20XD68–DEEP LEARNING LAB

FINAL PACKAGE REPORT

Twitter Sentiment Analysis using RNN

20PD12- Karthic Sreenivas A

20PD39- Vishnupriya G

ABSTRACT:

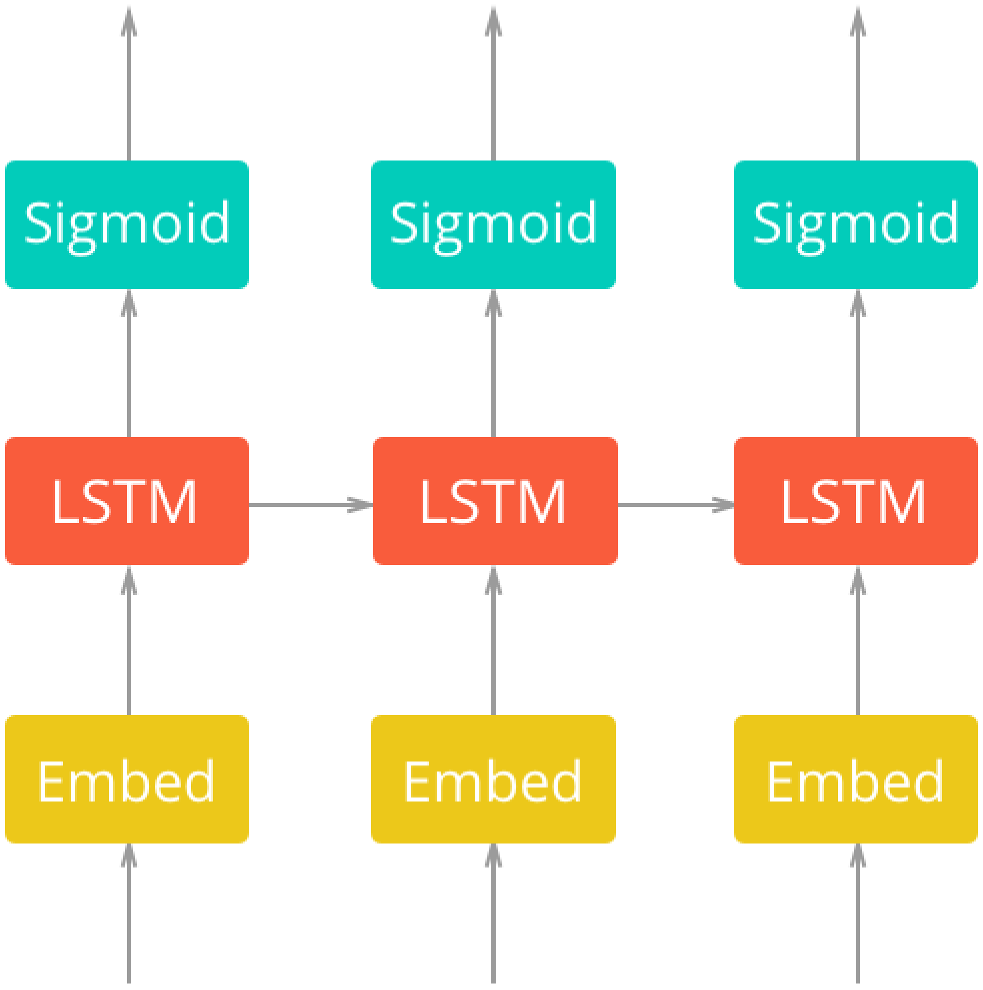
In this notebook, we implement a recurrent neural network to analyze how travelers in February 2015 expressed their feelings on Twitter. Using an RNN rather than a strictly feedforward network is more accurate since we can include information about the *sequence* of words.

A sentiment analysis job about the problems of each major U.S. airline. Twitter data was scraped from February of 2015 and contributors were asked to first classify positive, negative, and neutral tweets, followed by categorizing negative reasons (such as "late flight" or "rude service").

In this notebook, our goal is to identify whether a tweet is negative or non-negative (positive or neutral).

NETWORK ARCHITECTURE:

The architecture for this network is shown below:



- ★ **First, we'll pass in words to an embedding layer.** We need an embedding layer because we have tens of thousands of words, so we'll need a more efficient representation for our input data than one-hot encoded vectors.
- ★ **After input words are passed to an embedding layer, the new embeddings will be passed to LSTM and Bi-LSTM cells.** The LSTM cells will add *recurrent* connections to the network and give us the ability to include information about the *sequence* of words in the movie review data.
- ★ **Finally, the LSTM outputs will go to a sigmoid output layer.** We're using a sigmoid function because we have two classes (0 is negative and 1 is non-negative) and a sigmoid will output predicted sentiment values between 0-1.

We don't care about the sigmoid outputs except for the **very last one**; we can ignore the rest. We'll calculate the loss by comparing the output at the last time step and the training label.

DATASET:

The dataset - Twitter data was scraped from February of 2015 and contributors were asked to first classify positive, negative, and neutral tweets, followed by categorizing negative reasons (such as "late flight" or "rude service").

Shape of the dataset is as following :

- x_train: (11200, 30)

- y_train: (11200,)

- x_validation: (1400, 30)

- y_validation: (1400,)

- x_test: (1400, 30)

- y_test: (1400,)

- ★ **Pre-processing steps** include

  The first step when building a neural network model is getting the data into the proper form to feed into the network. Since we're using embedding layers, we'll need to encode each word with an integer. We'll also want to clean it up a bit.

  Here are the processing steps, we'll want to take:

  - We'll want to get rid of periods and extraneous punctuation.
  - We'll want to remove the web address, twitter id, and digit.

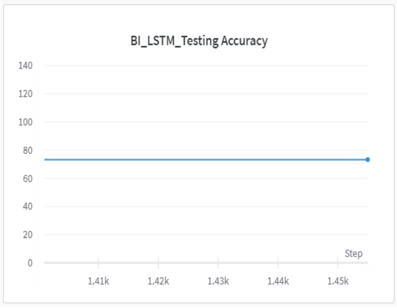
  First, let's remove all punctuation. Then get all the text without the newlines and split it into individual words.

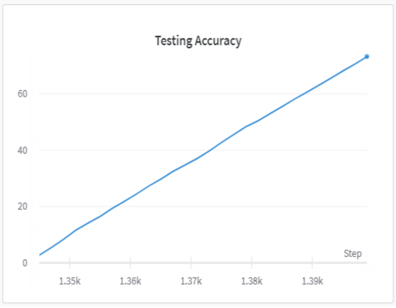
  Then, we remove the web address, twitter id, and digit.

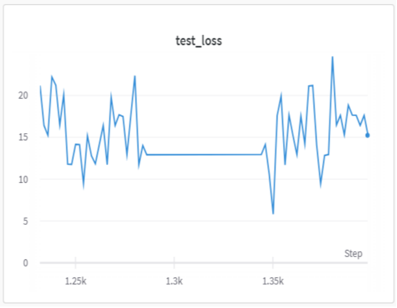
- ★ **Encoding the labels**
  - As mentioned before, our goal is to identify whether a tweet is negative or non-negative (positive or neutral). Our labels are "positive", "negative", or "neutral. To use these labels in our network, we need to convert them to 0 and 1.
- ★ **Padding sequences**
  - To deal with both short and very long reviews, we'll pad or truncate all our reviews to a specific length. For reviews shorter than some *seq_length*, we'll pad with 0s. For reviews longer than *seq_length*, we can truncate them to the first *seq_length* words. A good *seq_length*, in this case, is 30, because the maximum review length from the data is 32.
- ★ **DataLoaders and Batching**
  - After creating training, test, and validation data, we can create DataLoaders for this data by following two steps:
    - Create a known format for accessing our data, using TensorDataset which takes in an input set of data and a target set of data with the same first dimension, and creates a dataset.
    - Create DataLoaders and batch our training, validation, and test Tensor datasets.

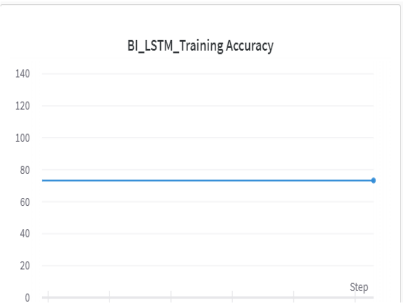
- **Network Architecture**
    - The layers are as follows:
        - An embedding layer that converts our word tokens (integers) into embeddings of a specific size.
        - An LSTM and Bi-LSTM layer defined by a *hidden_state* size and number of layers
        - A fully-connected output layer that maps the LSTM layer outputs to a desired *output_size*
        - A sigmoid activation layer which turns all outputs into a value 0-1; return only the last sigmoid output as the output of this network.
- **Instantiate the network**
    - Here, we'll instantiate the network. First up, defining the hyperparameters.
        - *vocab_size*: Size of our vocabulary or the range of values for our input, word tokens.
        - *output_size*: Size of our desired output; the number of class scores we want to output (negative/non-negative).
        - *embedding_dim*: Number of columns in the embedding lookup table; size of our embeddings.
        - *hidden_dim*: Number of units in the hidden layers of our LSTM cells. Usually larger is better performance wise. Common values are 128, 256, 512, etc.
        - *n_layers*: Number of LSTM layers in the network. Typically between 1-3.
- **Training**
    - Below is the typical training code. We'll use a cross entropy loss, which is designed to work with a single Sigmoid output. BCELoss, or Binary Cross Entropy Loss, applies cross entropy loss to a single value between 0 and 1. We also have some data and training hyper-parameters:
        - *lr*: Learning rate for our optimizer.
        - *epochs*: Number of times to iterate through the training dataset.
        - *clip*: The maximum gradient value to clip at (to prevent exploding gradients).
- **Testing**
    - We'll see how our trained model performs on all of our defined test_data, above. We'll calculate the average loss and accuracy over the test data.

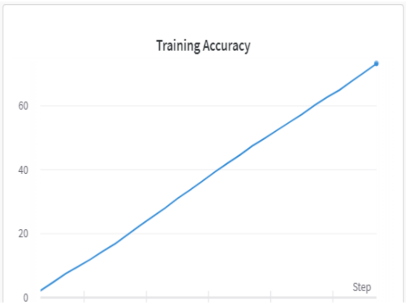
- ★ **Creating a predict function**
  - ○ We'll write a predict function that takes in a trained net, a plain text_review, and a sequence length, and prints out a custom statement for a non-negative or negative review.
- ★ **Improvement Done:**
  - ○ We have successfully implemented the Bi-LSTM model for the given dataset. We have created its respective function class, Model instance, fitted it into the training dataset and obtained accuracy values for all the models.
  - ○ We applied **TOMEK links Undersampling Technique** in order to overcome the skewness in the dataset where the **negative** sentiment tweets dominated.
  - ○ We also attempted to fit the GRU, Bi-GRU and Glove-Bi_LSTM models as an extension to LSTM and Bi-LSTM where the model function and instances worked perfectly, but when it came to integrating it with the training dataset, there was an error regarding the dimensions of the *input_tensor* which we found very hard to resolve even after referring to the documentation of PyTorch.
- ★ **Report Generation:**
  - ○ We have integrated **Weights and Biases** for report generation wherein details of training accuracy and test accuracy for every epoch has been monitored along with the disk utilization.

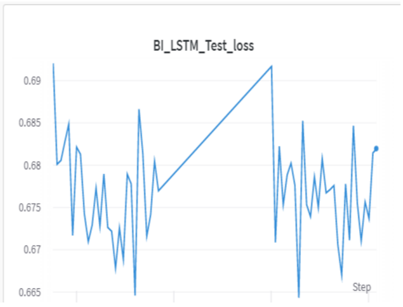


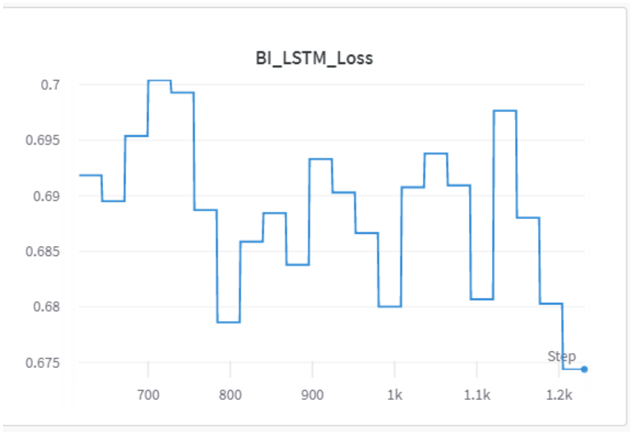


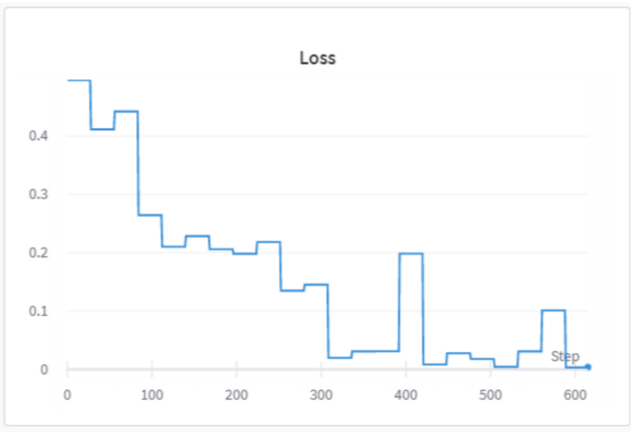












ENVIRONMENT:

The model is trained on **Google Colab Notebooks** environment.

ACCURACY:

- **LSTM** - 83.4%
- **Bi-LSTM** - 76.2% (No. of epochs = 10)