Assignment 3 Report – Deep Learning Course
Lyrics Classification Using LSTM

Faisal Omari – 325616894 – faisalomari321@gmail.com
Saji Assi – 314831207 – sajiassi86@gmail.com

Due Date: 22/3/2024

In the assignment we were requested to implement LSTM Network that classifies lyrics for 3 different artists.
We had a problem in getting a good accuracy of 70%+, but still the best accuracy we got is around 60% (which is the best saved model that the file of the test runs on) while in the best epoch during the batches we had an accuracy of 72% which is considered good for this difficult classification problem.
During the code implementation we have tried a lot of approaches to solve the problem and make it easier, the approaches are for Q1 and Q2:

1. Report your architecture, number of parameters, training time (real life time) and performance on the test set.
2. Add a plot showing train error, test error, and accuracy as a function of Global batch steps performed (not epochs!).

- The model was trained and tested on PC with GTX1060ti 6GB GPU.
- The batch size is 64, and no. of epochs is 150, while the best model (and the saved models) were selected according to the best accuracy while evaluating the model on the test set during the training loop.

1) Training the model with normal tokenization, without cleaning the input, with two LSTM layers, this gave as an accuracy of 67% and a loss of 1.68 for the best epoch. Architecture:

Layer	Parameters
Embedding	Vocab_size = len(vocab), embedding_dim = 500
LSTM	Hidden_dim = 256
LSTM	Hidden_dim = 256
FC	Hidden_dim = 256, output_dim = 3 (no. of artists)

Figure 1, with epochs and not global batched steps.

Run time: 150min.

2) Training the model with normal tokenization, without cleaning the input, with two layers of LSTM, this gave as an accuracy of 67% and a loss of 1.793.
Architecture:

Layer	Parameters
Embedding	Vocab_size = len(vocab), embedding_dim = 500
LSTM	Hidden_dim = 512
LSTM	Hidden_dim = 512
FC	Hidden_dim = 512, output_dim = 3 (no. of artists)

Figure 2, with epochs and not global batched steps.

Run time: 220min.

3) Training the model with normal tokenization, with cleaning the input from some phrases like (don't, \n, I, you, your…), with two layers of LSTM, this gave as an accuracy of 41.2% and a loss of 1.099, the accuracy is not good at all, but we can see that the loss in here is better than the previous two models.
Architecture:

Layer	Parameters
Embedding	Vocab_size = len(vocab), embedding_dim = 500
LSTM	Hidden_dim = 128
LSTM	Hidden_dim = 128
FC	Hidden_dim = 128, output_dim = 3 (no. of artists)

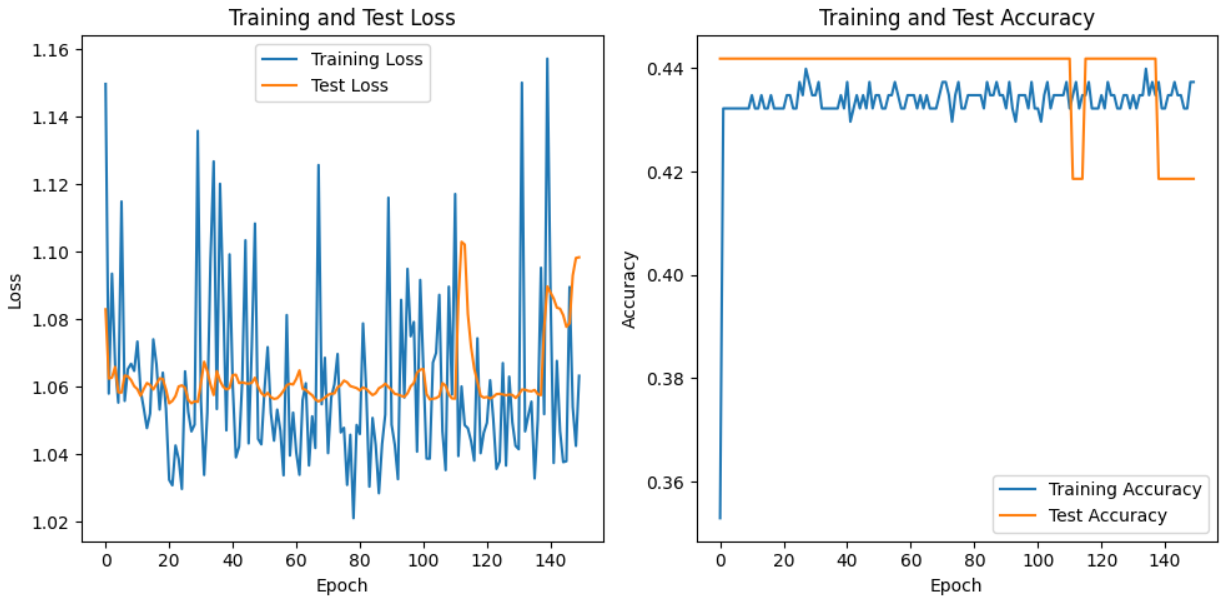

Figure 3, with epochs and not global batched steps.

Run time: 190min.

4) This is the best model we got:
Training the model with normal tokenization, without cleaning the datasets, with two layers of LSTM, this gave as an accuracy of 60.5% and a loss of 0.98, the accuracy is the best comparing to the previous models, and the loss is also the best comparing to the previous models, this model gave us an accuracy of 72% during the batches! (look at the attached graph)
Architecture:

Layer	Parameters
Embedding	Vocab_size = len(vocab), embedding_dim = 500
LSTM	Hidden_dim = 128
LSTM	Hidden_dim = 128
FC	Hidden_dim = 128, output_dim = 3 (no. of artists)

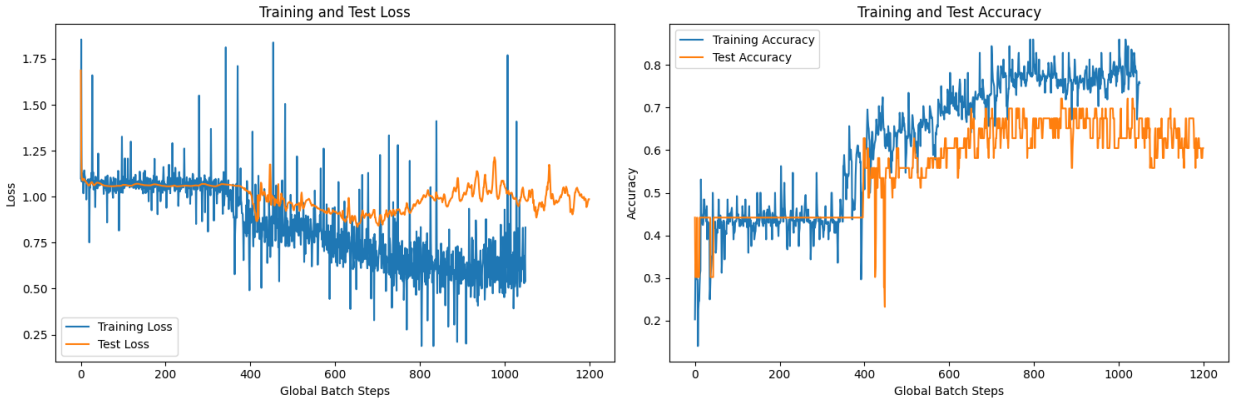

Figure 4, with global batch steps.

Run time: 130min.

3. Explain shortly how your design deals with inputs of varied length.

Since each input has a different size and the lyrics are not in the same length, we had to deal with this problem using collate function that does a padding process for all the inputs according to the input with the longest size, for example if there is 3 lyrics with the following lengths: 718, 199, 829 ➔ then the padding is going to be to 829 since it is the longest lyric. The padding value is 0.

```
def collate_batch(batch):
    lyrics, artists = zip(*batch)
    lyrics_tensor = [text_to_tensor(lyric) for lyric in lyrics]
    lyrics_tensor_padded = pad_sequence(lyrics_tensor, padding_value=0, batch_first=True)
    artists_tensor = torch.tensor([artist_to_label(artist) for artist in artists], dtype=torch.long)
    return lyrics_tensor_padded, artists_tensor
```

4. Explain shortly what LSTM you propagated forwards, and why.

The code utilizes LSTM networks for processing sequential data, such as song lyrics. LSTMs are chosen because they are adept at capturing long-range dependencies within sequences, making them suitable for tasks like lyric generation and artist

classification. The LSTM architecture consists of embedding layers to convert words into vectors, LSTM layers to process sequences, and fully connected layers for classification. During training, the model adjusts its parameters to minimize a chosen loss function. After training, the model is evaluated on a separate test set to assess its performance.

There are 3 python files attached to the submission zip:

- **Main.py** for training and evaluating the model.
- **Test_model.py** to test the model on the best model.
- **Clean.py** the code used to clean the trainset for the 3rd model.

In addition to the saved model (pth format) of the 4th model, you can test by the test_model.py file mentioned above.