A BERT Based Question Answering System
CS7643 Project 2020 Fall

Jiaqi Jiang, Jiatuan Luo, and Yanxin Ye
jjiang334@gatech.edu, jluo324@gatech.edu, yye79@gatech.edu
Georgia Institute of Technology
Atlanta, GA 30308, USA

Abstract

Language comprehension has always been one of the most intriguing tasks in artificial intelligence. In our project, we aim to create a question answering bot with language understanding capabilities by applying a fine-tuned version of Bidirectional Encoder Representations from Transformers, also known as BERT. And we experiment with different neural network architectures stacked on top of the BERT layers, such as CNN and MLP layers as well as test with different hyperparameters to examine their effect on model performance. We fine-tune our model on SQuAD benchmark and use ExactMatch (EM) and Macro-averaged F1 scores to benchmark model performance. Among the models we have tested, the MLP feedforward architecture with ReLU activation and a dropout layer produced the best result over other models, with a EM score of 72.50 and a Macro-averaged F1 score of 75.80. In this paper we will extensively discuss our experiment settings and results.

1. Introduction

The goal of our project is to develop a functioning AI with language understanding capability and is able to absorb information from data in the form of text and give accurate responses to questions associated with the text provided. Language comprehension has long been a point of interest in the realm of machine learning and deep learning. With the advent of the concept of attention in 2017 [6], it was made possible to produce adequate question answering bots with deep learning architectures that are capable of understanding sequential relationship between words such as RNN and LSTM. However, these architectures have their drawbacks, as RNN cannot process very long sequences and can suffer from gradient vanishing and exploding problems. And RNNs are inherently difficult to train, even more so for LSTM since tokens are passed in sequentially through the neural net. And traditional methods are prone to overfitting with limited amount of training data, yielding unsatisfactory performance. Then the rise of transfer learning and BERT [1] effectively introduced more superior methodologies to solve this problem through pre-training a set of encoders from transformers with enormous amount of text data, and then fine tune them to perform specific tasks, in our case, creating a chatbot that is capable of understanding language and formulate accurate response from text data.

Artificial intelligence with the capability of understanding text and gathering and regurgitating useful information can be applied across industries. Some of the use cases that comes to mind include helping investment banks understand the market and make investment decisions by quickly processing through thousands of news and articles from various sources generated each day, and serving as a knowledgeable information source to helping students learn more effectively.

In this paper, we build on top of a pre-trained BERT model and focus on fine-tuning the model to perform question answering tasks. And we experiment with different architectures and hyperparameters to produce the best results. We are able to achieve respectable model performance with the base pre-trained BERT with 12 layers and 110 million parameters, but due to limitations in computational power, GPU memory and RAM to be specific, we are unable to further boost our model performance with large BERT models and increasing the number of parameters to 336 million, which can be done readily with more computational resources.

2. Background

2.1. BERT Pretrained Approach

In this session we give a brief overview of BERT, a language representation model proposed by [1]. It is commonly used in NLP tasks and could be fine-tuned on downstream tasks.

BERT takes as input a concatenation of two segments

Table 1. QA task on SQuAD: EM and F1 scores on fine-tuned models.

Model	EM	F1
BERT-Large, Uncased	30.0	43.21
BERT-QA	11.42	46.55
BioBERT-QA	54.28	85.02
EHR-QA	70.0	84.91

(sequences of tokens), $x_1, ..., x_N$ and $y_1, ..., y_M$. Segments usually consist of more than one natural sentence. The two segments are presented as a single input sequence to BERT with special tokens delimiting them: $[CLS], x_1, ..., x_N, [SEP], y_1, ..., y_M, [EOS]$. M and N are constrained such that $M + N < T$, where T is a parameter that controls the maximum sequence length during training. The model is first pretrained on a large unlabeled text corpus and subsequently finetuned using end-task labeled data.

BERT uses the now ubiquitous transformer architecture [6], which we will not review in detail. During pretraining, BERT uses two objectives: masked language modeling (MLM) and next sentence prediction (NSP). BERT is trained on a combination of BOOKCORPUS [7] plus English WIKIPEDIA, which totals 16GB of uncompressed text. BERT is optimized with Adam[2].

2.2. BERT based Question Answering

BERT based transfer learning has been successfully used in question answering. [4] applied four fine-tuned models to answer questions in a patient discharge summary in an electronic health record (EHR): (1) **BERT-Large, Uncased**: Large, Uncased, Whole Word Masking BERT; (2) **BERT-QA**: BERT-Large, Uncased fine-tuned on the SQuAD dataset; (3) **BioBERT-QA**: BioBERT fine tuned using PubMed and PubMed Central publications; (4) **EHR-QA**: an extended BioBERT-QA fine-tuned on unstructured EHR data. The reported results as of exact match (EM) and F1 are shown in Figure1. BioBERT-QA performed better than the standard BERT and BERT-QA baseline models. It reflected in its strong F1, even though no target domain data was used in training. EHR-QA substantially improves the performance of BioBERT-QA. Results show that fine-tuned BioBERT-QA on specific EHR data achieves competitive performance, suggesting that it exploits the specific knowledge encoded in the domain description and improves the models' generalization.

3. Approach

QA models trained on small datasets are likely to overfit and make it challenging to meet needs. That is why we employ transfer learning through pre-trained language models, which is currently one of the most popular techniques to reduce the need for a large amount of training data. And QA models based on BERT embedding and transformer architecture have demonstrated high accuracy in language understanding task.

We implement the BERT fine-tuned on SQuAD2.0. We used the BERT tokenizer to embed the word to 768 dimensions, then feed it into the BERT model layers which includes 12 BERT layers and a BERT pooling layer. Lastly, we connect it with the QA output layers which includes multiple fully connected layers, it takes the output from the BERT model and output the start token and end token through a SoftMax layer.

We modify the QA output layers and compared their results. At training stage, we conduct two type of training process. In the first process, we apply transfer learning, utilize all the pre-trained weights and freeze all these BERT layer weights, we only train the weights of the QA output layers on the dataset. In the second process we unfreeze all the weights and allow all the weights to be trained on the dataset.

One of the main challenges we encounter is the limitation in computing power, which are the constraints of RAM and GPU memory. We have to decrease the batch size to avoid RAM out of memory problem which leads to a volatile training loss graph. The out of memory issue for GPU limit us to the smaller BERT model with 12 BERT layers and 110M parameters, compare to the full BERT model has 24 BERT layers and 336M parameters.

4. Experiment

4.1. Dataset

We use SQuAD2.0, which stands for Stanford Question Answering Dataset. It is a reading comprehension dataset, consisting of questions posed by crowd workers on a set of Wikipedia articles. The dataset is stored in a json file, each data has a reading passage (context), a question and its corresponding answer, the answer to the question is a segment of text or span from the corresponding reading passage, some of the question might be unanswerable. In total the dataset contains 100,000 questions with answers and 50,000 unanswerable questions.

4.2. Preprocessing

We focus on the fine-tuning phase of BERT to train our model to perform question answering tasks based on sections of text. We add a neural network architecture on top of BERT with an output layer of 2 neurons with 768 dimensions (1024 for large BERT) and a softmax activation function.

We use pytorch-transformers modules from huggingface libraries to tokenize and produce word embeddings with an embedding dimension of 768 for our training data. The vocabulary size is 30522, which is the number of word em-

beddings for the model. The training data is passed into the model in question-passage pairs, the tokenized sequences of questions and passages are concatenated and separated by a SEP token. The resulting sequences are padded or truncated to a length of 256. A positional embedding and a segment embedding are produced for each word, denoting the word's position in the text and which sequence it originally belongs to, the question or the passage. The input is the sum of the pre-trained embedding, the positional embedding, and the segment embedding. A CLS token is placed at the beginning of the input array, indicating that the answer to the question cannot be located in the passage. The output layer of the neural network architecture has two neurons, each with a dimension of 768. As a token is passed through the network, the dot product between the input and the outer layer produces two scores, activated by a softmax function, yielded the probability of the token being the start and end position of the answer from the passage. The words or tokens with the highest probabilities are classified as the start and end position. And the subset of the passage between the two positions are returned as the answer to the question. In short, the output linear layer of the architecture has two outputs, the pointers to the start and the end of the answer to the question in the section of text.

4.3. Model Architectures

Since we have fixed the BERT-Layer architectures, we create different QA output layers structures in our experiment. We start with our baseline model, the original BertForQuestionAnswer architecture where the QA output layer consists only one linear layer of 768 neurons. The architectures we have experimented with are as follows:

- **Model A**: added an additional fully connected layer of 768 neurons to baseline model.
- **Model B**: added an additional fully connected layer of 1536 neurons to baseline model.
- **Model C**: added ReLU activation and dropout on the model A after the fully connected layer.
- **Model D**: added two fully connected layers of 1536 and 768 neurons with dropouts to the baseline model.
- **Model E**: added ReLU activation with dropout on the model D after each fully connected layer.
- **Model F**: added an additional Conv2D layer with kernel size of 3 and stride 1 to the baseline model.

4.4. Settings

We run our training process on Google Colab with 16G of GPU and 25G of RAM. We implement our model in PyTorch. We use decoupled weight decay regularization Adam Optimizer (AdamW optimizer)[3] in the training process. Because our model output the start token and the end token of the answer passage through SoftMax function, we use cross entropy loss as our loss function.

We import the BertForQuestionAnswering model from the PyTorch transformers implemented from the huggingface library, the model consists of two main layers as described in section 3, BERT-layer and QA output layer. For the training process we load the pre-trained BERT-layer weights from the 'bert-base-uncased' model for initialization and transfer learning purposes.

We import the official evaluation script for SQuAD2.0 to evaluate our result. And some of the starter code we used for visualizing results are originated from the *Question Answering with a Fine-Tuned BERT* blog by Chris McCormick.

4.5. Metric

In order to evaluate the performance of the models, following [5], we used exact match (EM) and F1 score. We compute these two metrics over subset of questions whose answers could be found in the paragraph, subset of questions whose answers could not be found in the paragraph without an answer, and also for the two subsets combined. For our project, we focus on EM and F1 for the entire dataset, that is the combination of both subsets.

Exact match (EM): Percentage of exact word match to ground truth.

F1: Average overlap between the prediction and ground truth answer. The prediction and ground truth are treated as bags of tokens so F1 score can be computed. We then take the maximum F1 over all the ground truth answers for a given question and then average over all the questions.

5. Result

Due to the RAM constraint, 32 is the max batch size we could trained our model with without running out of memory. We choose a batch size of 32 rather than 16 because according to our observation, larger batch size leads to more stable loss descent.

Models with frozen BERT layer weights take around 25 minutes to run, while models with no frozen layers take around 70 minutes. During our experiment we find that models converge with just one iteration, further increasing the number of epochs will not increase model performance. We tune the model manually due to long running time, our best performing set of parameters is learning rate of 5e-5, beta of (0.90, 0.999), and Adam epsilon of 1e-8.

When comparing the loss graphs, we find that models for which we are performing optimization on the BERT layers perform better than their counterparts with frozen BERT Layers. Besides, models with frozen layers tend to perform extremely poorly on answering questions but do very well

on making the correct prediction that the answer could not be located in the paragraph. The loss graphs of models with unfrozen layers in figure 2 also converge faster and have smoother descent than the loss graph of frozen layers model as shown in figure 1.

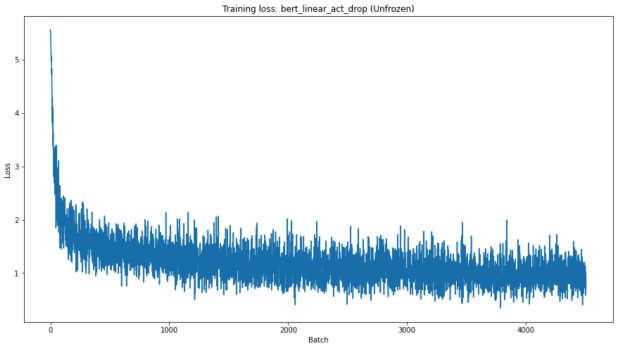

Figure 1. Loss Graph unfrozen-weight model

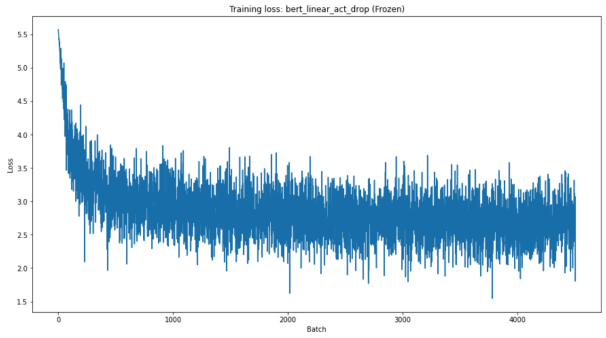

Figure 2. Loss Graph frozen-weight model

We compute the evaluation metrics for all models on the evaluation dataset as shown on table 2. When comparing the model performance, we find that the models trained on transfer learning with frozen BERT layer weight performs poorly in comparison to models with unfrozen layer weights. One possible reason of underperformance of the transfer learning model is that the BERT-Layer weight from the pre-trained model might not be specified to the question and answering task.

Our best performance model is the model with two fully-connected linear layers with ReLU activation and a dropout layer. The exact match (EM) score is 72.5 and F1 is 75.8.

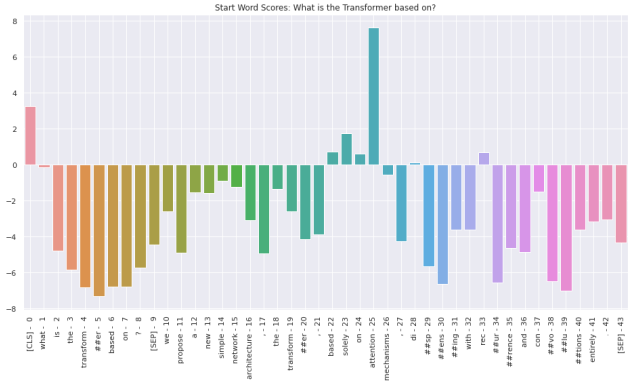

Figure 3. Start Score of each token.

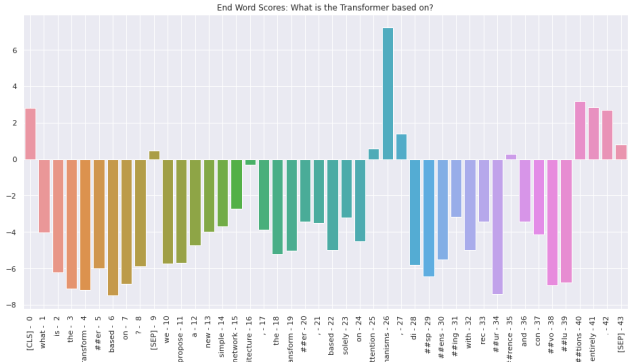

Figure 4. End Score of each token.

6. Visualization and Exploration

6.1. Apply Model to Novel Data

Today's pretrained language models are trained on massive, heterogeneous corpora. BERT was trained on BooksCorpus and Wikipedia. They have worked universally well in NLP tasks. To better understand how our QA model performs outside SQuAD, we showcase some question-answering pairs generated by our model on novel data. We use the abstract of "Attention is all you need[6]" as reference context and ask several questions.

Q: "What is the Transformer based on?"
A: "attention mechanisms"
Q: "How many BLEU do we achieve?"
A: "28"

We find that our model generalizes well in this Deep Learning literature. It can answer factoid questions properly. The model makes a minor mistake while answering numeric questions. It failed to detect the ".4" in "28.4". This can be improved by using larger BERT tokenizer fine tuned on SQuAD.

These are some questions that the model failed.

Table 2. QA task on SQuAD: EM and F1 scores on fine-tuned models.

Model	EM	F1	EM (with Ans)	F1 (with Ans)	EM (w/o Ans)	F1 (w/o Ans)
Baseline unfrozen	68.83	72.17	**68.54**	**75.23**	69.12	69.12
A unfrozen	70.58	73.73	65.42	71.72	75.73	75.73
B unfrozen	71.34	74.48	67.02	73.32	75.64	75.64
C unfrozen	**72.50**	**75.80**	66.68	73.30	78.30	78.30
D unfrozen	71.48	74.77	65.28	71.88	77.66	77.66
E unfrozen	71.10	74.28	66.88	73.25	75.31	75.31
F unfrozen	70.51	73.37	61.69	67.68	79.04	79.04
Baseline frozen	49.24	49.26	0.10	0.14	**98.23**	98.12
A frozen	47.65	48.05	3.15	3.94	92.03	92.03
B frozen	47.00	47.48	3.71	4.67	90.16	90.16
C frozen	49.36	49.62	2.78	3.31	95.81	95.81
D frozen	47.33	47.79	3.13	4.05	91.40	91.40
E frozen	46.74	47.46	8.25	9.68	85.13	85.13
F frozen	49.20	49.22	0.12	0.16	98.14	**98.15**

Q: "What machine translation tasks do we practice?"

A: "No answer found"

The golden answer is "English-to-German translation task" and "English-to-French translation task". The reported model performance on two machine translations tasks, "English-to-German translation task" and "English-to-French translation task", in two separate sentence. The model failed to detect them and returned "No answer found", which suggests the model is not good at capturing distant content.

Q: "What strengths of these models do the experiments show?"

A: "superior in quality"

The relevant context is "Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train." The golden answer is "more parallelizable and requiring significantly less time to train", while the model returned "superior in quality". The model did not catch that "strengths" is similar to "superior in quality" and failed to comprehend the question. Addressing expressivity and ambiguity of human language remains a challenge for QA systems.

6.2. Visualizing Start and End Scores

We visualize the start and end scores for every token in the input. For example, We showcase start and end scores of each tokens given a span from [6]: *We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely.* The question is *"What is the Transformer based on?"*

The start and end scores of each token in the above span is as Figure 3 and Figure 4. An alternative view which shows start and end scores together is as Figure 5.

7. Conclusion and Further Thoughts

In our project, we are able to successfully develop a question answering bot with language understanding capability using BERT and customized output layers. We start with a pre-trained BERT architecture and fine-tuned it to perform question answering tasks by adding output layers on top of BERT. Our best performing model is two fully-connected linear layers with ReLU activation and a dropout layer. It out performs the other models with an ExactMatch score of 72.50 and a Macro-averaged F1 score of 75.8. We discover that models with unfrozen layers drastically outperformed models in terms of ExactMatch and F1 scores, their training loss appear to be much less volatile. Another interesting finding is that models with frozen layers perform very poorly on answering questions but do well on prediction that the answer cannot be located in the passage,

Some limitations of our models include capturing distant information, question comprehension and recognizing numbers. It can be further improved by building bigger model, training the model longer and using fine-tuned tokenizer. Besides, we believe can boost model performance using a larger BERT architecture with 24 layers as well as other types of pre-train models, but due to computational limitation, we are unable to undertake the experiment in our Google Colab environment. But it would be a great direction for our future studies. We would also try different architectures such as RoBERTa, ALBERT, Match-LSTM, and ensemble methods to see if it can exceed our current results.

8. Deliverable

Our code repository for this project is available at:

https://github.com/markjluo/
DL-Project-A-Deep-Learning-Based-Chatbot

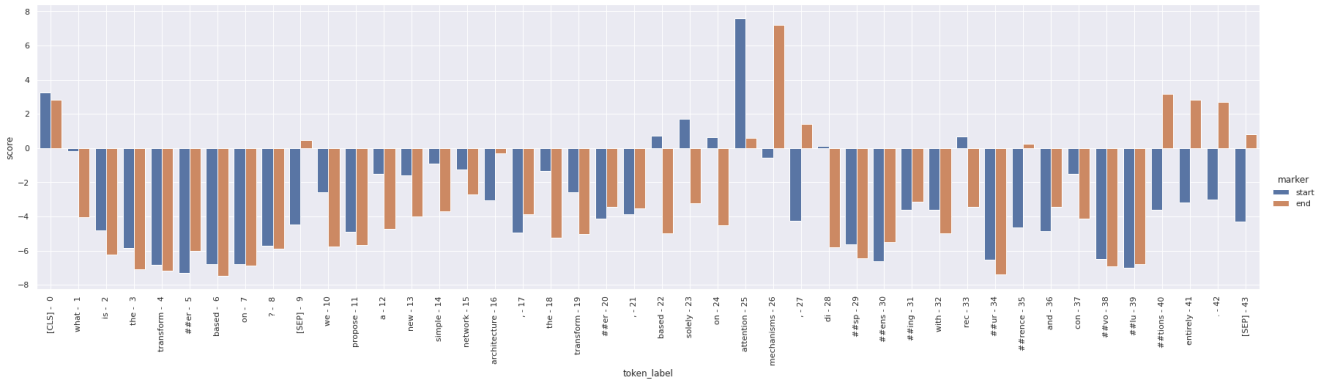

Figure 5. Alternative view of start and end scores

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[3] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[4] Tittaya Mairittha, Nattaya Mairittha, and Sozo Inoue. Improving fine-tuned question answering models for electronic health records. In *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*, pages 688–691, 2020.

[5] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[7] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.