CPSC- 8430

Homework 3 Report- Deep Learning

Extractive Question Answering

**Pragathi Pendem**

**GitHub Link:** <https://github.com/Pragathibruno/CPSC-8430_Deep-Learning_Bert-HW3_PragathiPendem.git>

**Introduction:**

In the area of Natural Language Processing and Information Retrieval known as "extractive question answering," it is necessary to offer a model context for it to forecast where an answer will be found in each passage. With programs like qa.py and tf squad.py, the SQuAD dataset of task-based questions and answers can be utilized to fine-tune a model.

A newer version of SQuAD 2.0 that focuses on speech-based questions and answers on diverse themes is called Spoken Squad. The Spoken Squad dataset intends to enhance the creation of spoken language processing question-answering models. Dealing with speech recognition failures and other sorts of noise in spoken language data, however, is a significant barrier to using this dataset. Using cutting-edge language models like BERT is one method to solve this obstacle (Bidirectional Encoder Representations from Transformers).

**Description of BERT Model:** In this Assignment we have used BERT model to perform the given task and below is the description for the model.

A pre-trained deep learning model called BERT in a full form we could say that Bidirectional Encoder Representations from Transformers for natural language processing, which is also a short form for NLP, was developed by Google (natural language processing). BERT is a very powerful and popular model that has proven to perform exceptionally well in across multiple of NLP applications, such as text classification, question answering, and language translation.

The BERT model architecture consists of several transformer encoders that may function in both directions, enabling them to take the entire input text's context into account. A method called Masked Language Modeling (MLM), which randomly masks some terms within the given input text and trains the model to anticipate the missing words taking as a starting point surrounding context, is employed to pre-train BERT.

Graphical user interface, diagram, application

Description automatically generated

A stack of transformer encoders makes up the BERT architecture, which can process incoming text in a bidirectional manner while taking the text's overall context into consideration. The Masked Language Modeling and Next Sentence Prediction tasks are used to pre-train BERT (NSP). Certain words in the input text are randomly masked by MLM, and the model must predict them based on the surrounding context. NSP makes predictions about the relationship between two input sentences. By incorporating a task-specific output layer, the pre-trained BERT model can be improved to perform at the cutting edge on a range of NLP tasks.

Hugging Face, a popular NLP library, provides pre-trained models for a range of NLP applications, including question-answering. A neural network that has been trained on a sizable amount of data to carry out a particular task is known as a pre-trained model. A variety of pre-trained BERT-based models with a focus on question-answering tasks are available in the Hugging Face library. Examples include "bert-base-uncased" and "bert-large-uncased-whole-word-masking-finetuned-squad". These models have been fine-tuned using massive datasets like SQuAD, which has enabled them to execute question-answering jobs with a high level of accuracy.

**Description of the Data Set:** In this assignment we have used Squad data set and below is the description if the dataset.

The spoken documents in the SpokenSQuAD dataset have text-based questions and answers to go along with them. The original SQuAD articles were transformed into spoken form using Google's text-to-speech technology, and then CMU Sphinx was used to turn the spoken form back into text to build the dataset. The test questions were still in written form. The SQuAD training set was used to create the SpokenSQuAD training set, while the SQuAD development set was used to create the testing set. A question was taken out of the dataset if there was no response to it in the article's related ASR transcription.

The F1 score is typically reported in evaluation datasets like the Stanford Question Answering Dataset (SQuAD). The finest outcomes to yet have been attained by BERT, which excelled on SQuAD. On SQuAD v1.1 and SQuAD v2.0, BERT received F1 scores of 93.2% and 92.7%, respectively.

**Results:**

Word error rate (WER) is a measurement that is widely used in speech recognition to determine how many errors there are between the predicted transcript and the reference transcript. As a result, WER is typically not used to evaluate question answering systems.

Text

Description automatically generated

**Implementation of Linear scheduler model:**

The learning rate is a hyperparameter in deep learning that controls how much the model should change the weights during training. a slow rate of learning might lead the model to gradually converge or become stuck in a local minimum, while a high the model may overshoot due to learning rate. the optimal weights and squander convergence.

A learning rate schedule that declines linearly with each epoch is known as a linear scheduler. This implies that the learning rate gradually decreases over time, which may aid in a faster convergence of the model.

The beginning learning rate, the number of epochs, and the desired final learning rate must all be specified to create a linear scheduler. Following that, you can get the learning rate for each epoch as follows:

lr = initial\_lr \* (1 - epoch / num\_epochs) + final\_lr \* (epoch / num\_epochs)

Based on the current epoch and the total number of epochs, this equation determines a linear interpolation between the starting learning rate and the end learning rate. The LambdaLR class in PyTorch enables you to create a linear scheduler and define a special function to change the learning rate at each epoch. Here's an illustration:

from torch.optim.lr\_scheduler import LambdaLR

initial\_lr = 0.001

final\_lr = 0.0001

num\_epochs = 10

optimizer = torch.optim.Adam(model.parameters(), lr=initial\_lr)

scheduler = LambdaLR(optimizer, lr\_lambda=lambda epoch: (1 - epoch / num\_epochs))

In this illustration, the Adam optimizer is being used, and its starting learning rate is set to 0.001. Also, we are developing a LambdaLR scheduler that interpolates linearly across 10 epochs between the initial learning rate and a final learning rate of 0.0001. A lambda function that receives the current epoch as input and outputs the matching learning rate is used as the lr lambda argument.

You can improve the accuracy of your model tuning and maybe improve job performance by utilizing a linear scheduler.

**Outcome of Linear Scheduler:**

Text

Description automatically generated with medium confidence

**Pre-Processing and Doc Stride:**

Doc stride is a parameter that controls how much overlap there should be between input segments during both training and inference in NLP models like BERT. Longer input texts that are longer than the model's maximum input length are handled using this overlap**.**

**Timeline

Description automatically generated with low confidence**

Timeline

Description automatically generated with low confidence