CPSC 8430 DEEP LEARNING HOMEWORK 3 REPORT EXTRACTIVE QUESTION ANSWERING

SATHWIKA ADIMULAM
Dr. Feng Luo

GitHub link:- https://github.com/sathwika-adimulam/DEEPLEARNING_HW3

Bonus Features Implemented

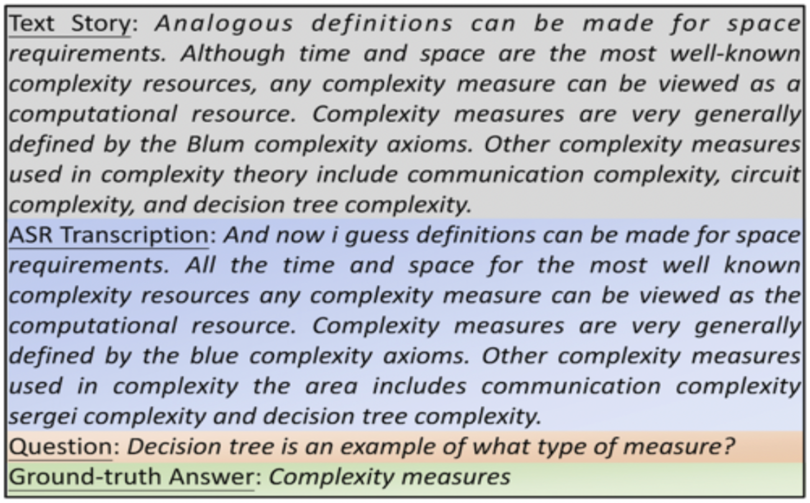
1. Automatic learning rate decay
2. Changing doc stride
3. Automatic mixed precision
4. Gradient accumulation
5. Post processing
6. Other pretraining models

Problem Statement

The objective is to create an extractive question-answering model that is effective and efficient and can increase the speed and accuracy of question-answering systems. The model's performance and effectiveness will be examined using standard datasets.

Dataset

The original SQuAD dataset is included in the Speaking SQuAD dataset. The document is in spoken form, the input question is in text form, and the answer is always a span in the document. The SQuAD articles were spoken using the Google text-to-speech system to create the Speaking SQuAD dataset, and the associated ASR transcriptions were created using CMU Sphinx. The testing set was produced from the SQuAD development set, while the Spoken SQuAD training set was produced from the SQuAD training set. The dataset consists of 5,351 pairs in the testing set and 37,079 pairs in the training set, with a word error rate of 22.77% for the training set and 22.73% for the testing set, respectively. The testing set's audio files had white noise added to them at two different levels, resulting in various word mistake rates, to evaluate the model's comprehension under worse audio quality. The question-answer pair was dropped from the dataset if the answer was not included in the ASR transcriptions of the related article. A sample SQuAD dataset is shown below.



Text Story: *Analogous definitions can be made for space requirements. Although time and space are the most well-known complexity resources, any complexity measure can be viewed as a computational resource. Complexity measures are very generally defined by the Blum complexity axioms. Other complexity measures used in complexity theory include communication complexity, circuit complexity, and decision tree complexity.*

ASR Transcription: *And now i guess definitions can be made for space requirements. All the time and space for the most well known complexity resources any complexity measure can be viewed as the computational resource. Complexity measures are very generally defined by the blue complexity axioms. Other complexity measures used in complexity the area includes communication complexity sergei complexity and decision tree complexity.*

Question: *Decision tree is an example of what type of measure?*

Ground-truth Answer: *Complexity measures*

Fig 1: SQUAD dataset sample

Abstract

A core problem in natural language processing is extractive question answering, which has several uses in industries like chatbots, virtual assistants, and search engines. Much progress has been made in this area as a result of recent developments in deep learning, attention-based systems, and transformer-based architectures like BERT. Despite this, extractive question answering is still difficult, especially when dealing with lengthy paragraphs and difficult queries. Also, a significant amount of processing power and specialized knowledge are needed for the BERT question-answering fine-tuning procedure. The SQUAD dataset, a benchmark dataset for extractive question answering in natural language processing, is the setting for this study's investigation into the performance of BERT. We will pay particular attention to the difficulties associated with tokenization and dealing with lengthy paragraphs because these issues are well known to have a significant impact on how well extractive question-answering models work. We will also investigate how various hyperparameters affect the BERT model's performance.

Requirements

1.Python
2.Torch
3.Scipy
4.Numpy
5.Pandas
6.SQuAD dataset

Loading the JSON dataset

The SQuAD dataset must be loaded from the JSON file using a JSON parser in order to transform it from JSON to CSV format. Once the dataset has been loaded, the relevant data, including the article ID, title, paragraph text, question text, and response text, can be extracted by iterating through the data. Afterwards, this data can be arranged into a CSV file, where each row corresponds to a single question-answer pair and the columns have the pertinent data. After that, the generated CSV file can be utilized as training and testing data for extractive question-answering models.

Model Execution

Base model

Tokenizing the input text, which involves dividing the text into smaller subunits like as individual words or subwords, is the first stage in employing BERT for extractive question answering. BERT tokenizes text into subwords using the WordPiece tokenizer in order to accomplish this. The BERT model then analyzes the subword tokens and produces embeddings for each token using the resulting subword tokens as input. These embeddings are used to locate the portion of the input text that corresponds to the response to the query and record the contextual meaning of each token in the input text.

For a dataset of question-answer pairs, such as the SQuAD dataset, the BERT model is then refined. The model is trained to recognize the start and end positions of the response in the passage when it is given a question and a passage of text that corresponds to it during training. This is accomplished by applying a softmax layer to the output embeddings that the BERT model produces. The start and end places of the answer in the input text are determined using the resultant probability.

Managing lengthy document, where it is more difficult to identify the relevant response, is one of the main challenges in extractive QA. Due to its capacity to process the full document as a single sequence, BERT has proven to be efficient at handling lengthy documents. This problem can be solved by using methods like sliding windows to divide the input text into more understandable chunks. The BERT model can then be used to construct the result by taking each of these smaller components and feeding it separately. It is possible to create a powerful extractive question answering system by optimizing a pre-trained BERT model on a dataset of question-answer pairs and assessing its performance.

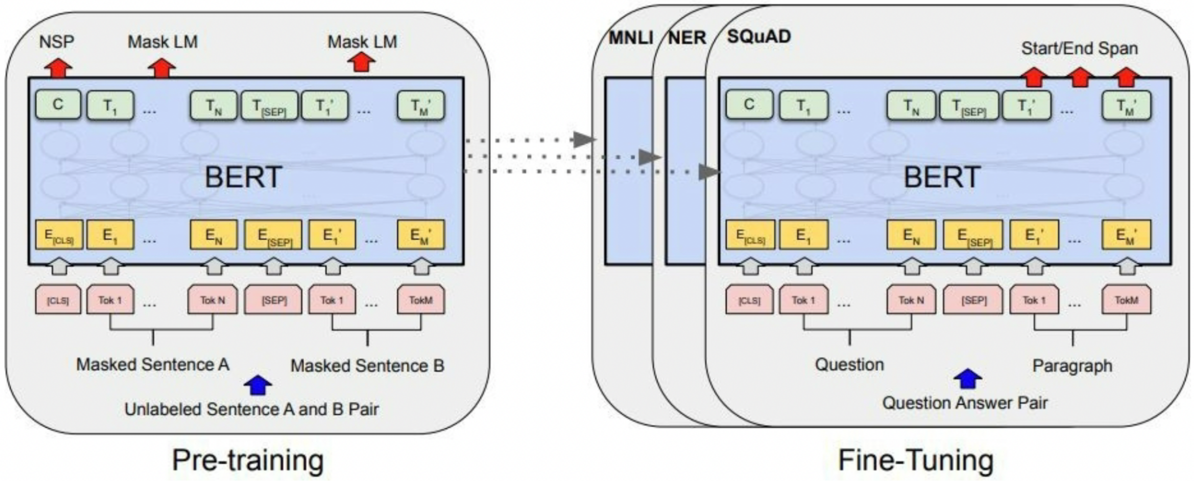
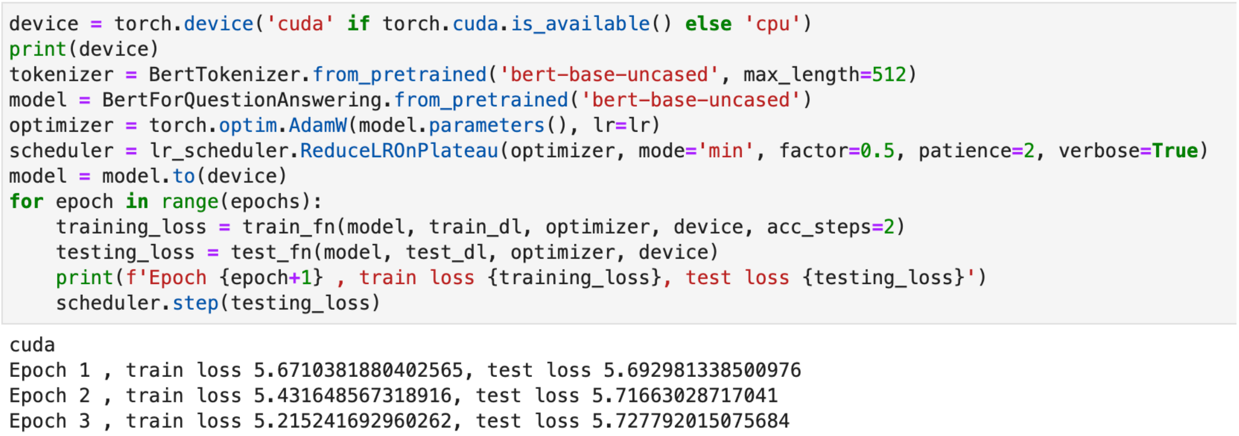


Figure 2: BERT MODEL

Results

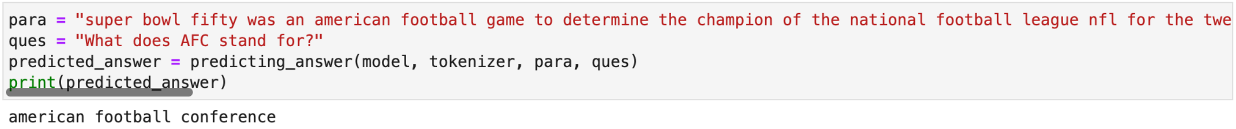
Here are the results obtained using the BERT method.



```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', max_length=512)
model = BertForQuestionAnswering.from_pretrained('bert-base-uncased')
optimizer = torch.optim.AdamW(model.parameters(), lr=lr)
scheduler = lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=2, verbose=True)
model = model.to(device)
for epoch in range(epochs):
    training_loss = train_fn(model, train_dl, optimizer, device, acc_steps=2)
    testing_loss = test_fn(model, test_dl, optimizer, device)
    print(f'Epoch {epoch+1} , train loss {training_loss}, test loss {testing_loss}')
    scheduler.step(testing_loss)
```

```
cuda
Epoch 1 , train loss 5.6710381880402565, test loss 5.692981338500976
Epoch 2 , train loss 5.431648567318916, test loss 5.71663028717041
Epoch 3 , train loss 5.215241692960262, test loss 5.727792015075684
```

Figure 3: Training and Testing loss



```
para = "super bowl fifty was an american football game to determine the champion of the national football league nfl for the twe
ques = "What does AFC stand for?"
predicted_answer = predicting_answer(model, tokenizer, para, ques)
print(predicted_answer)
```

```
american football conference
```

Figure 4: Predicted answer from sample case

Bonus Features Implemented

1. Automatic learning rate decay: This technique automatically reduces the learning rate during training to achieve better performance. In BERT, a warmup step is also used to gradually increase the learning rate at the beginning of training.

2. Changing doc stride: The doc stride is the number of tokens the sliding window moves when selecting a new context. By changing the doc stride, we can control how much overlap there is between different contexts, which can affect the model's performance and training time.

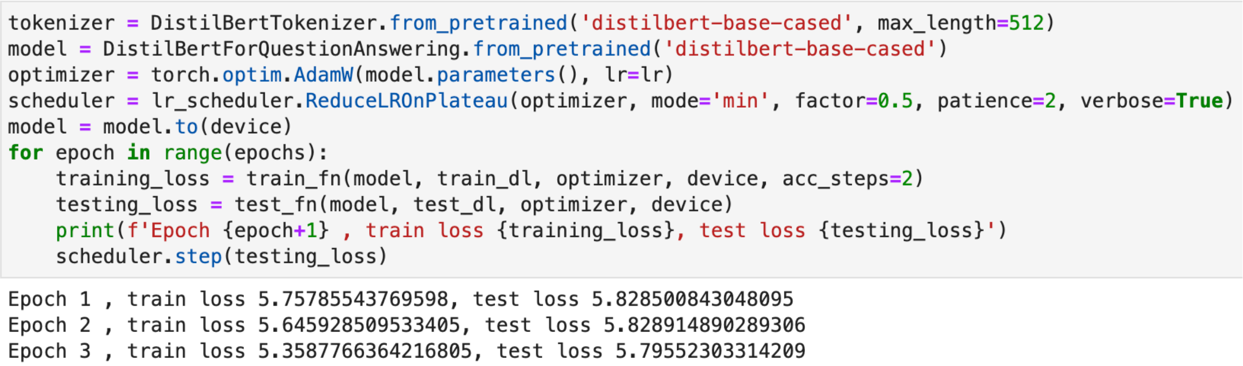
3. Automatic mixed precision: This is a technique that allows us to use lower-precision (e.g., half-precision) floating-point arithmetic to speed up the training process, while maintaining the accuracy of the model. BERT supports mixed-precision training with the Apex library.

4. Gradient accumulation: This technique involves accumulating gradients over multiple batches before updating the model weights. This can help reduce memory usage during training, as well as allowing us to use larger batch sizes without running out of memory.

5. Post processing: After the model has generated predictions for a given input, we can perform additional post-processing steps to improve the accuracy of the predictions.

6. Other pretraining models: While BERT is currently the most widely used pretraining model for extractive question answering, there are other pretraining models that may also be effective for this task. Examples include DistilBERT. Each of these models has its own strengths and weaknesses and may be better suited for different types of inputs or tasks.

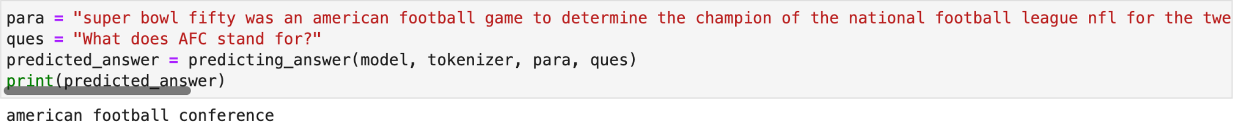
Below are the results obtained using DistilBERT method.



```
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-cased', max_length=512)
model = DistilBertForQuestionAnswering.from_pretrained('distilbert-base-cased')
optimizer = torch.optim.AdamW(model.parameters(), lr=lr)
scheduler = lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=2, verbose=True)
model = model.to(device)
for epoch in range(epochs):
    training_loss = train_fn(model, train_dl, optimizer, device, acc_steps=2)
    testing_loss = test_fn(model, test_dl, optimizer, device)
    print(f'Epoch {epoch+1} , train loss {training_loss}, test loss {testing_loss}')
    scheduler.step(testing_loss)
```

```
Epoch 1 , train loss 5.75785543769598, test loss 5.828500843048095
Epoch 2 , train loss 5.645928509533405, test loss 5.828914890289306
Epoch 3 , train loss 5.3587766364216805, test loss 5.79552303314209
```

Figure 5: Training and Testing loss



```
para = "super bowl fifty was an american football game to determine the champion of the national football league nfl for the twe
ques = "What does AFC stand for?"
predicted_answer = predicting_answer(model, tokenizer, para, ques)
print(predicted_answer)
```

```
american football conference
```

Figure 6: Predicted answer from sample case

Conclusion

The training and testing loss for extractive question answering were better in BERT method compared to distilBERT method.