NLP Assignment 1
Sentence Classification on the IMDB Dataset

Submission by: Apoorv Garg (PID: apoorvgarg)

I have used two different pre-trained language models from HuggingFace to train on this dataset.

- **bert-base-cased pretrained for the downstream task of sentence classification**. The name for this model in HuggingFace repo is: **BertForSequenceClassification**

- **distillbert-base-cased**. To this model, I have added 2 linear layers separated by a dropout layer and a RelU non-linearity.

I reveived the following metrics on the **test dataset:**

BertForSequenceClassification :

Precision	0.9159170903402425
Recall	0.9349301397205588
F1 score	0.9253259581193203
Accuracy	0.9244

Sentence Classifier: distillbert-base-cased + 2 Linear Layers + Dropout + ReLU:

Precision	0.9021823850350741
Recall	0.9241516966067864
F1 score	0.9130349043581149
Accuracy	0.9118

Since both models achieve the required metric of 0.90, I am **submitting the distilled bert model-based custom Sentence Classifier as my submission**. But, I am also attaching the notebook for the BertForSequenceClassification in the **extras** folder, for visualization purposes and because, it has helped me in making many decisions for my final submission.(But, it is not my final submission)

Performance metrics of Sentence Classifier on the Validation Dataset :

Precision	0.9033646322378717
Recall	0.9184566428003182
F1 score	0.9108481262327416
Accuracy	0.9096

Performace metrics of Sentence Classifier on the entire Training Dataset :

Precision	0.9993494144730257
Recall	0.9993994294579851
F1 score	0.9993744213397393
Accuracy	0.999375

Note: I have chosen distilled bert because of the ease of training(provided the constraint that the runtime gets disconnected after ~6+ hours of training and I wanted the loss to start converging. For distilled bert and linear layer combination(Sentence classifier), I saw that the loss was converging after the 5th epoch to the order of 1e-4. I could only train ~2-3 epochs for bert base model, but for distilled bert, I could train and infer on more than 5 epochs, and was sure on the loss convergence. Distilled bert tokenizer does not have token_type_ids encoding for the tokenizer.)

The entire process can be divided into:

1. Dataloader design:

   I read the csv file as pandas data frames and loaded them in the PyTorch dataloader class as custom datasets. This also involves the design of tokenizer.

   Tokeniser Design: I chose the cased version of the tokenizer to incorporate extra information in reviews like "BAD" instead of "bad".

   - I chose a **max-length of 256** because the average review length was 231.33 words on the training set.
   - I chose a **batch size of 16** (32 was giving an out-of-memory error.)
   - To incorporate linear layer on CLS token, use_special_tokens=True.

2. Classifier design:

   - I have tried both, bert-base and Distillbert as the language model for tokenizing and finetuning purposes. For two epochs, the F1 score on bert base cased was 0.92, while the F1 score was 0.9 on the Distilled Bert model. However, the error was still converging for both models. It was found that Distilled Bert with two Linear layers took approximately 30 minutes for one epoch of training and validation on 1600 samples, while BertForSequenceClassification took 150 minutes to train. This is a 4.44% upside in performance for a 400% increase in training time. This is the reason Distilled Bert has been chosen as the Language Model for fine-tuning.
   - The **CLS** token of the pretrained Language Model is the first value in the encodings, and has a size of 16 X 768 for each batch (i.e. a size of 768 for each review if squeezed). This token contains information on the entire sentence and can be used to train the classifier. Hence, a linear layer on CLS token with an output size od 2 should be good to train the downstream task and fine tune the language model. But, I have chosen 1 extra **hidden linear layer with a size of 20**, **a dropout of 0.2**, and a **ReLU non-linearity**(This is done so that I have more tuneable parameters for the linear layer. The parameters like size of hidden layer, type of non-linearity to choose were estimated from the following paper). [Parameter Efficient Transfer Learning for NLP]. I also confirmed this Dropout value of 0.2 as this is mentioned in the HuggingFace Distilled Bert

config parameters for the downstream LM-based sequence classification task.

Sentence Classifier:

```
CLS (768)-> Linear Layer (20) || Dropout(0.2) -> ReLU -> Linear Layer(2)
```

3. Optimizer, Learning Rate Scheduler And Loss Function Design:

- **Adamw** is used as the optimizer because of its faster convergence and automatic learning rate updation properties. This optimizer is also recommended in the previous paper and in the guide by Huggingface on sequence classification. The upper limit for learning rate in Adamw is configured to decay linearly from a maximum value of **5e-5**, and **number of warmup samples is set to 0**. I have used huggingface article link on how to train a sequence classifier.[Training a sequence classifier]
- Since this is a classification problem, standard PyTorch **CrossEntropyLoss** is used as the loss function.

4. Training and Validation Loop:

- It made sense to freeze the languange model and just train the hidden layer parameters for the fine tuning task, which is a standard machine learning policy. But it is mentioned in the HuggingFace sequence classification guide that entire model fine-tuning gives better results.

> Quoted text: “Note that if you are used to freezing the body of your pretrained model (like in computer vision) the above may seem a bit strange, as we are directly fine-tuning the whole model without taking any precaution. It actually works better this way for Transformers model (so this is not an oversight on our side. If you’re not familiar with what “freezing the body” of the model means, forget you read this paragraph."

- For training the following steps are used:

> 1. Read one batch of tokenized reviews and labels from the train data loader.
> 2. Convert these values to the GPU.
> 3. Zero the accumulated gradients (PyTorch limitation).
> 4. Run model to get the predictions.

5. Calculate the Loss.
6. Compute back-propagation on the loss.
7. Update the parameters.
8. Update the learning rate.
9. Repeat this process for each batch in the training dataloader (Covers the entire training space.)
10. This is 1 epoch of training. Calculate the F1 score and accuracy metrics on the validation set. Here, I use only **1600 samples** fron the validation set to save time.
11. Repeat this process for **5 epochs**.
12. Save the model after this process concludes.

It was noted that the loss for each batch of training samples falls to the order of 1e-4 for the fifth epoch.

5. Performace of the final model on Test, Validation and Training Sets:

- I have used sklearn to evaluate the precision, recall, and F1 scores. The saved model is loaded and run on the entire test set to calculate the F1 score. The resulting predictions are calculated and appended to a List. This list is finally converted to CPU and then sklearn API is called on the prediction and labels list to calculate Precision, Recall, F1 score, and Accuracy.

6. Code Explanation and Runtime Results:

A pdf version of the notebook is also attached in this folder. This pdf contains annotated text boxes on each subsection of code and also contains the results of some important code blocks.

7. Model Parmeters:

Batch Size	16
Max-length of tokenizer	256
Tokenizer and language model	Distilled bert
Tokenizer special labels	True

Linear Layer 1O output size	20
Dropout	0.2
Non-Linearity	ReLU
Linear Layer 2 Size	2
Optimizer	Adamw
LR Scheduler decay	Linear
LR Schduler ceiling	5e-5
LR warmup iterations	0
Loss Function	CrossEntropyLoss
Number of Training Epochs	5
Number of samples evaluated in validation iteration	1600

8. Appendix:

The batch loss for the entire training set for each epoch:

```
tensor(0.3428, device='cuda:0', grad_fn=<NllLossBackward>)
198
tensor(0.1465, device='cuda:0', grad_fn=<NllLossBackward>)
297
tensor(0.4405, device='cuda:0', grad_fn=<NllLossBackward>)
396
tensor(0.2062, device='cuda:0', grad_fn=<NllLossBackward>)
495
tensor(0.2421, device='cuda:0', grad_fn=<NllLossBackward>)
594
tensor(0.5980, device='cuda:0', grad_fn=<NllLossBackward>)
693
tensor(0.2064, device='cuda:0', grad_fn=<NllLossBackward>)
792
tensor(0.3631, device='cuda:0', grad_fn=<NllLossBackward>)
891
tensor(0.8403, device='cuda:0', grad_fn=<NllLossBackward>)
990
tensor(0.1605, device='cuda:0', grad_fn=<NllLossBackward>)
1089
tensor(0.1320, device='cuda:0', grad_fn=<NllLossBackward>)
1188
tensor(0.3393, device='cuda:0', grad_fn=<NllLossBackward>)
1287
tensor(0.3542, device='cuda:0', grad_fn=<NllLossBackward>)
1386
tensor(0.1650, device='cuda:0', grad_fn=<NllLossBackward>)
1485
tensor(0.2359, device='cuda:0', grad_fn=<NllLossBackward>)
1584
tensor(0.1936, device='cuda:0', grad_fn=<NllLossBackward>)
1683
tensor(0.3975, device='cuda:0', grad_fn=<NllLossBackward>)
```

```
1782
tensor(0.2257, device='cuda:0', grad_fn=<NllLossBackward>)
1881
tensor(0.1475, device='cuda:0', grad_fn=<NllLossBackward>)
1980
tensor(0.1107, device='cuda:0', grad_fn=<NllLossBackward>)
2079
tensor(0.4836, device='cuda:0', grad_fn=<NllLossBackward>)
2178
tensor(0.2015, device='cuda:0', grad_fn=<NllLossBackward>)
2277
tensor(0.0573, device='cuda:0', grad_fn=<NllLossBackward>)
2376
tensor(0.1891, device='cuda:0', grad_fn=<NllLossBackward>)
2475
tensor(0.2204, device='cuda:0', grad_fn=<NllLossBackward>)
1 epoch terminated. Evaluating on validation....
tensor([0, 0, 0,  ..., 0, 1, 0]) tensor([0, 0, 0,  ..., 0, 1,
0])
Precision =  0.9252577319587629
recall =  0.8831488314883149
F1 score =  0.9037130270610447
accuracy =  0.904375
99
tensor(0.1594, device='cuda:0', grad_fn=<NllLossBackward>)
198
tensor(0.0165, device='cuda:0', grad_fn=<NllLossBackward>)
297
tensor(0.3258, device='cuda:0', grad_fn=<NllLossBackward>)
396
tensor(0.1322, device='cuda:0', grad_fn=<NllLossBackward>)
495
tensor(0.0497, device='cuda:0', grad_fn=<NllLossBackward>)
594
tensor(0.3444, device='cuda:0', grad_fn=<NllLossBackward>)
693
tensor(0.3102, device='cuda:0', grad_fn=<NllLossBackward>)
792
tensor(0.2628, device='cuda:0', grad_fn=<NllLossBackward>)
891
tensor(0.6845, device='cuda:0', grad_fn=<NllLossBackward>)
990
tensor(0.0537, device='cuda:0', grad_fn=<NllLossBackward>)
1089
tensor(0.0600, device='cuda:0', grad_fn=<NllLossBackward>)
1188
tensor(0.2935, device='cuda:0', grad_fn=<NllLossBackward>)
1287
tensor(0.5236, device='cuda:0', grad_fn=<NllLossBackward>)
1386
tensor(0.1482, device='cuda:0', grad_fn=<NllLossBackward>)
1485
tensor(0.0602, device='cuda:0', grad_fn=<NllLossBackward>)
1584
tensor(0.0946, device='cuda:0', grad_fn=<NllLossBackward>)
1683
tensor(0.1272, device='cuda:0', grad_fn=<NllLossBackward>)
1782
tensor(0.1492, device='cuda:0', grad_fn=<NllLossBackward>)
1881
tensor(0.0307, device='cuda:0', grad_fn=<NllLossBackward>)
1980
tensor(0.1319, device='cuda:0', grad_fn=<NllLossBackward>)
2079
tensor(0.1310, device='cuda:0', grad_fn=<NllLossBackward>)
2178
tensor(0.1703, device='cuda:0', grad_fn=<NllLossBackward>)
2277
tensor(0.0588, device='cuda:0', grad_fn=<NllLossBackward>)
2376
tensor(0.0421, device='cuda:0', grad_fn=<NllLossBackward>)
2475
tensor(0.1267, device='cuda:0', grad_fn=<NllLossBackward>)
1 epoch terminated. Evaluating on validation....
```

```
tensor([0, 0, 0,  ..., 0, 1, 0]) tensor([0, 0, 0,  ..., 0, 1,
0])
Precision =  0.9427430093209055
recall =  0.8708487084870848
F1 score =  0.9053708439897697
accuracy =  0.9075
99
tensor(0.0145, device='cuda:0', grad_fn=<NllLossBackward>)
198
tensor(0.0063, device='cuda:0', grad_fn=<NllLossBackward>)
297
tensor(0.0242, device='cuda:0', grad_fn=<NllLossBackward>)
396
tensor(0.0094, device='cuda:0', grad_fn=<NllLossBackward>)
495
tensor(0.0172, device='cuda:0', grad_fn=<NllLossBackward>)
594
tensor(0.0457, device='cuda:0', grad_fn=<NllLossBackward>)
693
tensor(0.1246, device='cuda:0', grad_fn=<NllLossBackward>)
792
tensor(0.0259, device='cuda:0', grad_fn=<NllLossBackward>)
891
tensor(0.0150, device='cuda:0', grad_fn=<NllLossBackward>)
990
tensor(0.0044, device='cuda:0', grad_fn=<NllLossBackward>)
1089
tensor(0.1734, device='cuda:0', grad_fn=<NllLossBackward>)
1188
tensor(0.0481, device='cuda:0', grad_fn=<NllLossBackward>)
1287
tensor(0.3872, device='cuda:0', grad_fn=<NllLossBackward>)
1386
tensor(0.0050, device='cuda:0', grad_fn=<NllLossBackward>)
1485
tensor(0.0079, device='cuda:0', grad_fn=<NllLossBackward>)
1584
tensor(0.0081, device='cuda:0', grad_fn=<NllLossBackward>)
1683
tensor(0.0048, device='cuda:0', grad_fn=<NllLossBackward>)
1782
tensor(0.0142, device='cuda:0', grad_fn=<NllLossBackward>)
1881
tensor(0.0025, device='cuda:0', grad_fn=<NllLossBackward>)
1980
tensor(0.0131, device='cuda:0', grad_fn=<NllLossBackward>)
2079
tensor(0.0172, device='cuda:0', grad_fn=<NllLossBackward>)
2178
tensor(0.0037, device='cuda:0', grad_fn=<NllLossBackward>)
2277
tensor(0.0124, device='cuda:0', grad_fn=<NllLossBackward>)
2376
tensor(0.0053, device='cuda:0', grad_fn=<NllLossBackward>)
2475
tensor(0.2817, device='cuda:0', grad_fn=<NllLossBackward>)
1 epoch terminated. Evaluating on validation....
tensor([0, 0, 0,  ..., 0, 1, 0]) tensor([0, 0, 0,  ..., 0, 1,
0])
Precision =  0.9388816644993498
recall =  0.8880688806888068
F1 score =  0.9127686472819216
accuracy =  0.91375
99
tensor(0.0157, device='cuda:0', grad_fn=<NllLossBackward>)
198
tensor(0.0044, device='cuda:0', grad_fn=<NllLossBackward>)
297
tensor(0.0023, device='cuda:0', grad_fn=<NllLossBackward>)
396
tensor(0.0029, device='cuda:0', grad_fn=<NllLossBackward>)
495
tensor(0.0075, device='cuda:0', grad_fn=<NllLossBackward>)
594
```

```
tensor(0.0018, device='cuda:0', grad_fn=<NllLossBackward>)
693
tensor(0.0013, device='cuda:0', grad_fn=<NllLossBackward>)
792
tensor(0.0212, device='cuda:0', grad_fn=<NllLossBackward>)
891
tensor(0.0100, device='cuda:0', grad_fn=<NllLossBackward>)
990
tensor(0.0015, device='cuda:0', grad_fn=<NllLossBackward>)
1089
tensor(0.0008, device='cuda:0', grad_fn=<NllLossBackward>)
1188
tensor(0.0051, device='cuda:0', grad_fn=<NllLossBackward>)
1287
tensor(0.4181, device='cuda:0', grad_fn=<NllLossBackward>)
1386
tensor(0.0018, device='cuda:0', grad_fn=<NllLossBackward>)
1485
tensor(0.0071, device='cuda:0', grad_fn=<NllLossBackward>)
1584
tensor(0.0015, device='cuda:0', grad_fn=<NllLossBackward>)
1683
tensor(0.0010, device='cuda:0', grad_fn=<NllLossBackward>)
1782
tensor(0.0414, device='cuda:0', grad_fn=<NllLossBackward>)
1881
tensor(0.0009, device='cuda:0', grad_fn=<NllLossBackward>)
1980
tensor(0.0171, device='cuda:0', grad_fn=<NllLossBackward>)
2079
tensor(0.0011, device='cuda:0', grad_fn=<NllLossBackward>)
2178
tensor(0.0008, device='cuda:0', grad_fn=<NllLossBackward>)
2277
tensor(0.0009, device='cuda:0', grad_fn=<NllLossBackward>)
2376
tensor(0.0024, device='cuda:0', grad_fn=<NllLossBackward>)
2475
tensor(0.0012, device='cuda:0', grad_fn=<NllLossBackward>)
1 epoch terminated. Evaluating on validation....
tensor([0, 0, 0,  ..., 0, 1, 0]) tensor([0, 0, 0,  ..., 0, 1,
0])
Precision =  0.9109756097560976
recall =  0.9188191881918819
F1 score =  0.9148805878750766
accuracy =  0.913125
99
tensor(0.0012, device='cuda:0', grad_fn=<NllLossBackward>)
198
tensor(0.0196, device='cuda:0', grad_fn=<NllLossBackward>)
297
tensor(0.0016, device='cuda:0', grad_fn=<NllLossBackward>)
396
tensor(0.0014, device='cuda:0', grad_fn=<NllLossBackward>)
495
tensor(0.0048, device='cuda:0', grad_fn=<NllLossBackward>)
594
tensor(0.0010, device='cuda:0', grad_fn=<NllLossBackward>)
693
tensor(0.0008, device='cuda:0', grad_fn=<NllLossBackward>)
792
tensor(0.0008, device='cuda:0', grad_fn=<NllLossBackward>)
891
tensor(0.0011, device='cuda:0', grad_fn=<NllLossBackward>)
990
tensor(0.0008, device='cuda:0', grad_fn=<NllLossBackward>)
1089
tensor(0.0005, device='cuda:0', grad_fn=<NllLossBackward>)
1188
tensor(0.0008, device='cuda:0', grad_fn=<NllLossBackward>)
1287
tensor(0.3447, device='cuda:0', grad_fn=<NllLossBackward>)
1386
tensor(0.0007, device='cuda:0', grad_fn=<NllLossBackward>)
```

```
1485
tensor(0.0008, device='cuda:0', grad_fn=<NllLossBackward>)
1584
tensor(0.0006, device='cuda:0', grad_fn=<NllLossBackward>)
1683
tensor(0.0007, device='cuda:0', grad_fn=<NllLossBackward>)
1782
tensor(0.0007, device='cuda:0', grad_fn=<NllLossBackward>)
1881
tensor(0.0007, device='cuda:0', grad_fn=<NllLossBackward>)
1980
tensor(0.0315, device='cuda:0', grad_fn=<NllLossBackward>)
2079
tensor(0.0008, device='cuda:0', grad_fn=<NllLossBackward>)
2178
tensor(0.0006, device='cuda:0', grad_fn=<NllLossBackward>)
2277
tensor(0.0008, device='cuda:0', grad_fn=<NllLossBackward>)
2376
tensor(0.0007, device='cuda:0', grad_fn=<NllLossBackward>)
2475
tensor(0.0007, device='cuda:0', grad_fn=<NllLossBackward>)
1 epoch terminated. Evaluating on validation....
tensor([0, 0, 0,  ..., 0, 1, 0]) tensor([0, 0, 0,  ..., 0, 1,
0])
Precision =  0.9062119366626066
recall =  0.915129151291513
F1 score =  0.9106487148102816
accuracy =  0.90875
```