Name:

Yanlin Du

Problem Definition and Data Curation:

For this assignment, I worked on the 20 Newsgroups dataset and trained a model to classify the news category.
Here's the introduction of the dataset: The 20 Newsgroups dataset comprises around 18,000 newsgroup posts on 20 topics split into two subsets: one for training (or development) and the other one for testing (or for performance evaluation). The split between the train and test set is based upon messages posted before and after a specific date.

I chose to train a simple RNN model with Word2Vec as word embeddings.

Results and how they presented

Based on the evaluation result, I got around a 7% accuracy rate, which is higher than random guessing (5%).

In-depth analysis

Though my model is performing better than random guessing, I am still surprised by the low accuracy rate.

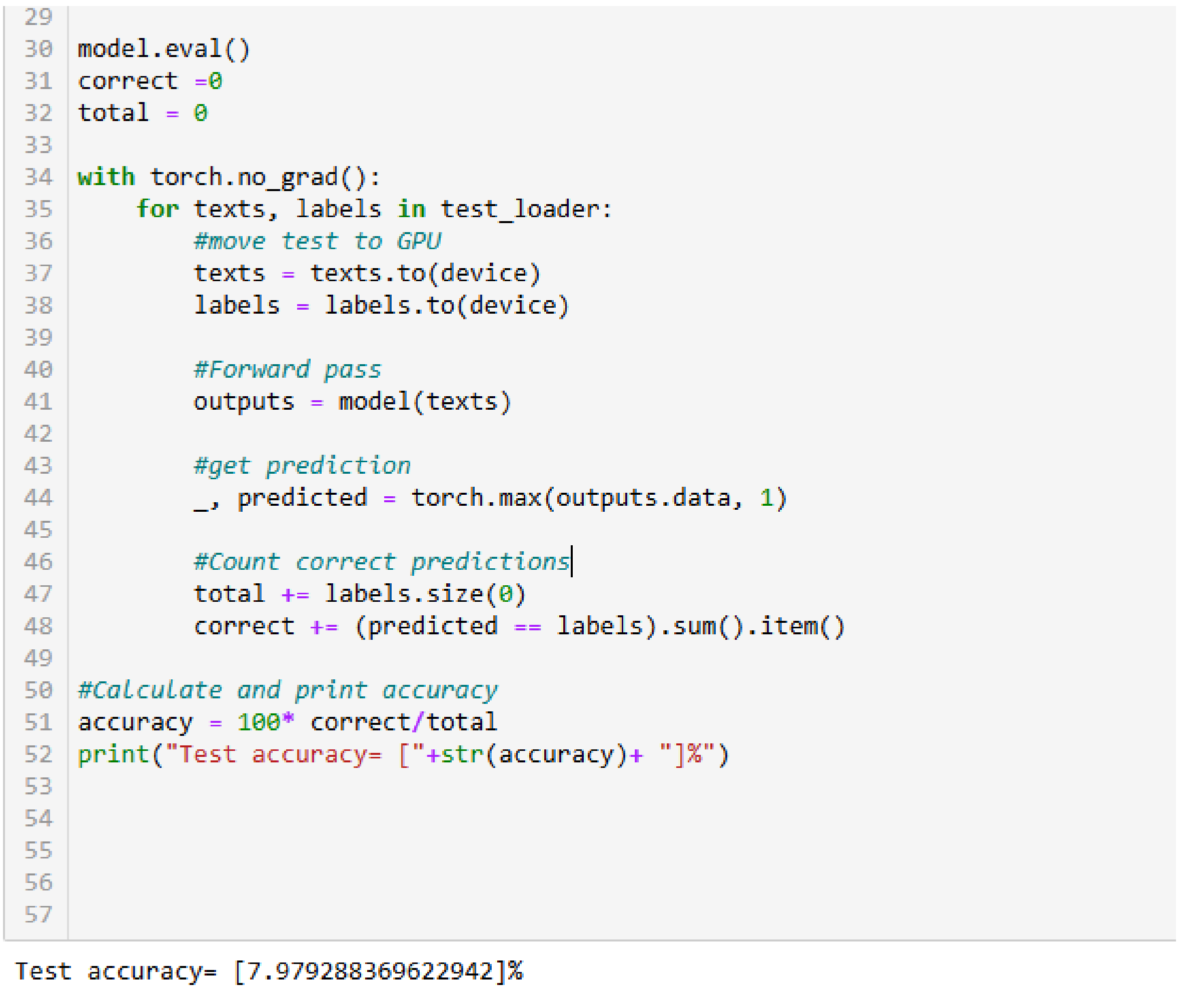
Based on what we learned from class.

I assume if I make these changes, the performance could be better:

1. Build a more complex model: My current model is relatively simple, so adding more hidden layers should improve performance.
2. Train more epochs: I assume training for more epochs will give me better results.
3. Add dropout layers: I haven't added dropout layers for simplicity, but I assume the performance will be better if I add dropout layers.

Lessons & Experience you learned in this project

This is the first time I tried to build a model from scratch and work on a dataset, I feel I learned a lot and started to understand the class concept more. Also I realize if I increase the layer more, it might even reduce the accuracy and I believe that is over fitting.



```python

model.eval()
correct =0
total = 0

with torch.no_grad():
    for texts, labels in test_loader:
        #move test to GPU
        texts = texts.to(device)
        labels = labels.to(device)

        #Forward pass
        outputs = model(texts)

        #get prediction
        _, predicted = torch.max(outputs.data, 1)

        #Count correct predictions
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

#Calculate and print accuracy
accuracy = 100* correct/total
print("Test accuracy= ["+str(accuracy)+ "]%")
```

```
Test accuracy= [7.979288369622942]%
```

As the result, the performance is around 8% accuracy which is higher than 5%