Deep Learning: CPSC 8430
Homework 3

-Sandeep Varma Sangaraju

GitHub Link:
https://github.com/sangaraju-01/Deep-learning-HW-3

INTRODUCTION:

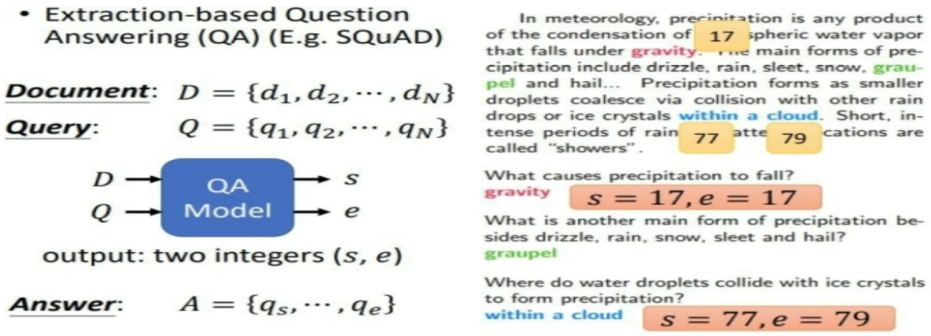
In this DeepLearning task, we are going to perform and train the **SQuAD-** Stanford Question Answering Dataset to solve the Question and Answering pairs that are in the dataset by using **BERT(Bidirectional Encoder Representations from Transformers)** model. Natural language processing presents a basic barrier in answering questions, which is why artificial intelligence and other Deep learning have models long focused on this problem.

These technologies enable users to ask questions in natural language and receive prompt, concise responses. QA(Question and Answering) systems are often used in conversational user interfaces and search engines, and they are usually effective at answering simple questions. Reading comprehension refers to the capacity to comprehend a text and then respond to inquiries about it. Because reading comprehension involves a grasp of natural language and familiarity with the outside world, it is challenging for machines to perform. For more challenging questions, they frequently only present the user with possible answers.

In this process, we have used the SQuAD dataset to train the model by using the BERT model. Usually, this dataset contains pairs of questions and answers. By using Tokenization, we will divide the sentence into chunks then it will easy to train the dataset.

Extractive based Question and Answering:

In this Task, we will be using extractive based QA.



Task Variants

There are different QA variants based on the inputs and outputs:

- **Extractive QA:** The model **extracts** the answer from a context. The context here could be a provided text, a table or even HTML! This is usually solved with BERT-like models.
- **Open Generative QA:** The model **generates** free text directly based on the context. You can learn more about the Text Generation task in its page.
- **Closed Generative QA:** In this case, no context is provided. The answer is completely generated by a model.

In the above Screenshot, we can three different QA variants, but in this task, we have used Extractive-based Question Answering.

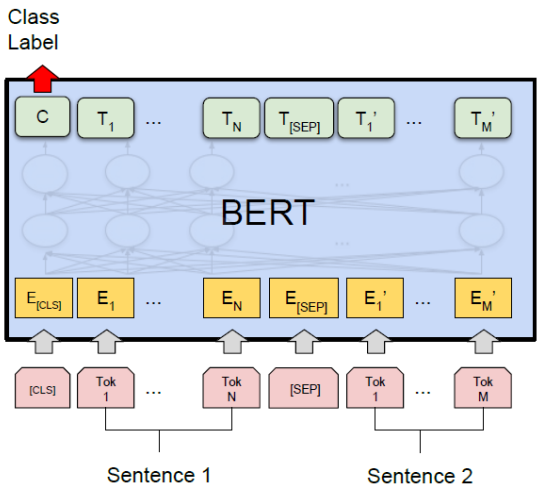
Problem Statement:

In this task, the given problem is to create and build a deep-learning language model for that. We have used the SQuAD dataset's question and Answer model with the BERT model's help. In this task, We are creating a model for answering questions that accept a question and a sentence of text as input and outputs an answer that can be found in the provided text. We are particularly training and testing our model on the SQuAD dataset (Spoken-SquAD).

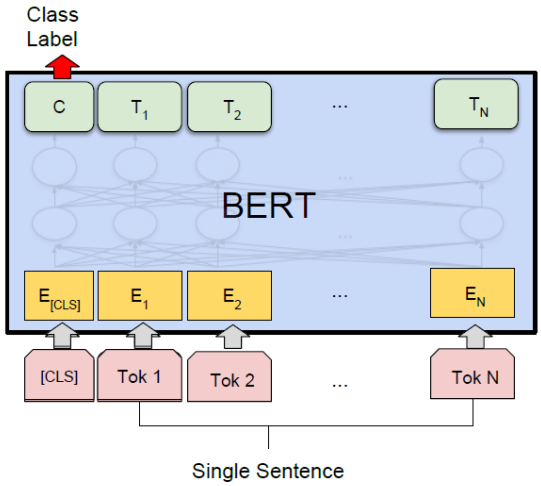
Model Used for this Task:

For this task, I have used the BERT**(BiDirectional Encoder Representations)** model is a kind of language model that uses the context to the left and right of each word in a phrase to help students develop a thorough knowledge of the language. To do this, a vast volume of unlabeled text is used to train the model and teach it the links between words and phrases. Consequently, BERT may be adjusted for certain natural language processing tasks, such as text categorization or question answering, producing incredibly accurate results.

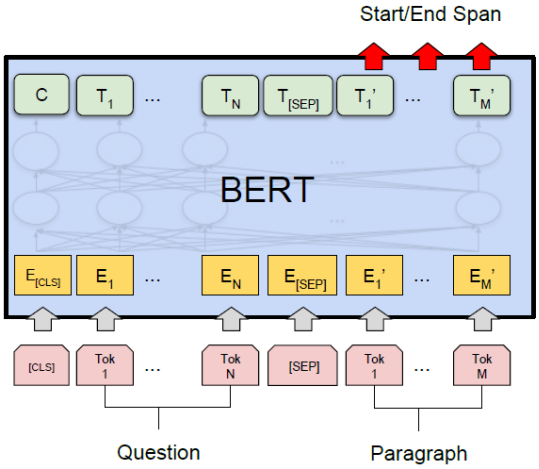
BERT uses the transformer architecture to create context word embeddings for specific natural language processing (NLP) applications, including text categorization, named entity identification, and question answering. The BERT model may be further customized for certain NLP tasks after training by adding a task-specific output layer to the pre-trained model and refining it using relevant data.



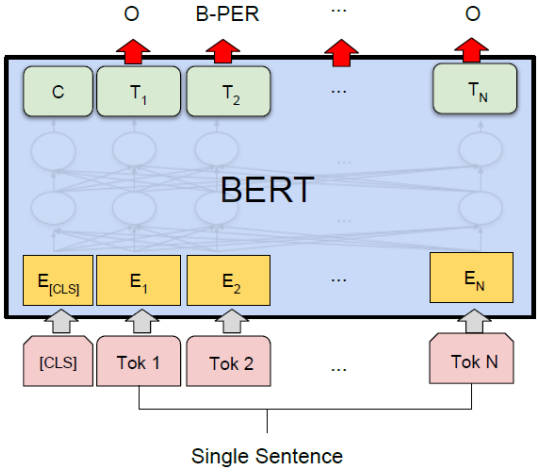
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(b) Single Sentence Classification Tasks: SST-2, CoLA



(c) Question Answering Tasks: SQuAD v1.1



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Requirements For this Task:

In this Task for building an Extractive based Question Answering, we have used some packages and Libraries and they are:

1. **Python**: To use Python, the computer must have Python installed.
2. **Pytorch**: we must install PyTorch in order to load and modify the BERT model because PyTorch is commonly used to construct BERT models.

3. **Tokenizer**: It is important to tokenize the input text using a particular tokenizer in order to get it ready for processing by a BERT model. During this procedure, the input text is changed into a series of tokens that the model can comprehend. A number of language models, notably BERT, include built-in tokenizers in the Hugging Face Transformers library.
4. **Hugging Face Transformers**: For working with pre-trained language models like BERT, the Hugging Face Transformers library offers a simple interface. To load and adjust the BERT model, we must install this library.
5. **Dataset**: To train and test our QA model,we require a dataset of questions and responses. Several datasets are accessible, including SQuAD (Stanford Question Answering Dataset), which has over 100,000 question-answer pairings.
6. **GPU**: It can be computationally expensive to fine-tune a BERT model on a big dataset, therefore, we need to employ a GPU to speed up the training process that's why we used Palmetto.

DATASET:

A popular benchmark for creating and assessing question-answering systems is the SQuAD dataset, often known as the Stanford Question Answering Dataset. It comprises over 100,000 question-answer pairs that a team of annotators extracted from Wikipedia articles. Building a machine learning model with the ability to understand the text and offer a precise answer to the question is the goal. The SQuAD dataset was converted into audio files using a two-step procedure. The written content was first translated into spoken English using Google's text-to-speech technology.

The spoken content was then converted back into text using CMU Sphinx. With the spoken SQuAD training and development sets, we created the Speaking SQuAD training and testing sets using these transcriptions. There are 5,351 question-answer pairs in the evaluation set and 37,111 in the training set of the SQuAD dataset. A word mistake rate of 22.77% is present in the training set. We added two levels of white noise to the audio files in the testing set, resulting in various word mistake rates, to test the model's performance in real-world scenarios with subpar audio quality. SQuAD has played a significant role in the development of natural language processing by enabling researchers to create models that are very precise at performing difficult linguistic tasks, such as question-answering.

Approach:

The approach for this task to train a BERT model, the script must read information from two JSON files that comprise questions, context, and responses. Before building a BERT model, the data is first read, preprocessed, and tokenized. A learning rate schedule is included in the code, and AdamW is used to optimize the model. The accuracy of the predictions is determined once the BERT model has been trained and confirmed. Saving the model is the last action.

The code will operate with two JSON files called "spoken train-v1.1.json" and "spoken test-v1.1.json," which are probably data for the model's training and testing. The code creates histograms of the length distribution for the context and question texts after importing the data. Moreover, it provides the route for the BERT model and the maximum sequence length.

EXECUTION:

The code designates a function called "get data" that retrieves data from the JSON file using a file path as input. Following that, it returns many pieces of data, including the total number of questions, the total number of questions that can be answered and those that cannot, as well as the contexts, questions, and answers.

Using the 'get data' method, the program initially loads the training and validation data from the appropriate JSON files. The replies in the data are changed to lowercase, and an additional attribute called "answer end" is introduced.

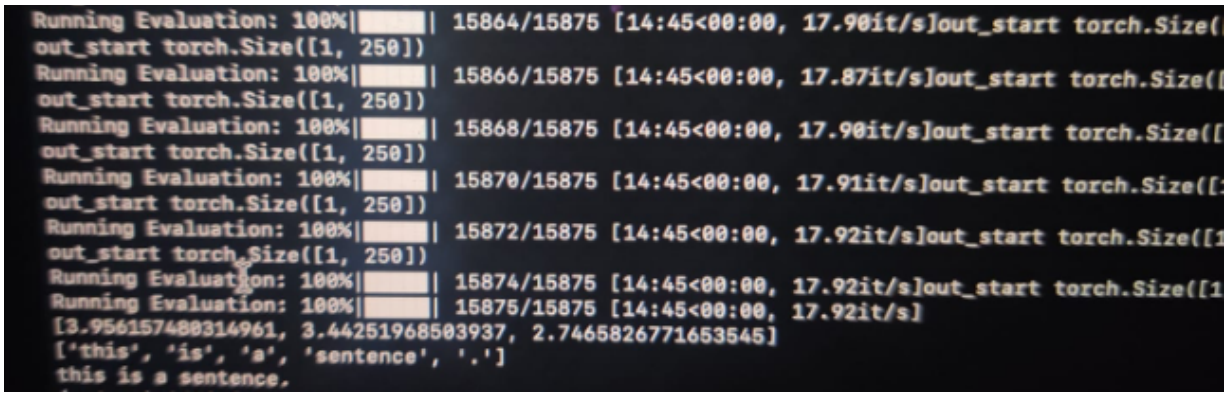
The BERT model path is set to "Bert-base-uncased," and the maximum context length is 250 next in the code. With a maximum length of 250, truncation set to True, and padding set to True, the questions and contexts are tokenized using the BertTokenizerFast.

The WER (word error rate) measure is used to assess the model's performance, and the scores are saved in a list named "wer list" at the end of each epoch. The code also outputs the training accuracy and loss throughout each epoch. As a result, the WER score is the accuracy metric utilized in this code, with lower values indicating greater performance by the model.

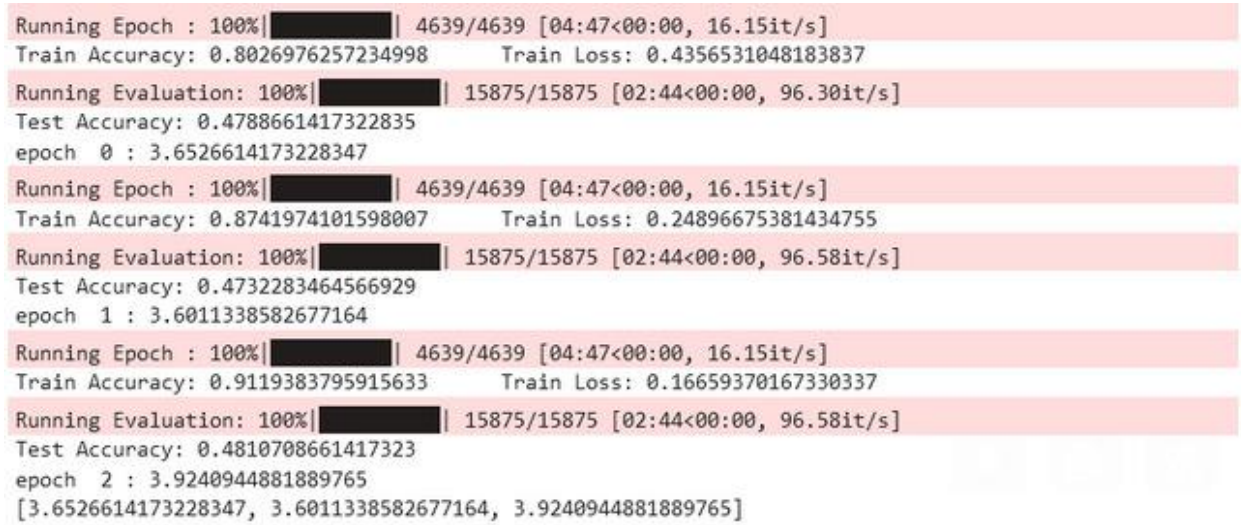
RESULT:

We can see the result for this task in below Screenshots.

In this Task, We will use an existing model from Hugging Face to build a system that can respond to queries. The system will use this model to comprehend the current context and produce pertinent responses to queries.

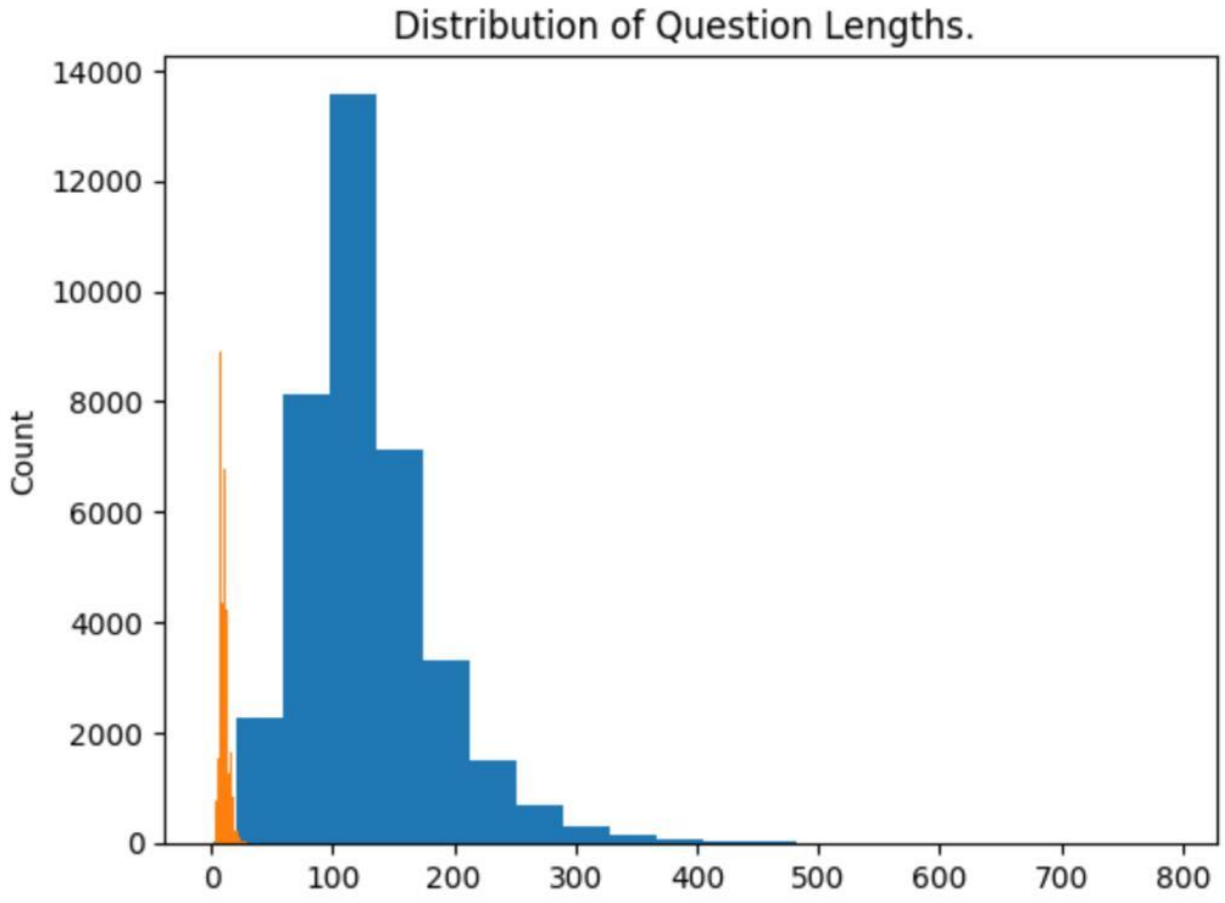


Pre-processing the input text ensures that a query-and-answer model can comprehend a question and provide pertinent answers. One efficient method is "doc stride," which entails segmenting the original document (context) into smaller chunks and generating replies for each area individually.



A pre-processing model's training accuracy and loss are shown in the diagram. The software reports the accuracy and loss for the training set for each epoch and evaluates the model's effectiveness using the WER metric, which determines the word error rate. A list is named, and the list is then updated with the WER scores. In the below output, we have used Tokenization, which produced the following set of tokens: ['this', 'is', 'a', 'sentence', '.'].

```
[4.189417322834646, 4.934740157480315, 3.7332913385826774]
['this', 'is', 'a', 'sentence', '.']
this is a sentence.
```



The values for the finished text are shown in the screenshot above. The distribution of question length must be considered while evaluating and enhancing a question-answering system in light of these parameters.