Question Answering Models for SQuAD 2.0

Ryle Zhou

Stanford University
rylezhou@stanford.edu

Abstract

Question answering is an important NLP task. QA systems allow a user to ask a question in natural language, and receive the answer to their question quickly. SQuAD 2.0 is a challenging natural language understanding task. Models preconditioned with BERT achieved better than human performance on SQuAD 1.1 and currently lead on SQuAD 2.0 top performance. In this project, my goal is to find out what to do to make BERT work for SQuAD 2.0 and see what can be improved. I combined BERT with BiDAF and made a new model called BERTPlusBiDILSTM.

1 Introduction

SQuAD 2.0 is more challenging than SQuAD 1.0 because except for completing the task of making the right prediction of the right answer, it requires my model to understand whether an answer is in certain span. In other words, it requires my model to make a decision of whether a hypothesis is supported by, contradicted by, or neutral with respect to an assumption. Moreover, my model must understand when a possible relationship between two entities is not entailed by the text. BERT contains deep and contextual(benefit of bidirectionally) word embeddings representation for each word. BERT has became the new trend for NLP tasks, including SQuAD 2.0, since released. I started my exploration by understanding what it is and how it works. BERT is a method of pre-training language representations, meaning that we train a general-purpose language model on massive corpus and then fine tuning this model for SQuAD 2.0. BERT learns useful text representations by being pre-trained on two different tasks: 1. In a sentence with two words masked, BERT is trained to predict what those two words are. 2. Given two sentences, BERT is trained to determine whether one of these sentences comes after the other in a piece of text, or whether they are just two unrelated sentences. I found it challenging to fine-tune BERT for SQuAD 2.0. I understand that the idea is to use BERT pretrained values with added different input and output layers that work effectual for SQuAD. Since BERT has many parameters, it adds complexity for the fine-tuning process. The input is the question and corresponding paragraph, while the output is the start, end answer span for the question.(*Figure* 1 below)

2 Approach

2.1 SQuAD 2.0 Unanswerable Example

Findings:
(1). There are about 33.38%(43,498 added to SQuAD 1.1) of negative examples(new) in SQuAD 2.0 dataset that are spread out in 64.48% of the total articles.
(2). There are about 60.87% articles contain negative examples in test dataset.
(3). There are about 72.92% articles contain negative examples in development.
Refer to *Figure 1* below.

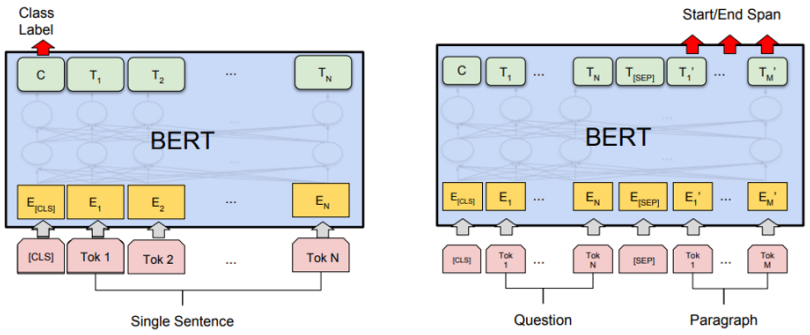

Figure 1: BERT for SQuAD 2.0

Article: Endangered Species Act
Paragraph: "*. . . Other legislation followed, including the Migratory Bird Conservation Act of 1929, a 1937 treaty prohibiting the hunting of right and gray whales, and the Bald Eagle Protection Act of 1940. These later laws had a low cost to society—the species were relatively rare—and little opposition was raised.*"

Question 1: "*Which laws faced significant opposition?*"
Plausible Answer: *later laws*

Question 2: "*What was the name of the 1937 treaty?*"
Plausible Answer: *Bald Eagle Protection Act*

(a) Two no answer questions with plausible answers in red. Relevant key words in blue.

	SQuAD 1.1	SQuAD 2.0
Train		
Total examples	87,599	130,319
Negative examples	0	43,498
Total articles	442	442
Articles with negatives	0	285
Development		
Total examples	10,570	11,873
Negative examples	0	5,945
Total articles	48	35
Articles with negatives	0	35
Test		
Total examples	9,533	8,862
Negative examples	0	4,332
Total articles	46	28
Articles with negatives	0	28

(b) Data Comparison between SQuAD 1.1 and SQuAD 2.0

Figure 2: SQuAD 2.0 Analysis

2.2 Baseline

2.2.1 BiDAF

I start with BiDAF model with character embeddings.

Figure 3 shows the architecture of BiDAF model.

Embedding Layer: character level embedding layer plus word embedding - maps each word to a vector space using character-level CNNs and a pre-trained word embedding model.
Contextual Embedding Layer: utilizes contextual cues from surrounding words to refine the embedding of the words. These first three layers are applied to both the query and context.
Attention Flow Layer: couples the query and context vectors and produces a set of query aware feature vectors for each word in the context.
Modeling Layer: employs a Recurrent Neural Network to scan the context.
Output Layer: provides an answer to the query.
Detailed eqauation for BiDAF can be found in the original paper *Bidirectional Attention Flow for Machine Comprehension*(Seo et al.,2017)

2.3 Building on top of Baseline

2.3.1 Bert

BERT's architecture is based on a bidirectional Transformer encoder. BERT Base has 12 encoder layers and BERT Large has 24 encoder layers. BERT Base has 768 hidden states and BERT Large has 1024 for feed forward networks. There are 12 attention heads and 16 for BERT Base and BERT Large respectively. BERT's input representation is able to represent a single text sentence or a pair of text sentences. Each token or each word is represented by the summation of its word embedding, a

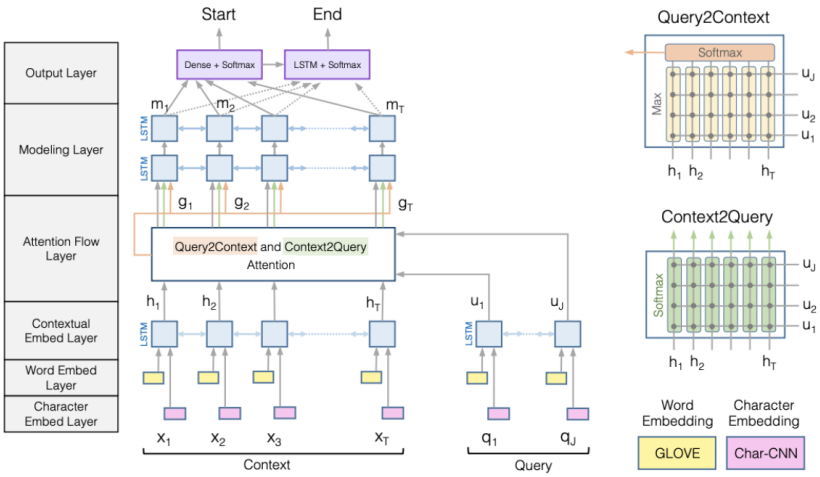

Figure 3: BiDirectional Attention Flow Model

learned positional embedding, and a learned segment embedding. The word embedding used in the paper is WordPiece embeddings in tokenlization. The positional embedding captures the order of the word in the sequence. The learned segment embedding associates tokens with a corresponding sentence if the input can be a pair of text sentences. Once BERT is pretrained, task-specific models are formed by adding one additional output layer, therefore a minimal number of parameters need to be learned from scratch which will be beneficial for the time that being spent on learning from zero.

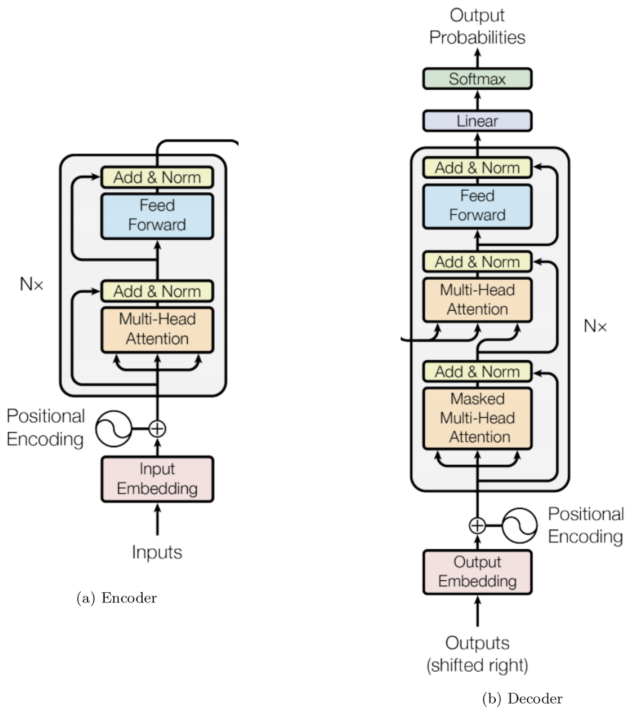

Figure 4: BERT model structure

2.3.2 BertPlusBiDiLSTM

BERT makes use of Transformer, an attention mechanism that learns contextual relations between words in a text. In its vanilla form, Transformer includes two separate mechanisms - an encoder

that reads the text input and a decoder that produces a prediction for the task. Since BERT has a much stronger attention mechanism, and BiDAF has the advantage of memorizing longer context, I came up with an idea of replacing the attention flow layer from BiDAF with BERT and run the last modeling layer and output layer to see if this can improve the performance of my baseline.

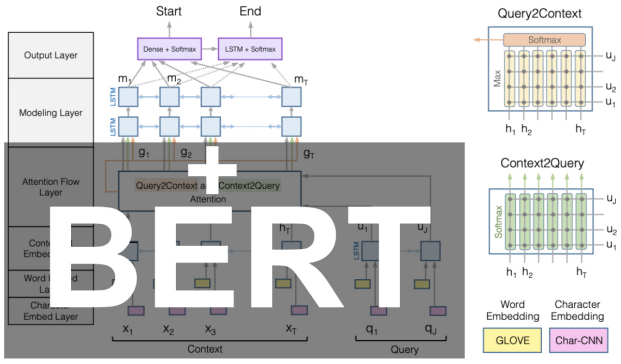

Figure 5: BERTPlusBiDiLSTM

3 Experiments

3.1 Data

I evaluate the model on SQuAD 2.0. SQuAD 2.0 combines existing SQuAD 1.1 data(collected from Wikipedia articles) with 53,775 new, unanswerable questions about the same paragraphs(paragraphs that are shorter than 500 characters were removed) written adversarially by crowdworkers to look similar to answerable ones. If questions crowdworkers wrote 25 or fewer questions on the article assigned, those questions were removed to ensure a high quality of the dataset. SQuAD 2.0 is a challenging natural language understanding task for existing models: a strong neural system that gets 86% F1 on SQuAD 1.1 achieves only 66% F1 on SQuAD 2.0.

3.2 Evaluation method

Exact match and F1 scores are used for evaluation.
Exact match (EM) assigns a full credit 1.0 if the predicted answer is equal to the gold answer and 0.0 otherwise.
F1 score computes the average word overlap between predicted and correct answers.

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{1}$$

3.3 Experimental Details

3.3.1 BiDAF with Character embeddings

Compare with the starter code, character embeddings are added on top of word embeddings. Word vectors and character vectors are from GloVe pre-trained result. And then character and word embeddings are concatenated together to pass to a 1D Convolutional Neural Network and a Highway Encoder(number of layers is 2). This process captures all information from words and characters. The output of the Highway Encoder are passed into a Bidirectional LSTM layer called RNN Encoder. The outputs of two directions are concatenated into contextual vectors. Followed by the BiDAFAttention layer(the layer that is making attention mechanism happening), attention scores are calculated base on each time step to see the similarity between context and query from the previous two directions. The attention vecters and contextual embeddings are combined together. Another modeling layer(2 layers of bidirectional LSTM) are used to integrate temporal information between contextual representations conditioned on the question. The final output layer is using attention layer outputs and modeling layer outputs to a bidirectional LSTM. Softmax is used for finally produce the start of the end of context.

No answer prediction
In order to avoid <unk> and predict no answer, out-of-vocabulary tokens were prepended to the beginning of each sentence. The model outputs soft-predictions as usual. If $P_{start}(0)$ ·P$_{end}(0)$ is greater than any predicted answer span, the model predicts no answer.

3.3.2 Solving BERT'Cuda runtime error(): out of memory' problem and finding hyper parameters that can make BERT run

BERT can achieve SOTA performace in theory. However, every time I try to stack more than a few samples(I tried 32, 16, 10, 6, 2, 1) for batch size I get a CUDA RuntimeError: out of memory error. This problem needs to be solved before I can do any other experiments.

Solution 1: Accumulating gradients During the loss.backward() operation, gradients are computed for each parameter and stored in a tensor associated to each parameter: parameter.grad. Changing accumulation gradient step number will change how many steps to accumulate when we sum the gradients of the number of backward operations in the parameter.grad tensors before calling optimizer.step() to perform a step of gradient descent. After I changed the gradient accumulation to 10(a number that's not too large and not too small in my opinion), batch size being 30,20 still didn't work. Only if I make batch size to 10 as well, my model started running. I think in this method , batch size is further regulated by accumulating gradients.

Solution 2: Making gradient checkpoint The idea is to back-propagate the gradients in small chunks along the model, trading the memory needed to store a full back propagation graph with the additional compute of a partial forward pass associated to each chunk. This turns out to be a slow method because additional computation for gradient checkpoints is added. I did not choose this solution given that gradient accumulation already works.

The way of coding also affects how memory is occupied. Since I added RNN to BERT, it increased memory usage of GPUs. Stuck with out of memory problem caused a majority of my experiment time. It is an important lessen to learn before my next project. Getting the right VMs and having using memory wisely in mind while coding should be always taken into considerations for any other tasks.

3.3.3 Hyper Parameter Fine Tuning for BERT

To fine-tune the BERT model, the first step is to define the right input and output layer. It turns out that model performance depends heavily on the hyper parameter values selected. When I first start running BERT, I kept getting "cuda run out of memory error" with even turnig batch size to 1. Knowing that most of the time stochastic gradient descent algorithms require larger batches than just a handful of examples to get decent results. In this case, training with large batches when GPU cannot handle them is the first problem to solve. I found out by adjusting gradients accumulation will help when training on Nv6 VM. The current running version of BERTPlusBiDiLSTM has gradient accumulation steps = 10 and train batch size = 10.

I tried to use Azure Machine Learning Service to define a search space for different hyper parameter options. I found out that the Azure MLS provides hyper parameter configurations to find the ones result in the best performance. For example, learning rate can be searched from 1e-4 to 1e-6. It can see if 1e-4 or 1e-5 has better result than 1e-6. And then Azure MLS can also visualize the result. After I started implement in Azure notebook, I realized this tool is not very easy to use because it requires almost another complete building up model from ground in iPython and if any mistakes happen the auto tune will not work. The search also takes a long time for each excution. After evaluation, I continued my experiment without taking the service.

3.4 Result

3.4.1 Data Comparison

Compared with the starter BiDAF model, Dev NLL: 03.20, F1: 59.59, EM: 56.39, AvNA: 66.46 I achieved, Dev NLL: 02.93, F1: 63.73, EM: 60.31, AvNA: 69.82. An increase of 4.14% in F1 and 3.92% increase in EM from BiDAF without character embeddings. BiDAF's performance is 2.27%

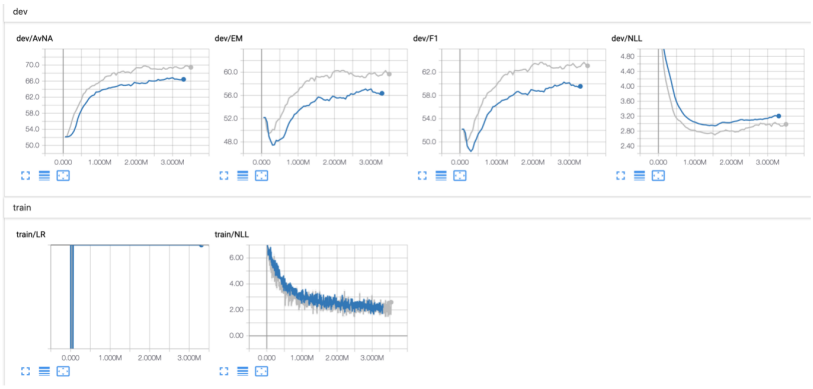

Figure 6: TensorBoard visulization of results while in training. Blue is the BiDAF without character embedings model, Grey indicates the starter BiDAF model

from the F1 score mentioned in the SQuAD 2.0 paper. *Know What You Don't Know: Unanswerable Questions for SQuAD*(Rajpurkar et al.,2018)

If it runs correctly, BERT model alone will have F1 score around 79-81% and EM of almost 80. Which would be about 16% increase from BiDAF baseline. Since I haven't finished training BERTPlusBiDiLSTM on Nv6 and Nv12, I don't have the date to demonstrate my result here yet. I cannot make any conclusion of whether BERTPlusBiDiLSTM is better than the single BERT model.

3.4.2 Takeaways

Model size matters. BERT large, with 345 million parameters, is the largest model of its kind. It is demonstrably superior on small-scale tasks to BERT base, which uses the same architecture with 110 million parameters.However, we need corresponding computational power to make BERT work. I need to learn more about how to evaluate different VMs for which task in the future. With enough training data, more training steps will result with a higher accuracy. For instance, the BERT base accuracy improves when trained on 1M steps compared to 500k steps with the same batch size some research paper. BERT's bidirectional approach converges slower than left-to-right approaches, because only 15% of words are predicted in each batch, but bidirectional training still outperforms left-to-right training after a small number of pre-training steps. This is proven by compare BERT with BiDAF.

4 Future Work

I will continue training until I get a working result from BERTPlusBiLSTM first.

4.0.1 Agile training methods

The choice of pre-trained embeddings for initializing word vectors has a significant impact on the performance of deep neural models for SQuAD 2.0. Developing from BERT will be my choice to begin with and continue with for SQuAD and other NLP tasks. But in this project I underestimate the fact that tuning BERT for specific task can be very time consuming. Because training each hypothesis or new idea, it takes a long time to train the entire SQuAD 2.0. This causes the turnaround time of each experiment to be very long(mostly over 12 hours). Expecially, when I added BiLSTM to BERT, each train took over 20 hours. One thing I learned for sure is that I need to always remember to print out loss values while training. Because for the first few experiments I ran, I got empty predictions although my model ran and showed progress. Building faster model to train on large dataset and reduce turnaround time should be taken into account every time. The more agile methods will more efficient.

4.0.2 Transformer XL

I haven't had a change to add transformer XL to BERT yet. Replacing BiLSTM with transformer XL hopefully can reduce the training time and have a positive result in improving accuracy. If it does not have a positive impact, other algorithms need to studied further more.

4.0.3 Visualization and Readability

During my experiments, I found it very hard to grasp whether my model is doing well and where to improve because I couldn't see where the model is marking the predicted answers compare with the correct answers or which related words to casue the selection of the predicted answers. The future work to do here is to find out a way to show these process.

4.0.4 No answer prediction

The main difficulty for SQuAD 2.0 is the no answer prediction. Since human language can be very complex and confusing for deep learning models to understand, except for letting the models see no answer cases(I think just giving no answer label is not sufficient for models to avoid making mistakes), it might be helpful to train model with the reasons that the answer for some questions is no answer.