Assignment 1 - Sentiment Analysis over IMDB Dataset

In this task we are trying to perform sentiment analysis over the IMDB dataset, we are working with the IMDB dataset which contains 50,000 movie reviews written by users. The objective is to predict the sentiment whether positive or negative for a review. To solve this task, I have utilized a pre-trained Bert [1] model using the hugging face library[2].

Dataloader

I have built a custom data loader to Prepare the data into the correct format that is required for the Bert Model, In that data loader I am reading the rows from the CSV which contains in the first column the text review, and in the second column, the label. For the text review, I am tokenizing it using a pre-trained tokenizer from the hugging Face library. The tokenizer plays a very important role as it converts the input text sentence into an array of single words/important characters. It then converts these words into IDs that are understandable by the model. For Bert models, a WordPiece tokenizer is used, it also adds the [CLS] and [SEP] token, from the tokenizer we also get the attention mask which tells the model which part of the tokenized array is relevant and which is just padding. Finally, we append the Labels to tokenized output and that is our model input. I have added padding of max_length to the tokenizer and a maximum length of 512 and the data loaders have a batch size of 8

The Model

I have used the pre-trained Bert model for this task. Specifically, the Bert-base-cased, that is a large language model trained on the task of masked word prediction and next sentence prediction. In my approach, I am fine-tuning based on sentiment analysis as a downstream task.

since the model outputs the encoded representation of the input sentence we will add a dropout since that allows us to prevent overfitting and a feed-forward linear layer to get the output shape of Bert model to the shape we desire. The Bert Model provides two outputs, the sequenced output which is the encoded representation for every token in the sentence, and the pooled output which is the encoded representation of the entire sentence. I have used the pooled output in my implementation.

The linear layer is of the size of the output hidden dimension as mentioned in the config (768) and the dropout probability is the same as the hidden layers dropout in the model from the config(0.1).To compute the loss I am using the CrossEntropyLoss[3] between the final logits and labels from PyTorch.

Optimizers & Training

I have used the AdamW[4] optimizer with the learning rate of 5e-5 and a linear scheduler that linearly reduces the LR until it reaches 0. I have used AdamW as its a very effective optimizer, it maintains per parameter learning rates and changes them accordingly. AdamW improves upon the Adam optimizer and changes the weight decay method used. The Model takes 2 hours to complete one epoch and within the limits of google Colab of 6hours, I am able to train for up to

1 epoch safely without losing GPU access. I tried multiple times to train for multiple epochs but either due to a disconnection or free GPU limit reached I couldn't complete the training.
While training I am calculating the accuracy as well I have observed that the model starts with an accuracy of 50% as its a binary task and then slowly increases to 96%,

Observations and Hyperparameters

In my training, I observed that the max_length of the tokenized string affects the accuracy, I trained the model with lengths of 256 and 512 with 512 I observed more accuracy but in return, it takes longer to train, and the batch size is limited to 8. With the 256 length, we can keep the batch size 16 which makes the training faster. I decided to Train with a tokenization length of 512. Other hyperparameters such as the learning rate and schedulers, I decided the values based on online learning resources and hugging face tutorials[5]. I also observed a peculiarity that a model trained with token length 256 can be used with inputs of token length 512 during evaluation and it lends it greater accuracy and F1 score, so we can devise a method to train quickly on 256 length tokens and then inference on 512 length.

All hyperparameters used in the model and training

Token max length= 512
Batch Size = 8
Epochs = 1
Optimizer = AdamW
Scheduler = Linear
Learning Rate = 5e-5
Number of Linear Layers = 1
Output Linear layer hidden size = 768
Dropout probability= 0.1
Loss = CrossEntropyLoss

Results

After training my model I have received the following results

DATA SPLIT	F1	ACCURACY
TEST	93.71	93.7
DEV	93.38	93.4
TRAIN	96.9	96.9

Further Discussion

In My current approach, I have used the Bert Model for this task, However, other models may be used for this task as well such as DistillBert[6] which is a lighter more efficient model, or RoBerta[7] even larger and powerful model for this task. Furthermore, techniques can be used to add one more stage of fine-tuning known as pre-finetuning can be added after pre-training, this kind of finetuning will be done on multiple tasks at once, this method is described in the MUPPET [8] paper. That model will then yield even better results. Also, techniques such as prompt learning [9] can also be used to solve this task where we are not fine-tuning the model rather generating prompts in such a way that the model will generate the next word in the prompt which will be the response. For example, we can use a prompt like "I didn't like the movie at all and the acting was really bad, the movie was _____" where the model will predict the empty word and then that word can be used to assess whether the sentiment was positive or negative. This method also reduces the training data requirement of finetuning and provides good results.

References

1. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.
2. Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. CoRR, abs/1910.03771, 2019.
3. Z. Zhang and M. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In Advances in Neural Information Processing Systems, pages 8792–8802, 2018
4. Loshchilov, I. and Hutter, F. (2017). Fixing weight decay regularization in adam. arXiv preprint arXiv:1711.05101.
5. https://huggingface.co/transformers/training.html
6. Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. CoRR, abs/1910.01108.
7. Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. CoRR, abs/1907.11692, 2019
8. Armen Aghajanyan, Anchit Gupta, Akshat Shrivastava, Xilun Chen, Luke Zettlemoyer, and Sonal Gupta. 2021. Muppet: Massive multi-task representations with pre-finetuning.
9. Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. arXiv preprint arXiv:2104.08691.