Sentimental Analysis for App Reviews Using BERT Model

1. Introduction:

In this report, it gives an idea about the Transformers and Bert transformer model. Transformers were used to solve the various NLP tasks. Most of the organizations around the world used transformers in developing models for NLP tasks which includes text translation from one language to another, chatbots help to answer the questions and many more. And the Transformer library has a key object known as pipeline (), which is the bridge between preprocessing and postprocessing text. Main purpose to create a pipeline is to provide single API through which any transformer models can be loaded, train the model, and save. There are many pipelines available, and each pipeline will select pretrained model to sense the text. Moving forward, you will understand the working of the transformer and how to use them.

In this assignment I'm using BERT (Bidirectional Encoder Representations from Transformers) transformer model, which is trained on the vast amount of the text data like Google's book corpus, Wikipedia text, and more different text, using MLM (mask Language Model) and NSP (Next sentence prediction). The main feature of the Bert is a bidirectional, which considers the text from left to right that helps to encode the words from the sentence. There are three main components for the BERT model that are encoder, decoder, and attention layer, where each component has the different functionality. Using the pipelines the trained model is used to predict the masked word in sentence, gives the summarization of the text.

2. Dataset

For this assignment, I have chosen the dataset called app-reviews which is loaded from the dataset library. This dataset contains app reviews and ratings from the customer, which classified the reviews into negative [0] and positive [1] , where 0,1 are the binary values.

3. Methodology:

3.1 Architecture of BERT

The transformer which has the several layers of self-attention mechanisms, based on the Bert's architecture. Below figure 1 shows the size and architecture as it contains a stack of transformer encoder layer, which is one of the components of the model. There are numerous heads in each layer, which are fully connected neural networks boosted with the mechanism. And the weight representation is created by each head computing the key, value, and query vector for each input in a sequence.

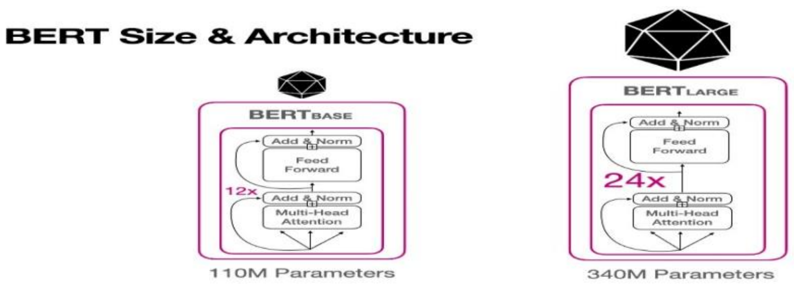
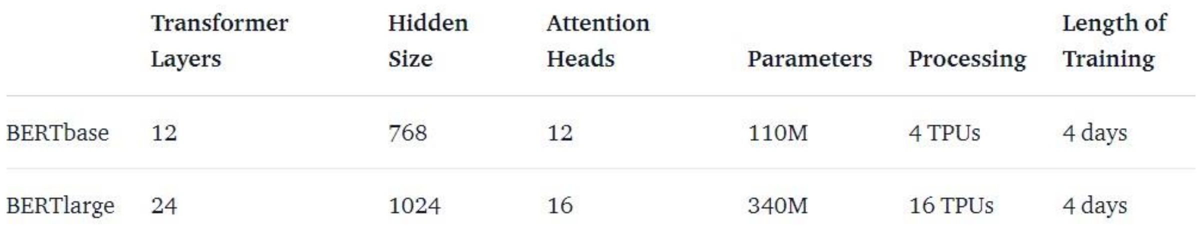


Figure 1: Bert architecture and size image

In the below figure two describes the Bert architecture and layers which has the two parameter blocks, one with 12 transformer blocks have 110 million parameters, 768 hidden size and 12 attention heads which is Bert Base and other one is 24 transformer blocks with 340 million Parameters, 1024 hidden size and 16 attention head known as Bert large.



	Transformer Layers	Hidden Size	Attention Heads	Parameters	Processing	Length of Training
BERTbase	12	768	12	110M	4 TPUs	4 days
BERTlarge	24	1024	16	340M	16 TPUs	4 days

Figure 2: Bert Architecture glossary

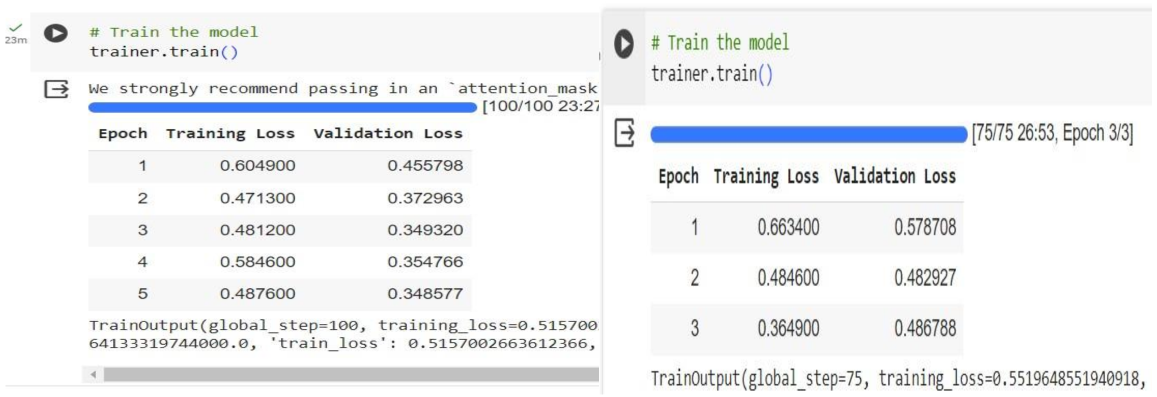
3.2 Bert Pre-trained and Fine-tuning framework

Bert has a two-step framework that is pre-training and fine-tuning. During the pretraining, as mentioned in the introduction Bert is pretrained on the vast corpus data, which helps to predict the masked word in a sentence. This not only masks the word, but also teaches how to create a contextualized word embedding, also captures the human language understanding. Once the pretrained is completed, using labelled data, the pretrained Bert model can be tuned for certain downstream tasks. Updating the parameters in the model can be the key factor to reach the targeted tasks and decent accuracy. The important thing is that, while fine tuning this approach usually requires a small amount of labelled dataset. Once the model is trained, through the reverse propagation, the parameters are adjusted to minimize the loss function. Once its done, this can be tested on a new data text for testing to check the trained model’s performance and its accuracy.

4.<u>Results (Training and Fine-tuning model)</u>

In this code, I have used a pretrained BertForSequenceClassification and BertTokenizer. During the training metrics both training loss and validation loss are being monitored. The training loss tells you how well the model is fitted into training data and the validation loss tells you, how well the model is generalized to new data or the untested data. As the image shown below (Figure 3), the training loss decreases initially from epoch 1 to epoch 3, which shows its learning from the

data, but during epoch 4 there is slight increase in training loss, which may be overfitting during the training process and then decreases.
The validation loss seems to decrease consistently, indicating that the model is improving its performance on unused or untested data.



```
# Train the model
trainer.train()
```

```
We strongly recommend passing in an `attention_mask
                                          [100/100 23:27
Epoch  Training Loss  Validation Loss
  1       0.604900        0.455798
  2       0.471300        0.372963
  3       0.481200        0.349320
  4       0.584600        0.354766
  5       0.487600        0.348577
TrainOutput(global_step=100, training_loss=0.515700
64133319744000.0, 'train_loss': 0.5157002663612366,
```

```
# Train the model
trainer.train()
```

```
                                   [75/75 26:53, Epoch 3/3]
Epoch  Training Loss  Validation Loss
  1       0.663400        0.578708
  2       0.484600        0.482927
  3       0.364900        0.486788
TrainOutput(global_step=75, training_loss=0.5519648551940918,
```

Figure 3: Model performance metrics

5. **Conclusion:**
Over the period of epochs, the training loss and validation loss decreases, indicating that the model is consistently learning and adjusting or transferring to the new data. With the model effective learning from the data and achieving low validation loss, the application of the BERT model to the "App review dataset" delivers positive outcomes.

Google Colab Notebook link:

https://colab.research.google.com/drive/10Z7gyrcO7xjh84ZNJChXnpGwk5YT9p9H?usp=sharing

References:

1. **Muller, (2022), BERT 101 State Of The Art NLP Model Explained. Available at : https://huggingface.co/blog/bert-101.**
2. **Follow,(2018), BERT Explained: State of the art language model for NLP, https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270.**
3. **huggingface.co, (n.d.), Fine-tuning a model with Keras, Hugging Face. https://huggingface.co/learn/nlp-course/chapter3/3?fw=tf.**