Sentiment Analysis

Introduction:

The Rotten Tomatoes movie review dataset is a corpus of movie reviews used for sentiment analysis, originally collected by Pang and Lee [1]. In their work on sentiment treebanks, Socher et al. [2] used Amazon's Mechanical Turk to create fine-grained labels for all parsed phrases in the corpus. This competition presents a chance to benchmark your sentiment-analysis ideas on the Rotten Tomatoes dataset. You are asked to label phrases on a scale of five values: negative, somewhat negative, neutral, somewhat positive, positive. Obstacles like sentence negation, sarcasm, terseness, language ambiguity, and many others make this task very challenging. Our goal is to build a sentiment classification model utilising the current SOTA method in NLP i.e. Transformers. We built the BERT model.

Data :

We received the data from Kaggle. Following is the data description we found on kaggle's website:

The dataset is comprised of tab-separated files with phrases from the Rotten Tomatoes dataset. The train/test split has been preserved for the purposes of benchmarking, but the sentences have been shuffled from their original order. Each Sentence has been parsed into many phrases by the Stanford parser. Each phrase has a PhraseId. Each sentence has a SentenceId. Phrases that are repeated (such as short/common words) are only included once in the data.

The sentiment labels are:

0 - negative
1 - somewhat negative
2 - neutral
3 - somewhat positive
4 - positive

Steps followed:

Pre-processing Data:

We started by reading the data into a Pandas Dataframe using th read_csv function. Because we were working with .tsv (tab seperate values) files we need to specify that we will be taking tab characters as the delimiters. The Phrase column contains all of our text data that we processed. We saw that there were many copies through segments of the same answer. We could reduce the amount of noise in our dataset by removing those duplicates.

We then tokenized this text to create two input tensors; our input IDs, and attention mask. We saved our tensors within two numpy arrays, which were of dimensions len(df) * 512 - the 512 is the sequence length of our tokenized sequences for BERT, and len(df) the number of samples in our dataset. Now we began tokenizing with a BertTokenizer from hugging face library. Post tokenisation we saved data as Numpy binary files.

Input Pipeline:

As we were using TensorFlow we used the tf.data.Dataset, and we created TF dataset object using from_tensor_slices. Each sample in our dataset was a tuple containing a single Xids, Xmask, and labels tensor. However, when feeding data into our model we need a two-item tuple in the format (\<inputs>, \<outputs>). Now, we had two tensors for our inputs - so, what we do is enter our \<inputs> tensor as a dictionary:

```
{
   'input_ids': <input_id_tensor>,
   'attention_mask': <mask_tensor>
}
```

Next, we need to shuffle our data, and batch it. We took batch sizes of 16 and drop any samples that don't fit evenly into chunks of 16. The final step was to split our data into training and validation sets. For this we used the take and skip methods, creating and 90-10 split. we saved both to file using tf.data.experimental.save.

Build and Train:

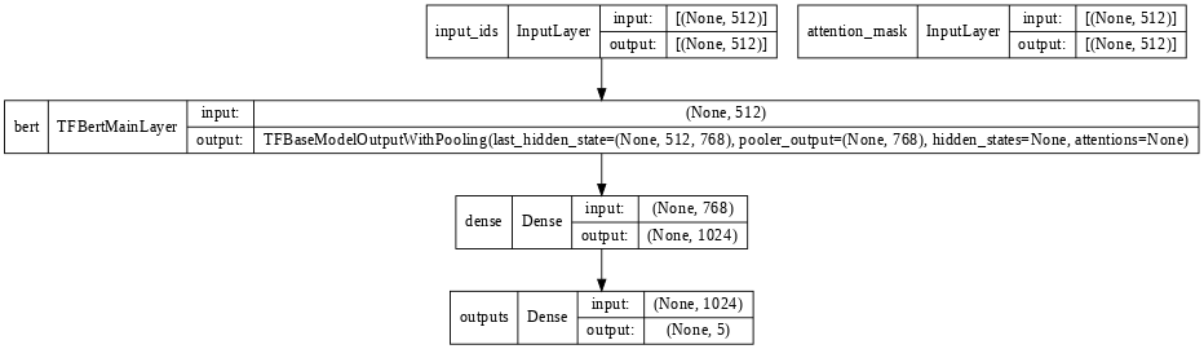
Now we were ready to begin building our sentiment classifier and begin training. We will be building out what is essentially a frame around Bert, that will allow us to perform language classification. First, we can initialize the Bert model, which we will load as a pretrained model from transformers.

Now we need to define the frame around Bert, we need:

- Two input layers (one for input IDs and one for attention mask).
- A post-bert dropout layer to reduce the likelihood of overfitting and improve generalization.
- Max pooling layer to convert the 3D tensors output by Bert to 2D.
- Final output activations using softmax for outputting categorical probabilities.

We can now define our model, specifying input and output layers. Finally, we can freeze the Bert layer because Bert is already highly trained, and contains a huge number of parameters so will take a very long time to train further. Nonetheless, if you'd like to train Bert too, there is nothing wrong with doing so. Our model architecture is now setup, and we can initialize our training parameters.

Following is how our model looks like:



Load and Predict:

We've now built our model, trained it, and saved it to file - now we can begin applying it to making predictions. First, we load the model with tf.keras.models.load_model.

Before making our predictions we need to format our data, which requires two steps:

- Tokenizing the data using the bert-base-cased tokenizer.
- Transforming the data into a dictionary containing 'input_ids' and 'attention_mask' tensors.

Following is final dataframe.

	PhraseId	SentenceId	Phrase	Sentiment
0	156061	8545	An intermittently pleasing but mostly routine ...	3
15	156076	8546	Kidman is really the only thing that 's worth ...	3
93	156154	8547	Once you get into its rhythm ... the movie bec...	3
117	156178	8548	I kept wishing I was watching a documentary ab...	3
158	156219	8549	Kinnear does n't aim for our sympathy , but ra...	3