DL-Ops Assignment-2 Report

Task: Implement a neural network and utilize and analyze RNN and LSTMs.

Objectives: Seq2Seq modeling.

Question 1.

- Build a seq2seq model which contains the following layers
- input layer for character embeddings
- one encoder which sequentially encodes the input character sequence (Latin)
- one decoder which takes the last state of the encoder as an input and produces one character output at a time (native).

Question 2.

- Loss vs. hyper-params.
- Input embedding size: 16, 64
- number of encoder layers: 1,3
- number of decoder layers: 1,3
- hidden layer size: 16,64

Question 3.

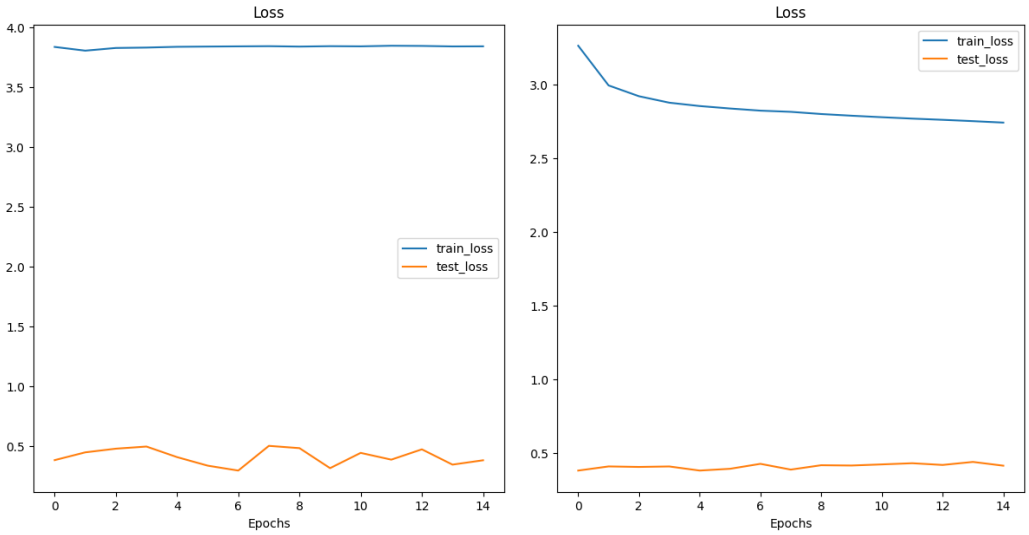
- Does RNN takes less time to converge than LSTM? Reason to support your answer.
- Does dropout leads to better performance? Proof to support your answer.
- Using a smaller size in hidden layer does not give good results? Proof to support you result.

Question 4.

- Adding attentions.
- Plot the accuracy/loss.
- Report the test accuracy.
- Does the attention-based model performs better than the LSTM? Proof to support
- your answer.

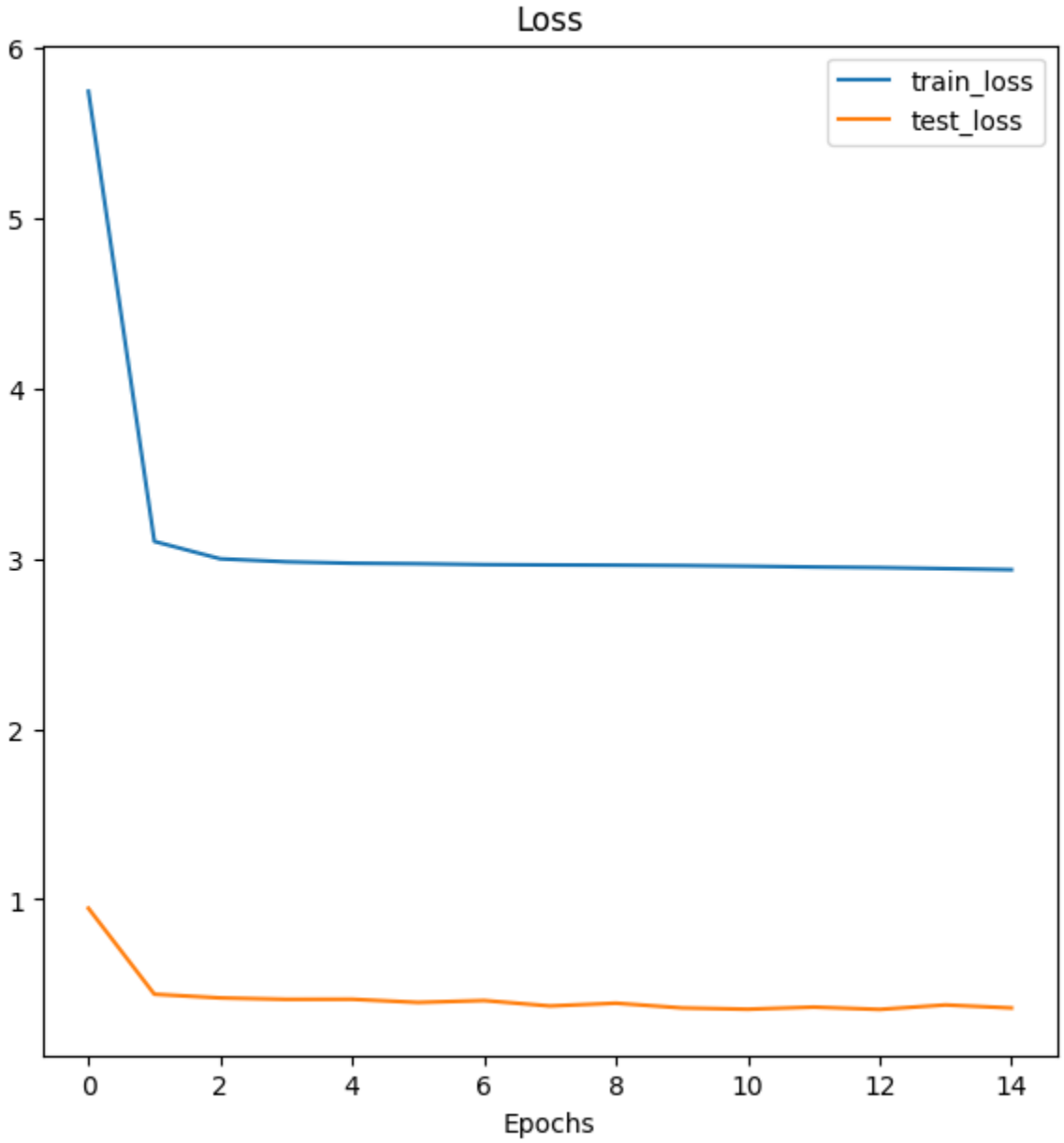
Procedure:

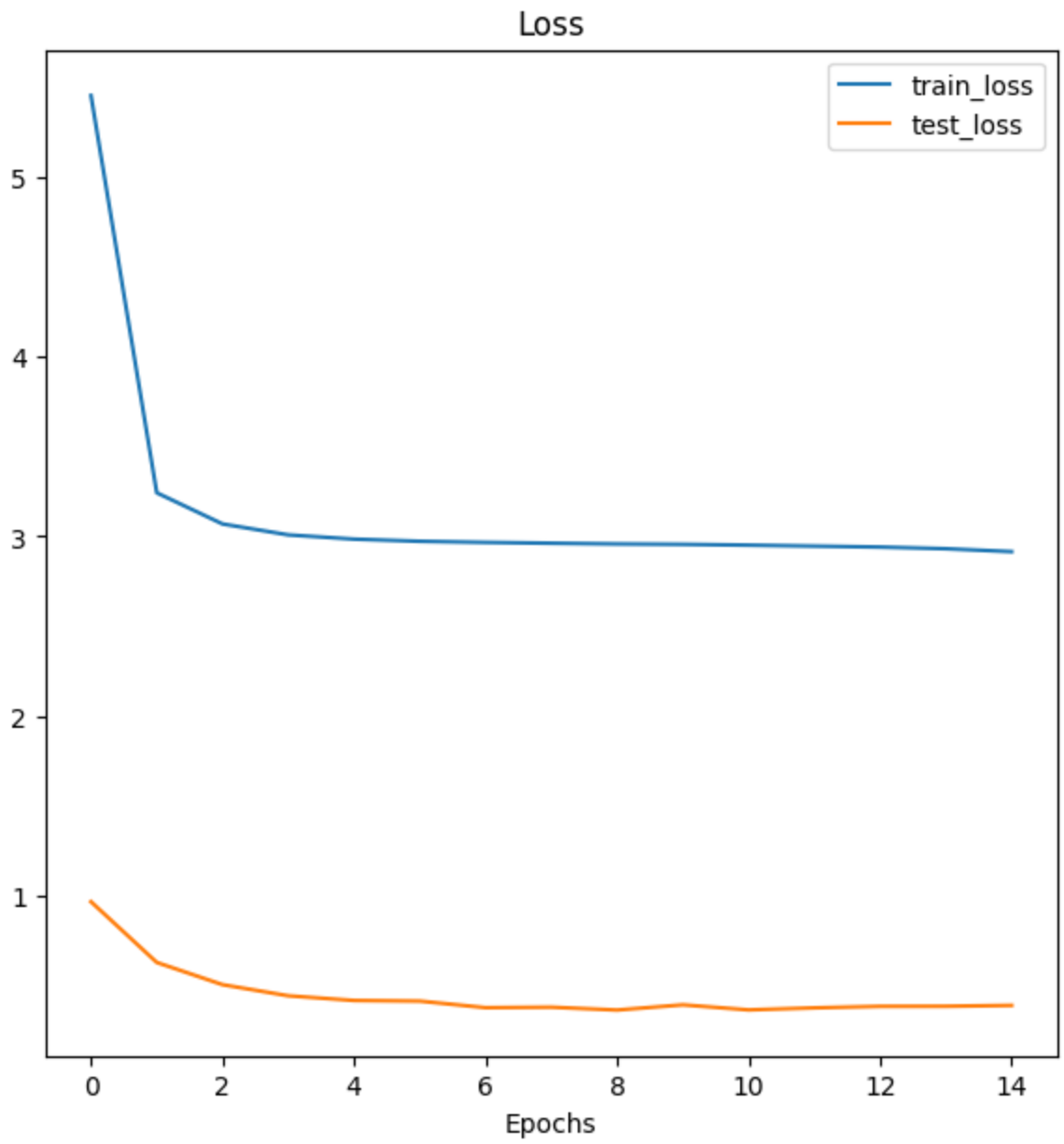
- Import required packages. (PyTorch, NumPy, Matplotlib, torchmetrics …etc.)
- Checking GPU availability.
- Setting up devices.
- Load the dataset with the help of pandas.
- Process the dataset with all possible pre-processing.
- Define Custom Dataset class for data iterators.
- Converting Dataset to Dataloaders.
- Define the Models for the Seq2Seq RNNs and LSTMs, respectively.
- Make utils functions to plot and train the models.
- Make Plot loss curves function, training step, testing step, and training loop for the model.
- Define model instances.
- Train the model using above functions.
- Results default RNN and LSTM models



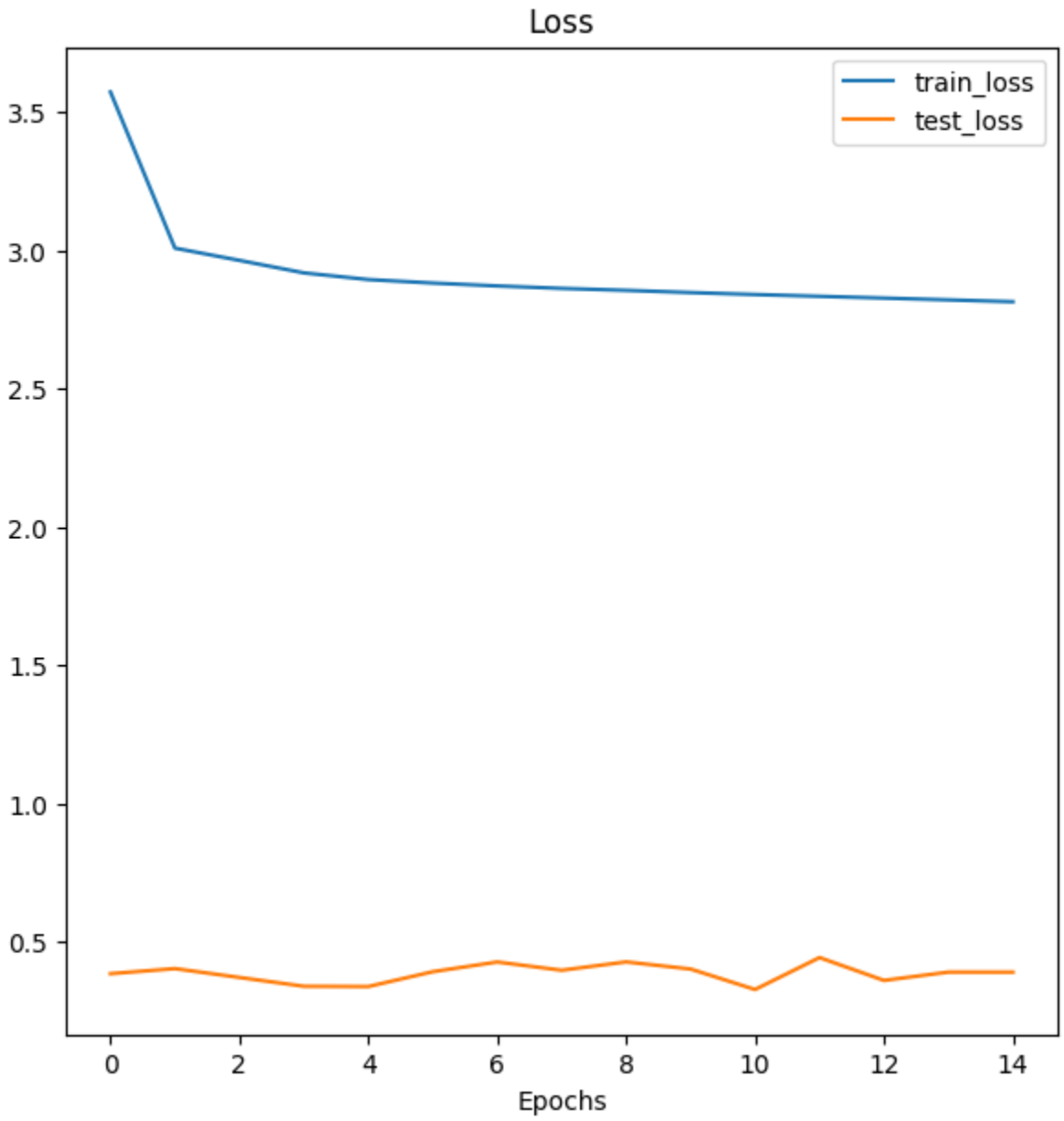
- Fig 1: is for rnn, shows that it is nit converting as much as lstm one, please note that left figure is for RNN and right is for LSTM.
- Results for loss vs hyperparms.

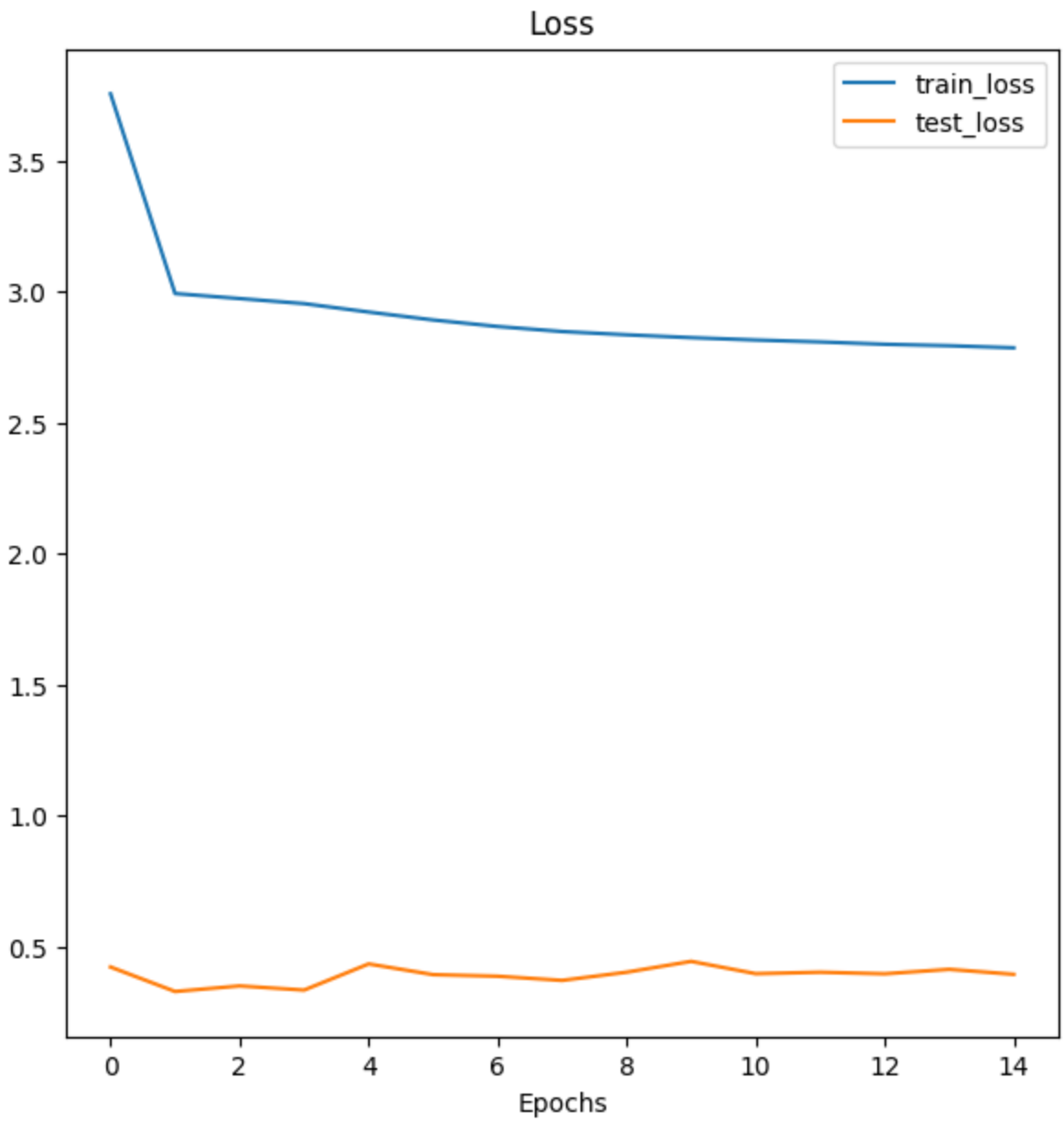
- SET 1



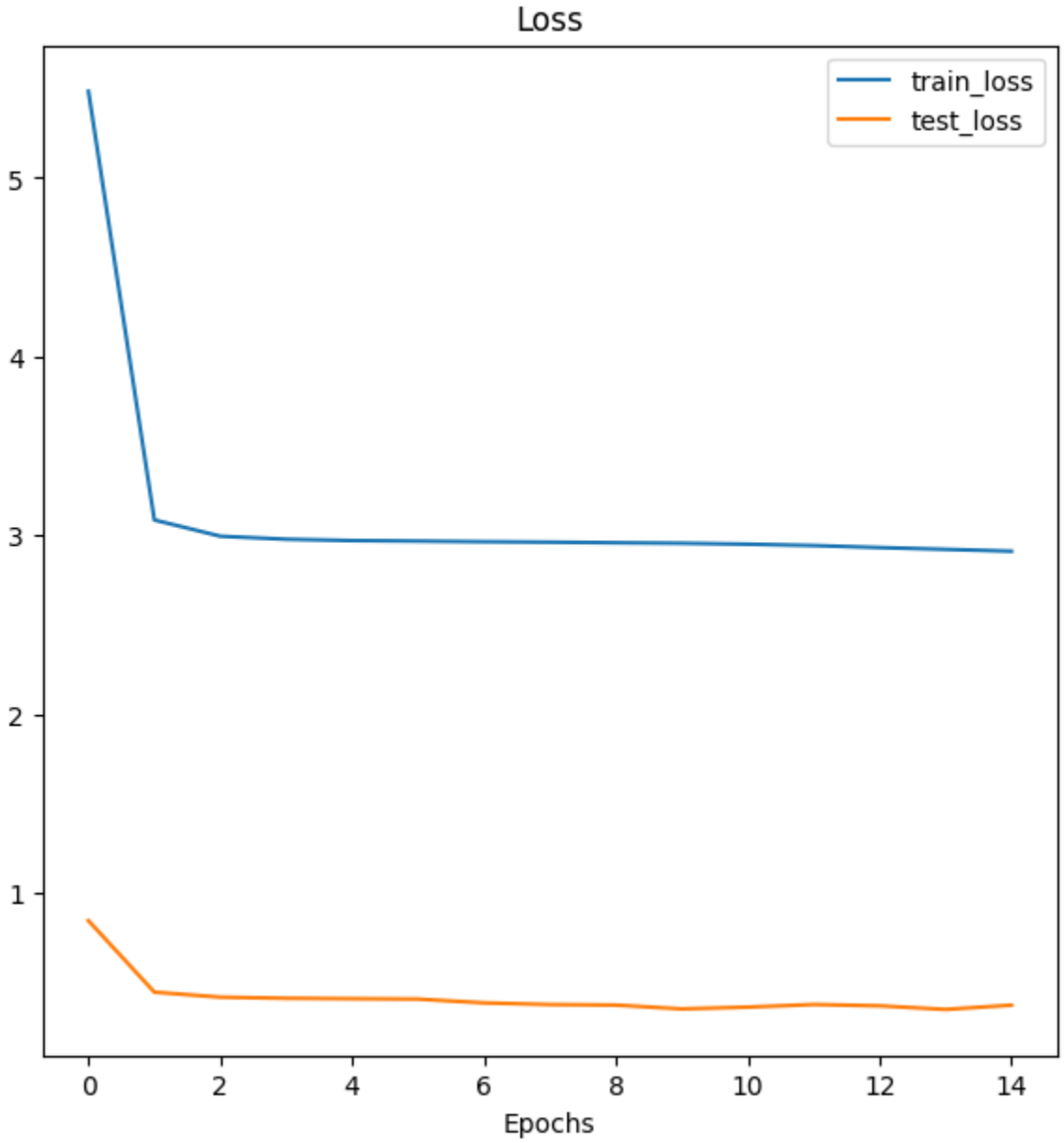

Loss
train_loss
test_loss
5
4
3
2
1
0
2
4
6
8
10
12
14
Epochs

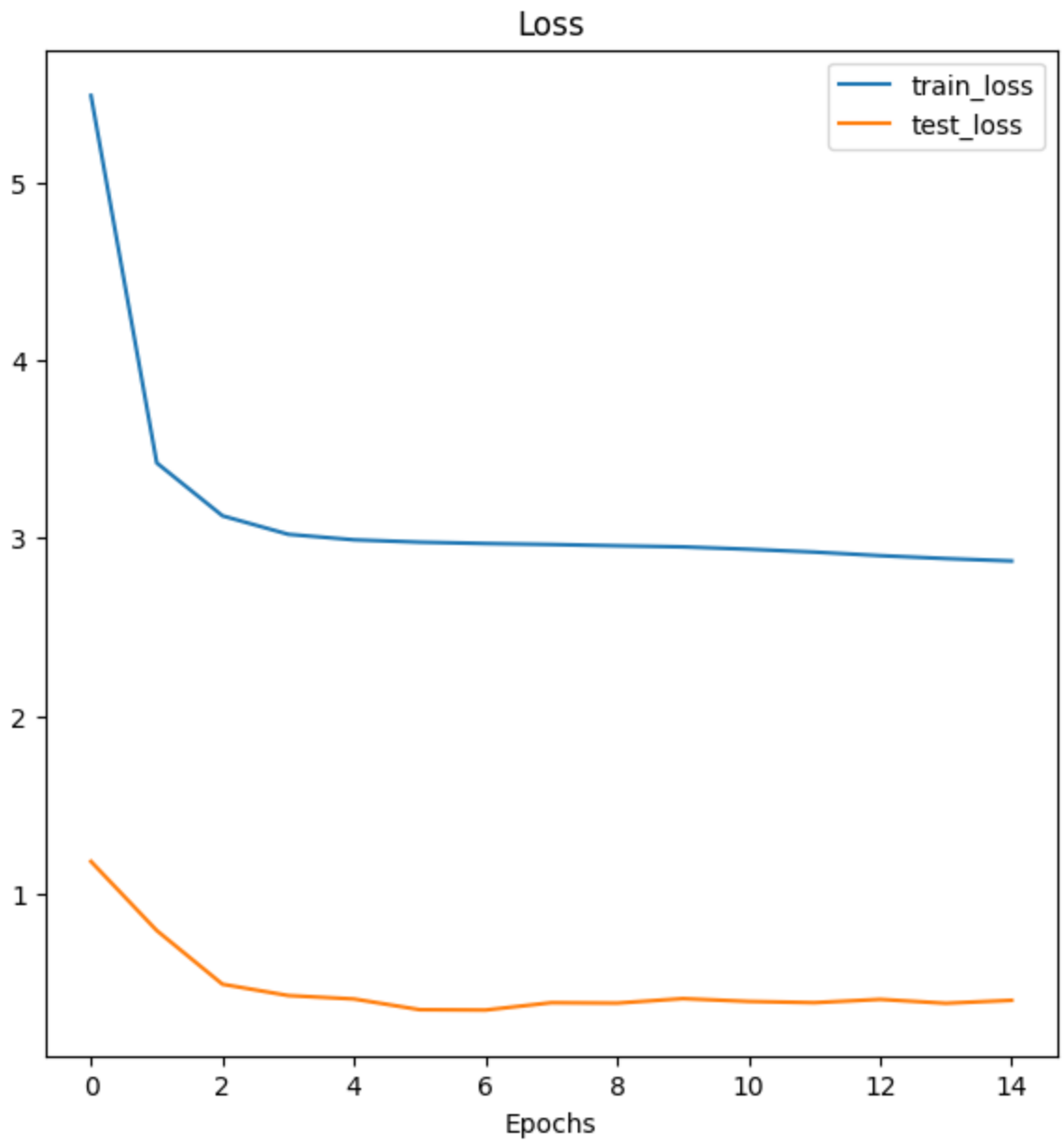
- Set 2


Loss
train_loss
test_loss
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
2
4
6
8
10
12
14
Epochs

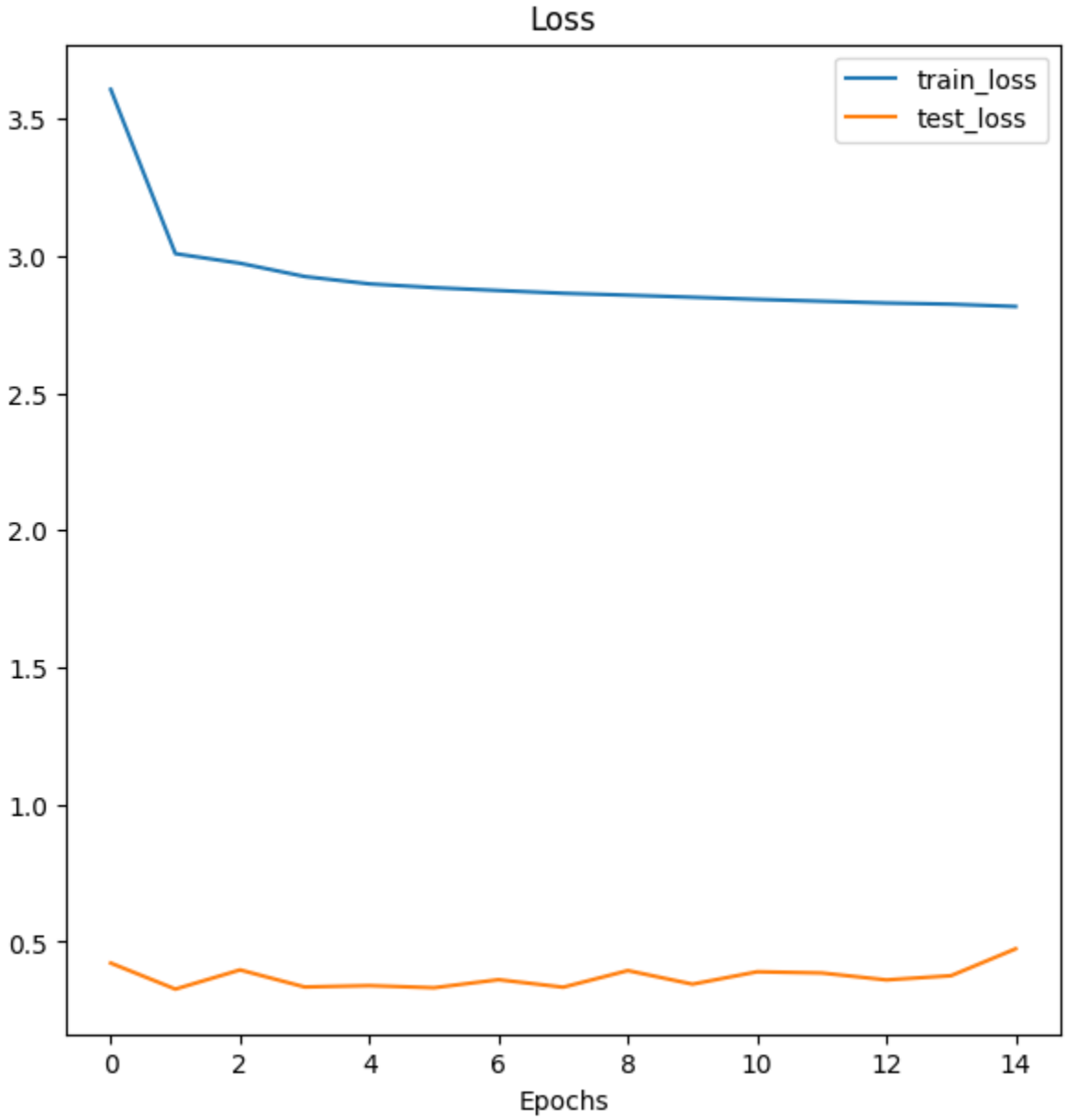

Loss
train_loss
test_loss
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
2
4
6
8
10
12
14
Epochs

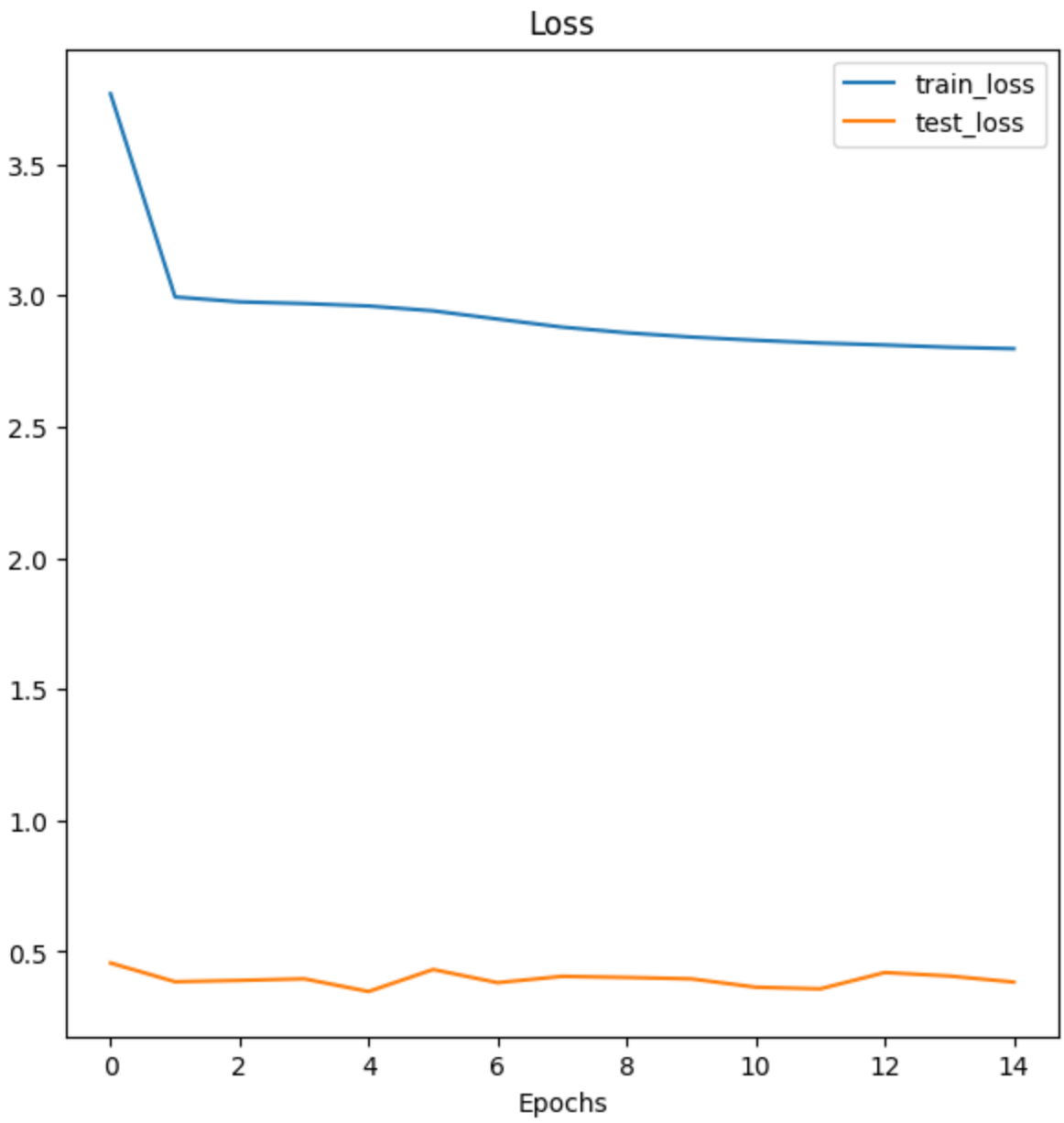
- Set 3




Loss
train_loss
test_loss
5
4
3
2
1
0
2
4
6
8
10
12
14
Epochs

- Set 4


Loss
train_loss
test_loss
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
2
4
6
8
10
12
14
Epochs



- The explanation for the graphs.
- The set 3 hyper params are performing best than others.
- This is happening because embedding size and the number of units used, also note that the rnn plots are much sharper than the lstms because of suddent changes in the O and H.
- It is not necessarily true that RNNs (Recurrent Neural Networks) take less time to converge than LSTMs (Long Short-Term Memory Networks) in all cases.
- The reason for this is that both RNNs and LSTMs have different architectures and properties that can affect their convergence time. RNNs are simpler and have fewer parameters than LSTMs, which could potentially make them faster to train and converge. However, RNNs also have a problem known as vanishing gradients, which can cause

the gradients to become very small during backpropagation and make it difficult for the network to learn long-term dependencies.

- LSTMs, on the other hand, were designed to overcome the vanishing gradient problem by introducing a more complex architecture that includes memory cells and gates. The gates allow the network to selectively forget or remember information over time, which makes it easier to learn long-term dependencies. However, the additional complexity of the LSTM architecture also means that it has more parameters to learn, which could potentially make it slower to converge.
- In practice, the convergence time of RNNs and LSTMs depends on the specific problem being solved, the size of the network, and the amount of training data. In some cases, RNNs may converge faster than LSTMs, while in other cases, LSTMs may converge faster. Ultimately, the best approach is to experiment with both architectures and choose the one that works best for the specific problem at hand.
- Does dropout leads to better performance, All the above graphs are shown for the training loops that used dropouts with p=0.5 and helps the model to convergre better.
- Using a smaller size in the hidden layer does not give good results and can be proved by looking at the plots for set 1.
- The LSTM with attention Model Performs better than regular LSTM.
- The Results for training the models with attentions.

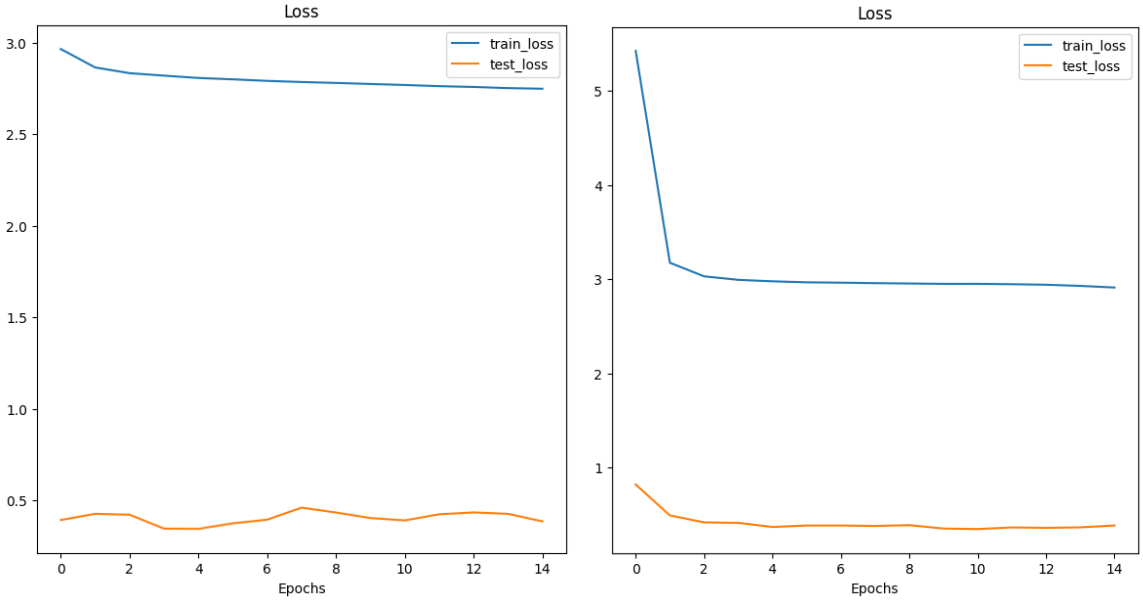


Fig: RNN with attention & LSTM with attention

- We can prove the better performance of LSTMs with attention using the loss curve shown above.

Objectives: Predict the global active power.

Question 1.

- Split the dataset into train and test (80:20) and do the basic preprocessing. [10]
- Use LSTM to predict the global active power while keeping all other important
- features and predict it for the testing days by training the model and plotting the real global active
- power and predicted global active power for the testing days and comparing the results.
- Now split the dataset in train and test (70:30) and predict the global active power for the testing days and compare the results with part (ii).

Procedure:

- Import required packages. (PyTorch, NumPy, Matplotlib, torchmetrics …etc.)
- Checking GPU availability.
- Setting up devices.
- Load the dataset with the help of pandas.
- Process the dataset with all possible pre-processing.
- Making 80:20 split the dataset into training testing.
- Define Custom Dataset class for data iterators.
- Converting Dataset to Dataloaders.
- Define the Models for the Seq2Seq RNNs and LSTMs, respectively.
- Make utils functions to plot and train the models.
- Make Plot loss curves function, training step, testing step, and training loop for the model.
- Define model instances.
- Train the model using above functions.
- Results for the predecions for LSTM:

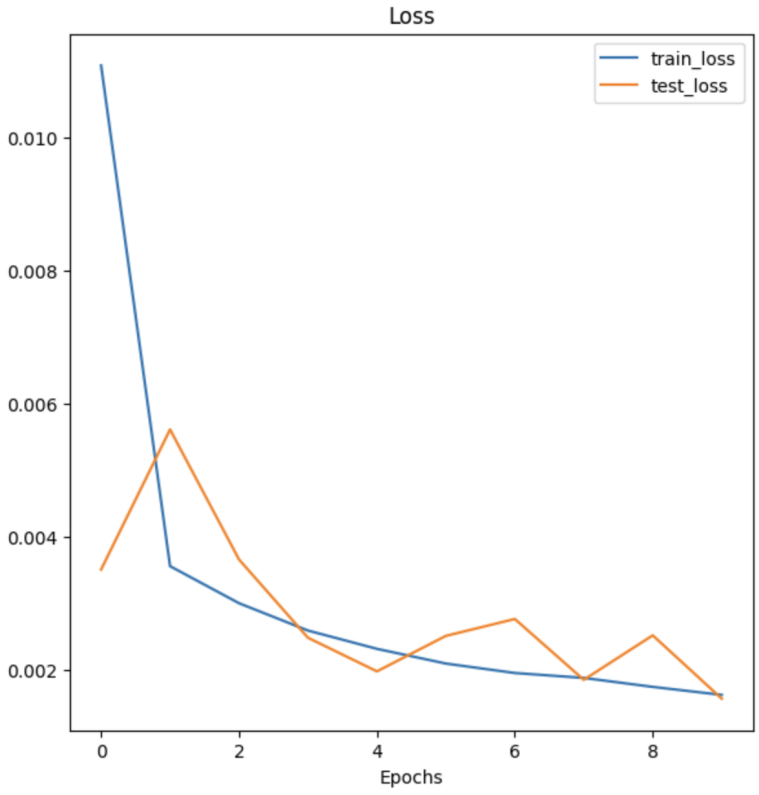


Fig: Training loss for the loss

- The above graph shows that the model can be further improved since test loss can be further minimized.
- Making Split for 70:30 and repeating the experiments.
- Result:

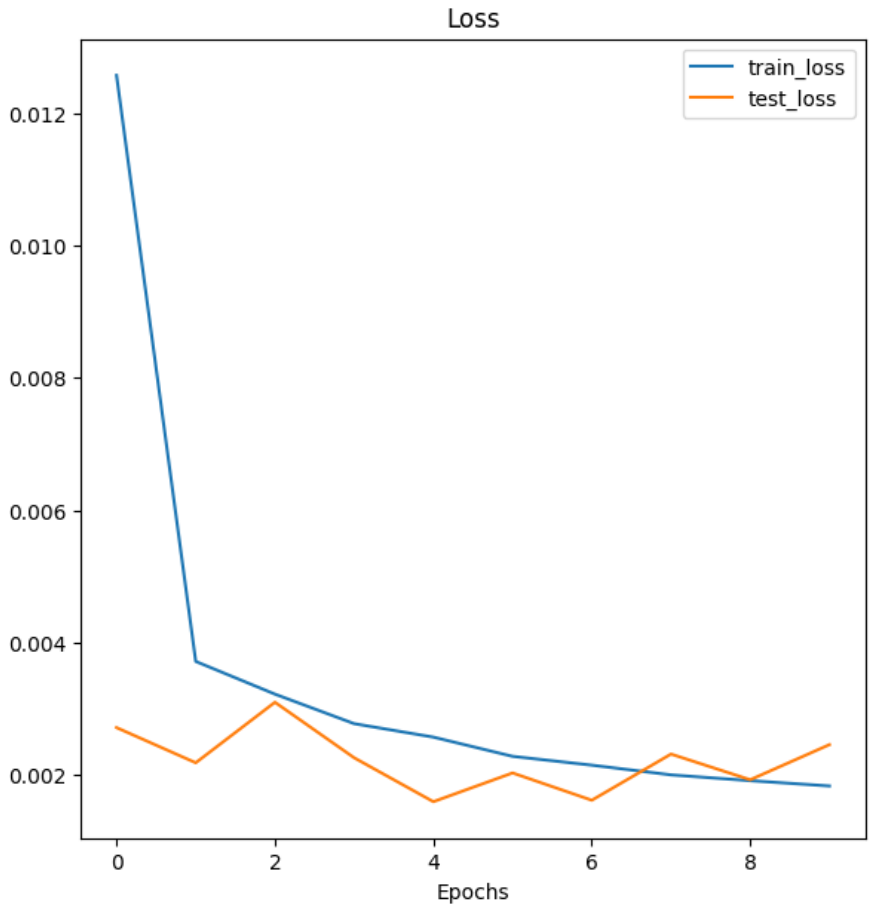


Fig: training loss for split 70:30

- This graph shows that the model started overfitting after the 8th epoch because of the 10 percent less data availability.
- Also, the loss values for 80:20 split are significantly less then the 70:30 split the reason is again the amount of data available during the training.

References:-

- https://pytorch.org/text/stable/index.html
- https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html
- https://github.com/bentrevett/pytorch-seq2seq
- https://www.kaggle.com/code/columbine/seq2seq-pytorch
- https://pytorch.org/tutorials/beginner/data_loading_tutorial.html