2c = S1 \cdot S2^T \cdot C^T$ where $S2$ is the column wise normalized similarity matrix. The output of this is the concatenated $[\ c \quad c2q \quad c \times c2q \quad c \times q2c\ ]$

3.4 Model Encoding Layer

Each model encoder consists of 7 Encoder blocks and is coded from scratch using the EncoderBlock described above. We have 3 of the model encoding layers as in 1. We share the weights of the three encoders.

3.5 Output Layer

Finally, the **Output layer** uses a linear transformation and softmax to perdict the start and end tokens. $p_{start} = softmax(W_0[M_0; M_1]$ and $p_{end} = softmax(W_1[M_0; M_2])$. where the M_n represent three model encoders, and W_n are linear layers. The score of any span in the product of the start and end probabilities. We also restrict the span to be within a certain value. (15 in our case).

In addition, we also observe that our model favors long answers to short ones. Also we see that most of the answers are short. To compensate for the effect, we reduce the score of longer spans when calculating the maximum.

4 Experiments

We are using the provided modified SQuAD v2.0 data to measure performance of Question Answering using F1 and EM matrices.

4.1 Hyper parameters

We used Adams optimizer with a learning rate of 0.001, $\beta_1 = 0.8, \beta_2 = 0.999, \epsilon = 10^{-7}$

For regularization, we used dropout on all layers (and sub layers) with dropout probability 0.1. We also L2 decay on all training variables with $\lambda = 3 * 10^{-7}$. Also, we applied Exponential Moving Average to all training variables with a decay of .9999

4.2 Experiments

We use 300 dimensional GLoVe vectors as word embeddings (apart from character and POS embeddings described before). We also tried 300 dimensional fasttext embeddings. (3). Surprisingly, this did not give us better results.

Due to resource constraints ran many experiments on a smaller network to try out features. Finally we ran a few of them at full scale.

Table 1 shows some complete experiments with a smaller network, which gave us clues on which ones to expend resources on bigger experiments.

Model	F1	EM
FastText + character	58.39	56.43
Glove + character	59.29	55.55
Glove + character + positional	60.17	57.39

Table 1: Experiments with a Small network

We ran our models on Micorsoft Azure using $\frac{1}{2}$ of NVDIA Telsa M60 GPU.

One of the main reasons for using QANet was the advertised training speed. However, contrary to expectation, we found that training was extremely slow. We also tried to run this in a Tesla P100. We did not complete a full run, since it was extremely expensive, but we still got about 120 iterations per second vs 800 iterations per second for the LSTM based BiDAF model. The main reason for this is that the size of the model is much larger than LSTM, so we can run LSTM with a much larger batch size.

Table 2 Shows the results on the DEV set. The numbers in brackets refer to the size of the model.

Model	Steps	Training Time	F1	EM
BiDAF baseline	3 M	6hs	60.118	57.016
QANet (96) word embeddings only	2.8 M	20 hrs	60.205	56.663
QANet (128) + Char embedding	2.5 M	21 hrs	62.71	59.25
QANet (128) + Char embedding + POS embedding	2.5 M	21 hrs	63.92	60.65

Table 2: Results

5 Analysis

Our model makes a few different types of errors. In the following examples, the context is shortened in interest of space.

5.1 Longer prediction

Sometimes our model predicts something longer than the model answer.

Question: What long term agenda was the acts of plundering Muslim lands by the West?
Context: .... Westernizing Muslims were actually agents of the West serving Western interests, and that the acts such as "plundering" of Muslim lands was part of a long-term conspiracy against Islam by the Western governments.
Answer: conspiracy
Prediction: conspiracy against Islam

In this case, we could argue that the model was reasonable and more descriptive.

5.2 Incorrect prediction

Some answers was answered wrong since our model misunderstood the question.

Question: What did Geroge Lenczowski do to the price of oil on October 16, 1973?
Context: In response to American aid to Israel, on October 16, 1973, OPEC raised the posted price of oil by 70%, to $5.11 a barrel. .... George Lenczowski notes, "Military supplies did not exhaust Nixon's eagerness to prevent Israel's collapse...
Answer: N/A
Prediction: $5.11 a barrel

In this case, the model interpreted looked the question as "What was the price of oil on October 16, 1973?" . However the model missed the fact that Geroge Lenczowski did not have anything to do with it.

5.3 Questionable model answers

Some answers are somewhat questionable. For example:

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") .... 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: DTIME(f(n))

In this case, the model output an "expression" of the time required. While the given answer "state transitions" is potentially correct, the expression DTIME(f(n)) is also correct. Both the answer and the prediction needs more information to be fully correct.

6 Conclusion and Future work

In this paper we implement the QANet architecture with a few modifications and achieve a reasonable performance on answering questions on the SQuAD dataset.

Our implementation of the QANet architecture was somewhat disappointing with respect to speed. The paper says that they could train their model in 3 hours with a batchsize of 32 on NVIDIA Tesla P100 GPUs. We have access to a much older generation of GPU ($\frac{1}{2}$ of M60). We could only use a batch size of 8. A higher batch size results in an out of memory error. So, although the QANet model is parallelizable, we are not able to utilize this capability.

We tried a few iterations on a more expensive Tesla V100 GPU, but still could not get the desired speedup. The next step would be to optimize the implementation and attempt to make it more memory efficient so that we can fit a larger batch into memory.

Future work would probably use state of the art PCE based models like BERT and ELMO. We would like to include our changes, for example POS tagging and discounting longer answers in a PCE based model to get better performance.

References

[1] *PyTorch*. https://pytorch.org. – Accessed: 2019-03-01

[2] BIRD, Steven ; KLEIN, Ewan ; LOPER, Edward: *Natural Language Processing with Python*. 01 2009. – ISBN 978-0-596-51649-9

[3] BOJANOWSKI, Piotr ; GRAVE, Edouard ; JOULIN, Armand ; MIKOLOV, Tomas: Enriching Word Vectors with Subword Information. In: *arXiv preprint arXiv:1607.04606* (2016)

[4] CHEN, Danqi ; FISCH, Adam ; WESTON, Jason ; BORDES, Antoine: Reading Wikipedia to Answer Open-Domain Questions. In: *CoRR* abs/1704.00051 (2017)

[5] DEVLIN, Jacob ; CHANG, Ming-Wei ; LEE, Kenton ; TOUTANOVA, Kristina: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: *CoRR* abs/1810.04805 (2018)

[6] KLEIN, Guillaume ; KIM, Yoon ; DENG, Yuntian ; SENELLART, Jean ; RUSH, Alexander M.: OpenNMT: Open-Source Toolkit for Neural Machine Translation. In: *Proc. ACL*, URL https://doi.org/10.18653/v1/P17-4012, 2017

[7] RAJPURKAR, Pranav ; ZHANG, Jian ; LOPYREV, Konstantin ; LIANG, Percy: SQuAD: 100, 000+ Questions for Machine Comprehension of Text. In: *CoRR* abs/1606.05250 (2016)

[8] SEO, Min J. ; KEMBHAVI, Aniruddha ; FARHADI, Ali ; HAJISHIRZI, Hannaneh: Bidirectional Attention Flow for Machine Comprehension. In: *CoRR* abs/1611.01603 (2016)

[9] VASWANI, Ashish ; SHAZEER, Noam ; PARMAR, Niki ; USZKOREIT, Jakob ; JONES, Llion ; GOMEZ, Aidan N. ; KAISER, Lukasz ; POLOSUKHIN, Illia: Attention Is All You Need. In: *CoRR* abs/1706.03762 (2017)

[10] YU, Adams W. ; DOHAN, David ; LUONG, Minh-Thang ; ZHAO, Rui ; CHEN, Kai ; NOROUZI, Mohammad ; LE, Quoc V.: QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension. In: *arXiv e-prints* (2018), Apr, S. arXiv:1804.09541