Student Name: Stefanos Vavoulas

Artificial Intelligence II
Assignment 4

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Develop a sentiment classifier by fine-tuning the pretrained BERT-base model

- Preprocessing, Data Cleansing, Tokenization - Mapping

Preprocessing of the data remained the same as in previous Assignments (lowercase words, excluding all newline characters, urls, mentions, emojis, punctuation, as well as a list of english stop words, meaning words that would be of no use to my model).

Tokenization: I loaded the bert-base-uncased model for the tokenizer (we use the same one for the model), and then I used tokenizer.encode_plus to encode the reviewes into mapped ids for each word. After trying multiple values for max_length, I ended up choosing 128, which gave our model peak performance, while keeping the training time as small as possible.
I also truncated and padded all reviews to that same length and returned the attention mask for each mapped review (an attention mask is essentially a list of 0s for pad tokens and 1s for actual word ids that helps in training time, since the model ignores padding tokens).
Finished by converting the reviews - now mapped word ids -, attention masks and sentiment labels to tensors.

- Preparation for Training

I created a dataset from the above mentioned tensors (word ids, attention masks, sentiment labels), and then split it 85-15 to a training and validation dataset respectively.
After that, I loaded both datasets into dataloaders, made the model run training in the GPU, chose number of epochs, optimizer and learning rate (chosen values explained later).
I also attempted to implement a scheduler step, modifying the learning rate for each epoch, but it didn't seem to yield any noticeable results, especially since I chose to train my model for only 2 epochs, as I'll mention later, so I ended up removing it.

- Training and Evaluation

After training the model with different numbers of epochs, learning rates, batch sizes and max sequence lengths in tokenization, I ended up choosing:
2 epochs, since my model seemed to overfit the training set for anything over that (explained with example later)
learning rate = 3e-5 or 0,00003, as it seemed to balance the best speed of convergence and accuracy of the model
batch size = 32, as it seemed to balance training time and generalization performance
max_length = 128, as already explained

For the last example I ran, the model seemed to perform very well in the validation set (F1 score

of around 0,903) for the above mentioned hyperparameters
We can also observe that up to 2 epochs it doesn't overfit the training set:
Epoch 1: Training loss = 0.32, Validation Loss = 0.26
Epoch 2: Training loss = 0.18, Validation loss 0.25

When running examples with 3 or more epochs, I found that, while training loss continued to decrease significantly (epoch 3: training loss = 0.10), validation loss started increasing (epoch 3: validation loss = 0.32), meaning our model was now overfitting the training set, and would become less efficient in generalization performance. The observation was confirmed, since the f1 score for the third epoch on the validation set was lower (0.895), compared to that of the second epoch.

- Final Comment

My observation was that, if needed, training could be greatly optimized time-wise by setting max sequence length = 64, learning rate = 5e-5, epochs = 1 and the model still produced great results in the validation set (f1 score of around 0.85) in less than 8 minutes on the Tesla T4 GPU. However, I think I achieved the best balance of accuracy and speed with the hyperparameters I chose, reaching an accuracy of 0.9 in less than 30 minutes.