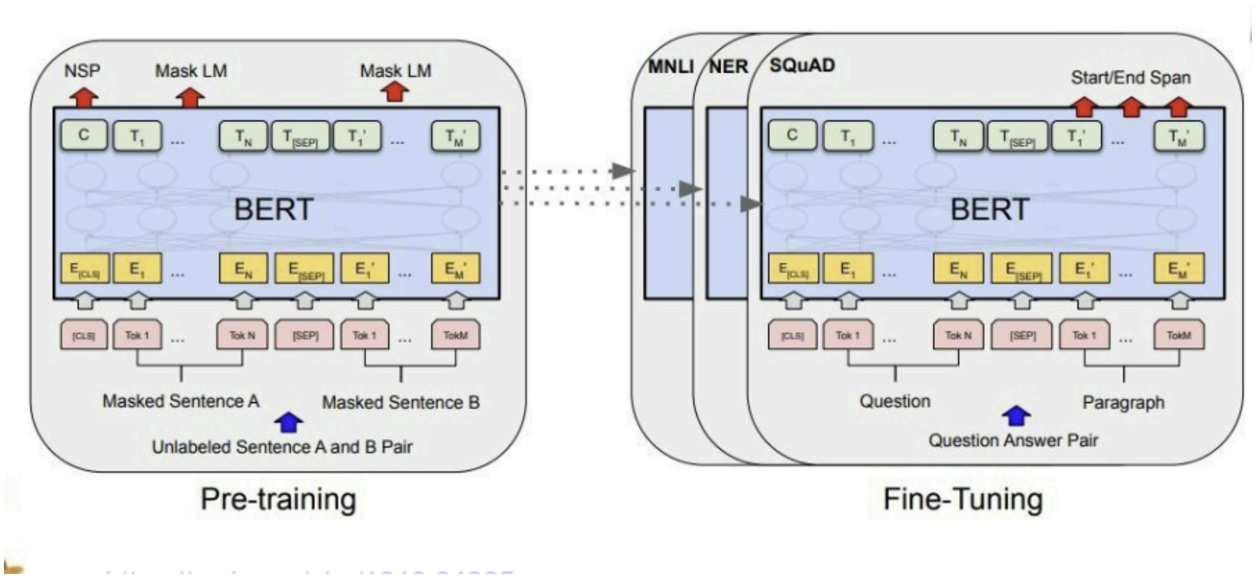
CPSC 8430: Deep Learning

Assignment 3

Saurabh Sharma
03/23/2024

1. About the model

BERT is a Deep Learning language model used for Natural Language Processing. It is an open-sourced architecture and it stands for Bidirectional Encoder Representations from Transformers. Fundamentally, BERT uses a transformer architecture, a kind of neural network architecture intended to handle textual material that is presented sequentially. By using a bidirectional method, BERT is able to analyse the left and right context of each word at the same time, in contrast to typical models that analyse text in a sequential manner. The "Masked Language Model" (MLM) and the "Next Sentence Prediction" (NSP) task during pre-training are the two main processes that enable this bidirectional comprehension. Each layer of the transformers that make up BERT is made up of feed-forward neural networks and self-attention processes. By encoding contextual information about each word in a phrase, these transformers allow BERT to produce high-quality language representations that accurately capture complex semantic connections.



Google has released a pre-trained BERT model and all the NLP researchers train this pre trained BERT model on their Dataset to optimize it to their domain. This is called transfer learning and it has proved very efficient and useful in the real of Deep Learning and Generative artificial neural networks prominently.

2. About the Dataset

We are going to use Spoken-SQuAD dataset. The Spoken-SQuAD dataset is a corpus specifically designed for training and evaluating spoken question answering systems. It comprises audio recordings of user-generated questions along with corresponding spoken answers. Each interaction in the dataset is transcribed into text format, creating pairs of questions and answers. The dataset covers a wide range of topics and domains, providing a diverse set of linguistic contexts and accents encountered in natural conversational settings.

Each instance in the Spoken-SQuAD dataset consists of multiple components:

1. Audio Recordings: These are recordings of users asking questions verbally. The audio files capture various aspects of spoken language, including intonation, rhythm, and pronunciation. These recordings simulate real-world scenarios where users interact with question answering systems using speech.

2. Transcriptions: The audio recordings are transcribed into text, providing a written representation of the spoken questions and answers. These transcriptions include any linguistic modifications made during speech, such as corrections, hesitations, and disfluencies, to maintain the authenticity of the spoken interactions.

3. Question-Answer Pairs: Each instance in the dataset consists of a pair of a question and its corresponding answer. These pairs form the basis for training and evaluating spoken question answering systems. The questions cover a wide range of topics and may vary in complexity and length, while the answers provide relevant information or responses to the questions posed.

4. Annotations: The dataset may include annotations or metadata associated with each instance, such as the topic or domain of the question, speaker demographics, or additional contextual information. These annotations provide insights into the characteristics of the spoken interactions and can aid in the development and evaluation of question answering models.

Researchers often use the Spoken-SQuAD dataset to train machine learning models to understand and respond to spoken questions accurately. By leveraging the audio recordings and transcriptions, models can learn to recognize diverse accents, speech patterns, and linguistic nuances present in natural conversations. Additionally, the dataset serves as a benchmark for evaluating the performance of spoken question answering systems, enabling researchers to measure accuracy, fluency, and comprehension across different tasks and domains.

And this is exactly what we are going to do here. We are going to train BERT to give out answers to spoken questions.

3. Evaluation Metric

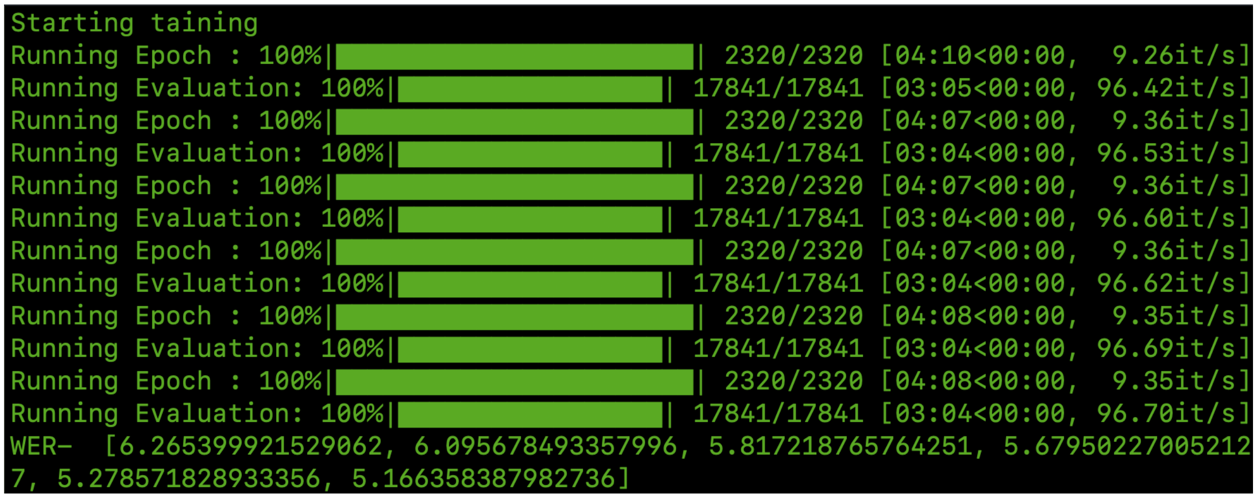
In our project, we employ Word Error Rate (WER) as a key metric to assess the accuracy and performance of our automatic speech recognition (ASR) system. WER quantifies the disparity between the transcribed output generated by our ASR system and the reference transcript. It measures the percentage of words that are incorrectly recognized, taking into account substitutions, deletions, and insertions. By utilizing WER, we gain valuable insights into the effectiveness of our ASR system in accurately converting spoken utterances into text.

To calculate WER, we tally the total number of errors, including substitutions (words that are incorrectly recognized), deletions (words that are missed), and insertions (words that are incorrectly added). We then divide this total by the number of words in the reference transcript. The resulting ratio is then multiplied by 100 to express the error rate as a percentage. This metric allows us to objectively evaluate the performance of our ASR system across various datasets, languages, and speech styles. By tracking WER throughout our project, we can identify areas for improvement and refine our ASR system to achieve higher levels of accuracy and reliability.

4. Base BERT model performance

Here we used a simple BERT model without any optimizations done to it. We downloaded the model from huggingface and it is found in my github repository as BERT_BASE.py.

We only ran it for 6 epochs because BERT is a very deep model and takes a lot of time to train. The below screenshot shows the output performance in terms of WERs of the baseline model after 6 epochs. We ran the model on Clemson’s Palmetto supercomputer cluster.



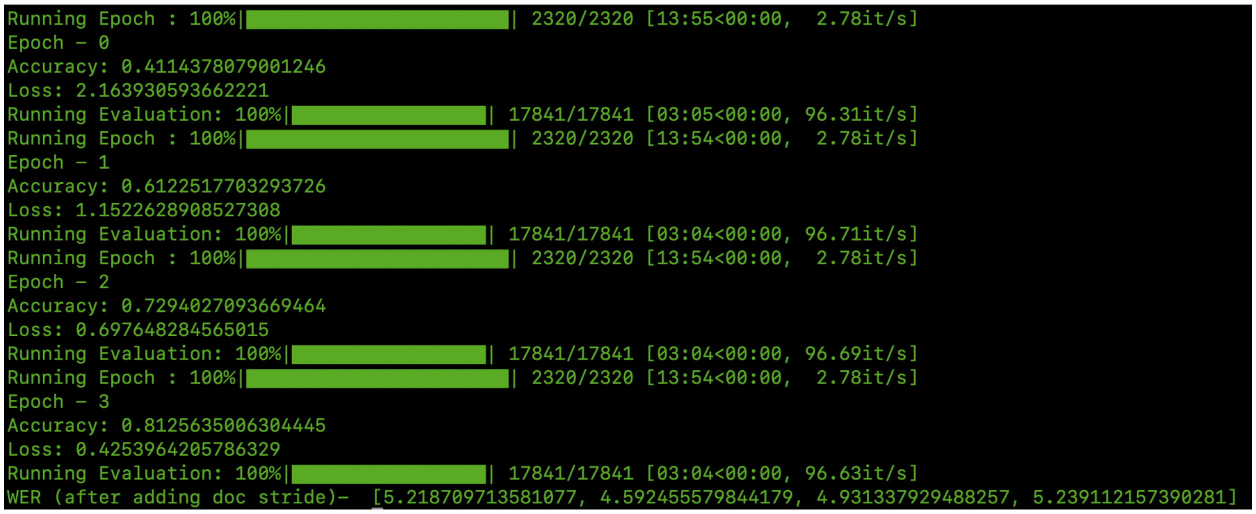
```
Starting taining
Running Epoch : 100%|██████████████████████████████| 2320/2320 [04:10<00:00,  9.26it/s]
Running Evaluation: 100%|██████████████████████| 17841/17841 [03:05<00:00, 96.42it/s]
Running Epoch : 100%|██████████████████████████████| 2320/2320 [04:07<00:00,  9.36it/s]
Running Evaluation: 100%|██████████████████████| 17841/17841 [03:04<00:00, 96.53it/s]
Running Epoch : 100%|██████████████████████████████| 2320/2320 [04:07<00:00,  9.36it/s]
Running Evaluation: 100%|██████████████████████| 17841/17841 [03:04<00:00, 96.60it/s]
Running Epoch : 100%|██████████████████████████████| 2320/2320 [04:07<00:00,  9.36it/s]
Running Evaluation: 100%|██████████████████████| 17841/17841 [03:04<00:00, 96.62it/s]
Running Epoch : 100%|██████████████████████████████| 2320/2320 [04:08<00:00,  9.35it/s]
Running Evaluation: 100%|██████████████████████| 17841/17841 [03:04<00:00, 96.69it/s]
Running Epoch : 100%|██████████████████████████████| 2320/2320 [04:08<00:00,  9.35it/s]
Running Evaluation: 100%|██████████████████████| 17841/17841 [03:04<00:00, 96.70it/s]
WER-  [6.265399921529062, 6.095678493357996, 5.817218765764251, 5.67950227005212
7, 5.278571828933356, 5.166358387982736]
```

Now we will try different optimizations to improve the performance of our BERT model on the Spoken-SQuAD dataset.

5. Updating the doc_stride

Now we will try changing the doc_stride value to improve the performance. Doc_stride is the distance between the start or end positions of two consecutive windows. In the baseline model the doc_stride is given as 150 words and that means that the whole paragraph comes under one window. Now the issue is what if the important information at either end of the window then it doesn't get processed effectively. To solve this, we will use a shorter doc_stride aka the window size and make it sliding window so that every important region of the paragraph comes at the center of the window at some point.

Below is the output and we can clearly see that the performance was improved to a great extent.



```
Running Epoch : 100%|██████████████████████████████| 2320/2320 [13:55<00:00,  2.78it/s]
Epoch - 0
Accuracy: 0.4114378079001246
Loss: 2.163930593662221
Running Evaluation: 100%|██████████████████████| 17841/17841 [03:05<00:00, 96.31it/s]
Running Epoch : 100%|██████████████████████████████| 2320/2320 [13:54<00:00,  2.78it/s]
Epoch - 1
Accuracy: 0.6122517703293726
Loss: 1.1522628908527308
Running Evaluation: 100%|██████████████████████| 17841/17841 [03:04<00:00, 96.71it/s]
Running Epoch : 100%|██████████████████████████████| 2320/2320 [13:54<00:00,  2.78it/s]
Epoch - 2
Accuracy: 0.7294027093669464
Loss: 0.697648284565015
Running Evaluation: 100%|██████████████████████| 17841/17841 [03:04<00:00, 96.69it/s]
Running Epoch : 100%|██████████████████████████████| 2320/2320 [13:54<00:00,  2.78it/s]
Epoch - 3
Accuracy: 0.8125635006304445
Loss: 0.4253964205786329
Running Evaluation: 100%|██████████████████████| 17841/17841 [03:04<00:00, 96.63it/s]
WER (after adding doc stride)-  [5.218709713581077, 4.592455579844179, 4.931337929488257, 5.239112157390281]
```

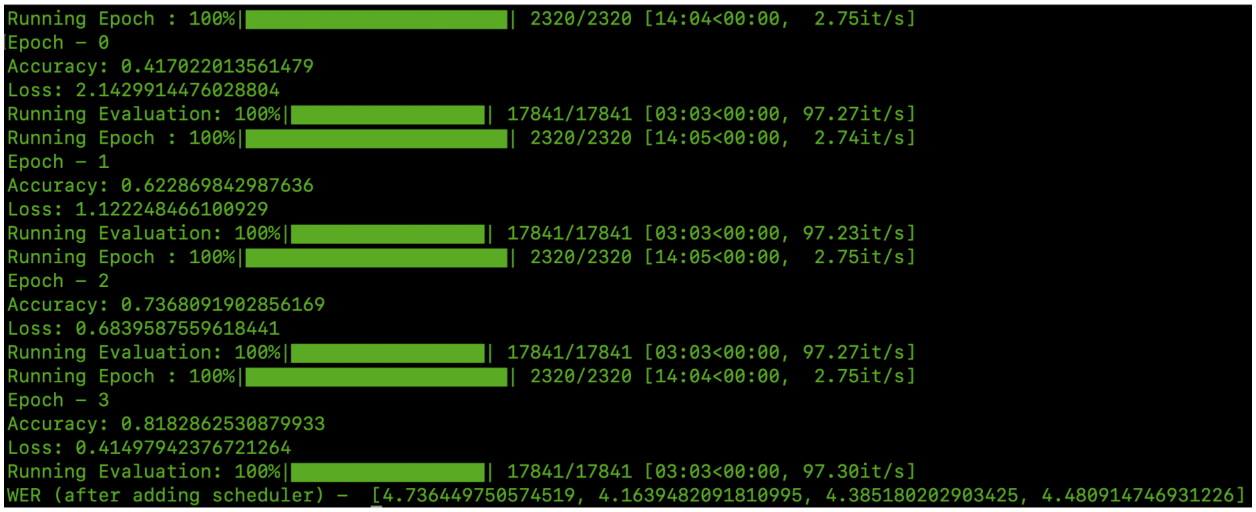
.

We can clearly see that the word error rate is lower than the baseline model. Now let's try adjusting the learning rate.

6. Adjusting the learning rate using a scheduler

I used a scheduler from huggingface which automatically adjusted the learning rate to its optimal value to give us performance improvements. First, lets understand how it works.

Adjusting the learning rate decay helps the model converge more effectively during training. Learning rate decay refers to the gradual reduction of the learning rate over the course of training epochs. This adjustment is crucial because it allows the model to fine-tune its parameters more efficiently. One key advantage of learning rate decay is its ability to prevent the model from overshooting or oscillating around the minimum of the loss function. By initially setting a relatively high learning rate and progressively decreasing it as training progresses, the model can make larger updates to its parameters in the early stages of training, helping it escape from local minima and explore the solution space more extensively. As training progresses and the model approaches convergence, reducing the learning rate prevents the model from making overly large updates that may disrupt the fine-tuning process, allowing it to converge more smoothly towards the optimal parameters.



```
Running Epoch : 100%|██████████████████████| 2320/2320 [14:04<00:00,  2.75it/s]
Epoch - 0
Accuracy: 0.417022013561479
Loss: 2.1429914476028804
Running Evaluation: 100%|█████████████████| 17841/17841 [03:03<00:00, 97.27it/s]
Running Epoch : 100%|██████████████████████| 2320/2320 [14:05<00:00,  2.74it/s]
Epoch - 1
Accuracy: 0.622869842987636
Loss: 1.122248466100929
Running Evaluation: 100%|█████████████████| 17841/17841 [03:03<00:00, 97.23it/s]
Running Epoch : 100%|██████████████████████| 2320/2320 [14:05<00:00,  2.75it/s]
Epoch - 2
Accuracy: 0.7368091902856169
Loss: 0.6839587559618441
Running Evaluation: 100%|█████████████████| 17841/17841 [03:03<00:00, 97.27it/s]
Running Epoch : 100%|██████████████████████| 2320/2320 [14:04<00:00,  2.75it/s]
Epoch - 3
Accuracy: 0.8182862530879933
Loss: 0.41497942376721264
Running Evaluation: 100%|█████████████████| 17841/17841 [03:03<00:00, 97.30it/s]
WER (after adding scheduler) -  [4.736449750574519, 4.1639482091810995, 4.385180202903425, 4.480914746931226]
```

Here we can see that it gives us the biggest performance improvement till now. This is the most optimal BERT model that I could tune-up for this assignment and it is well within the baseline. This just shows how important it is to tune the learning rate optimally even after having an optimal architecture for the task, to get the full performance out of a deep learning architecture.