Homework 1

by

Krima Doshi
1222303329

CSE576: Natural Language Processing

Masters in Computer Science

Arizona State University
September 2, 2021

1 Using the Sentiment140 dataset available on Kaggle, apply BERT Encoding and MLP to train, validate and test a model for sentiment analysis.

1.1 Comparison of hyper-parameters tuned in model:

Batch Size	**Max length**	**Layer Topology**	**Epoch**	**Learning Rate**	**Accuracy (Training, Validation, Testing)**
64	512	(512, 256, 64, 32, 1)	(20,5)	0.5	80.48, 73.37, 73.20
64	512	(512, 256, 64, 32, 1)	25	0.5	80.48, 73.37, 73.20
64	512	(512, 256, 64, 32, 1)	25	0.1	82.67, 67.07, 67.10
64	512	(512, 256, 64, 32, 1)	50	0.1	89.34, 68.80, 66.23
64	512	(512, 256, 64, 32, 1)	100	0.01	74.62, 59.79, 60.29
64	64	(64, 128, 64, 32, 1)	47	0.5	81.49, 68.16, 68.07
64	64	(64, 256, 64, 32, 1)	50	0.1	70.01, 57.5, 57.91
16	32	(64, 256, 64, 32, 1)	50	0.1	80.27, 68.6, 67.98
16	64	(64, 256, 64, 32, 1)	(32,15)	(0.1,0.01)	80.95, 67.96, 67.67

1.2 Observations/Notes:

- Only one output node and translated them into 0s and 1s using a threshold 0.
- 5 epochs were not sufficient to get the desired accuracy so the model was trained for longer epochs.
- With a learning rate of ¡0.1, the model was taking too long to train and converge, so a higher learning rate was used. In some cases, the model also got stuck at the local minima with that learning rate.

- For some parameters, instead of training continuously, resetting the SGD helped increase the accuracy.

1.3 Output Graphs:

Graphs of some noteworthy observations:

1. I would consider this case the best as it not only got a good accuracy but predicted the given sentences correctly.

```
[53] # Post-training evaluation
     evaluate_model(train_inputs, train_labels, data_name="Training")
     evaluate_model(validation_inputs, validation_labels, data_name="Validation")
     evaluate_model(test_inputs, test_labels, data_name="Testing")

     Training
             Loss: 0.6042267084121704
             Accuracy: 0.8048285714285714
     Validation
             Loss: 0.6400972604751587
             Accuracy: 0.7337333333333333
     Testing
             Loss: 0.6404624581336975
             Accuracy: 0.7320333333333333
```

```
def predict_sentence(sentence):
    test_sents=[sentence]
    test_sents = [tokenizer.convert_tokens_to_ids(tokenize_truncate(sent, tokenizer, max_input_length)) for sent in test_sents]
    sent_inputs = pad_sequences(test_sents, maxlen=max_input_length, dtype="long", truncating="post", padding="post")
    sent_inputs = torch.LongTensor(sent_inputs)
    # test_labels = torch.FloatTensor(test_labels.to_list())
    # Initiate model in evaluation mode
    model.eval()
    # Squeeze output to 1D
    pred_labels = model(sent_inputs)
    return "pos" if (pred_labels>THRESHOLD).float() else  "neg"

print(predict_sentence("It was a good experience"))
print(predict_sentence("It was a bad experience"))
print(predict_sentence("That was terrible"))
```

```
pos
neg
neg
```

Figure 1: Model Accuracy and Loss with 20 and 5 epochs and 0.5 learning rate.

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')

fig = plt.figure()
ax = plt.axes()

# x = np.linspace(0, 10, 1000)
ax.plot(range(EPOCHS), train_accuracy)
ax.plot(range(EPOCHS), validation_accuracy)
# train_accuracy
```

```
[<matplotlib.lines.Line2D at 0x7f77067ef990>]
```

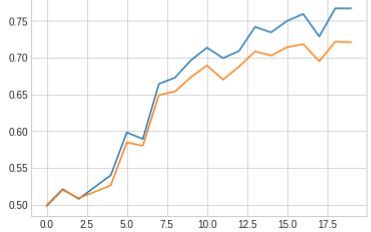

Figure 2: Training Accuracy and validation accuracies per epoch

```
# Post-training evaluation
evaluate_model(train_inputs, train_labels, data_name="Training")
evaluate_model(validation_inputs, validation_labels, data_name="Validation")
evaluate_model(test_inputs, test_labels, data_name="Testing")
```

```
Training
        Loss: 0.5899195671081543
        Accuracy: 0.8267714285714286
Validation
        Loss: 0.6674506664276123
        Accuracy: 0.6706666666666666
Testing
        Loss: 0.6665695905685425
        Accuracy: 0.6710666666666667
```

```
def predict_sentence(sentence):
    test_sents=[sentence]
    test_sents = [tokenizer.convert_tokens_to_ids(tokenize_truncate(sent, tokenizer, max_input_length)) for sent in test_sents]
    sent_inputs = pad_sequences(test_sents, maxlen=max_input_length, dtype="long", truncating="post", padding="post")
    sent_inputs = torch.LongTensor(sent_inputs)
    # test_labels = torch.FloatTensor(test_labels.to_list())
    # Initiate model in evaluation mode
    model.eval()
    # Squeeze output to 1D
    pred_labels = model(sent_inputs)
    return "pos" if (pred_labels>THRESHOLD).float() else  "neg"

print(predict_sentence("It was a good experience"))
print(predict_sentence("It was a bad experience"))
print(predict_sentence("That was terrible"))
```

```
pos
neg
pos
```

Figure 3: Model Accuracy and Loss with 25 epochs and 0.1 learning rate.

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')

fig = plt.figure()
ax = plt.axes()

# x = np.linspace(0, 10, 1000)
ax.plot(range(len(train_accuracy)), train_accuracy)
ax.plot(range(len(validation_accuracy)), validation_accuracy)
# train_accuracy
```

```
[<matplotlib.lines.Line2D at 0x7f1f2d30de90>]
```

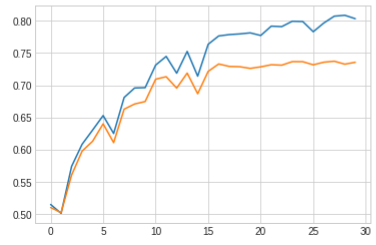

Figure 4: Training Accuracy and validation accuracies per epoch

2.

```
[18] # Post-training evaluation
     evaluate_model(train_inputs, train_labels, data_name="Training")
     evaluate_model(validation_inputs, validation_labels, data_name="Validation")
     evaluate_model(test_inputs, test_labels, data_name="Testing")

     Training
             Loss: 0.5562483072280884
             Accuracy: 0.8934
     Validation
             Loss: 0.6682621836662292
             Accuracy: 0.6680666666666667
     Testing
             Loss: 0.6710196733474731
             Accuracy: 0.6623666666666667
```

```
def predict_sentence(sentence):
    test_sents=[sentence]
    test_sents = [tokenizer.convert_tokens_to_ids(tokenize_truncate(sent, tokenizer, max_input_length)) for sent in test_sents]
    sent_inputs = pad_sequences(test_sents, maxlen=max_input_length, dtype="long", truncating="post", padding="post")
    sent_inputs = torch.LongTensor(sent_inputs)
    # test_labels = torch.FloatTensor(test_labels.to_list())
    # Initiate model in evaluation mode
    model.eval()
    # Squeeze output to 1D
    pred_labels = model(sent_inputs)
    return "pos" if (pred_labels>THRESHOLD).float() else  "neg"

print(predict_sentence("It was a good experience"))
print(predict_sentence("It was a bad experience"))
print(predict_sentence("That was terrible"))
```

```
pos
neg
pos
```

Figure 5: Model Accuracy and Loss with 50 epochs and 0.1 learning rate.

3.

```
[25] # Post-training evaluation
     evaluate_model(train_inputs, train_labels, data_name="Training")
     evaluate_model(validation_inputs, validation_labels, data_name="Validation")
     evaluate_model(test_inputs, test_labels, data_name="Testing")

     Training
             Loss: 0.6222451329231262
             Accuracy: 0.7462857142857143
     Validation
             Loss: 0.6969066858291626
             Accuracy: 0.5979333333333333
     Testing
             Loss: 0.6943069100379944
             Accuracy: 0.6029
     (0.6943069100379944, 0.6029)
```

```
def predict_sentence(sentence):
    test_sents=[sentence]
    test_sents = [tokenizer.convert_tokens_to_ids(tokenize_truncate(sent, tokenizer, max_input_length)) for sent in test_sents]
    sent_inputs = pad_sequences(test_sents, maxlen=max_input_length, dtype="long", truncating="post", padding="post")
    sent_inputs = torch.LongTensor(sent_inputs)
    # test_labels = torch.FloatTensor(test_labels.to_list())
    # Initiate model in evaluation mode
    model.eval()
    # Squeeze output to 1D
    pred_labels = model(sent_inputs)
    return "pos" if (pred_labels>THRESHOLD).float() else  "neg"

print(predict_sentence("It was a good experience"))
print(predict_sentence("It was a bad experience"))
print(predict_sentence("That was terrible"))
```

```
pos
pos
neg
```

Figure 6: Model Accuracy and Loss with 100 epochs and 0.01 learning rate.

4.