Deep Learning Humor Classification on Yelp Reviews

Neha Sajnani (17CE096)

Kalind Sarda (17CE101)

Vishal Shah (17CE118)

Abstract:

Humour is an abstract, high-level use of language that's largely subjective. We plan to build a model capable of recognizing humor within Yelp reviews. Despite the straightforward binary labels, we believe that this domain exemplifies real-world, subtle humor since the “reviews” aren’t forced to be “funny” all the time. Using a set of 120000 Yelp reviews, where we used the “funny” votes from other users as an indicator for humor, we trained a baseline shallow neural network model that simply averages the word embeddings to one vector and feeds it into a neural network. We then tried different models such as FFN, RNNs, and LSTMto see if it could capture more complexity and it did, with LSTM proving to be the best model because of it being more complex to be able to capture humor via sarcasm.

Introduction:

Classifying tongue text may be a popularly studied area, especially within the fields of sentiment analysis and opinion mining. In these fields, data scientists use various statistical methods to classify the emotions or opinions of the author of a bit of text, often supported the positive and negative connotations of words like "sarcasm," "explanation" "bored" and "afraid"

Humour is naturally subjective; whether something is deemed funny or not depends on the audience’s interpretations, preconceptions, and private taste. Words rarely have a humorous connotation from the definition alone; it's the contexts and structures during which they're used that make or break a joke.

Given the underlying structures of comedy, it should be possible to develop a model capable of recognizing the constructs of humor to a point. To that end, some researchers have tried to apply machine learning to acknowledge the punchlines of jokes. Some limit their scope to specific joke constructs, such as "that’s what she said" jokes or puns. Others have used the transcripts of sitcoms like explosion Theory as training material, counting on the inclusion of canned laugh tracks as positive labels. Whatever the source material, researchers have used Support Vector

Machines (SVM), conditional random fields (CRF), Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN) to recognize humor.

For this task, we choose classifying Yelp reviews as funny or not funny. Yelp reviews are primarily interactions by which users recommend or criticize restaurants to every other, but users also are able to vote reviews as "funny." employing a dataset of 120,000 reviews, we plan to classify humor by way of upvotes. We use RNNs, FFNs, and an LSTMs to capture the varied structures of humor.

Related Work:

There is an outsized body of labor concerning sentiment analysis. While the last word goal of predicting mood or affinity is different from predicting humor, it's still an exercise in classifying text supported word choice and structure. Thus, many of the methods are directly transferable. Of particular interest to us was Pedro M Sosa's work on LSTM-CNN models, which he used for sentiment analysis on consumer tweets just like the author, we hope that the order-preserving LSTM, combined with the feature selection capabilities of CNNs, will help learn structure.

There are a couple of other similar explorations on humor that we reference. Kiddo and Brun, as well as Chen and Soo, use short jokes of fairly standardized structures - "that's what she said" jokes and puns. Bertero and Fung analyze transcripts of massive Bang Theory, complete with laugh tokens where canned laughter audio is overlayed within the show. Chen and Lee used the organic laughter arising in TED talks to capture humor in "Predicting Audience's Laughter Using Convolutional Neural Network."

Of course, we depend also on landmark developments in deep learning and natural language processing, including the invention of the LSTM and developments in word embeddings.

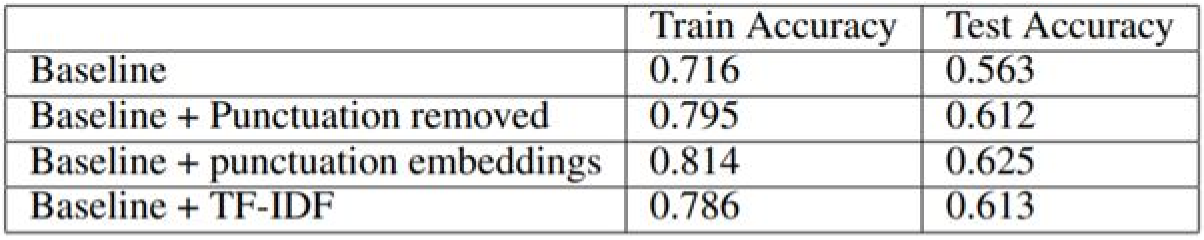
Experiments:

Data: Our data set consists of 120,000 yelp reviews, each coming with a text field for the actual review and a "funny" upvote count that provides the number of people who found the particular review funny. Exactly 60,000 reviews are labeled as "funny" and 60,000 are labeled as "not funny", just to make sure our models don't have too much of a skewed fit to one class. Given that we have only the number of upvotes on whether or not a review is funny, and that our task is binary classification on humor, we labeled the "funny" reviews as reviews that at least one upvote on the "people who found this funny" field, and "not funny" as reviews that had 0 in that field. The dataset itself has a wide variation of reviews, from ones on finer Italian cuisine to ones on simple fast-food chains of certain locations, and a lot of the dialect and speaking differs as well. Many reviews straddle between formal and informal plenty of times, with the more formal reviewers generally being from “pros” as labeled in the dataset and the more informal reviews coming from younger and newer people on the website. There is a survey of all of the types of reviews out there, however.

Evaluation method: Since this is just a binary classification task, for our metrics we used not only accuracy but also precision and recall as well as normalized confusion matrices to properly understand which misclassifications were the weaknesses of the particular algorithm. Because we are dealing with humor, we think it important that inputs marked "humorous" are consistently funny, so we prioritize precision.

Experimental Details and Results: We initially ran our baseline method with 2 epochs through the training data, a stochastic gradient descent optimizer, and a learning rate of 0.001. However, there were some serious underfitting issues with this initial pass, which made us reformulate these initial configurations. We then tried to increase the number of epochs to 8, but the underfitting wasn’t alleviated all that much. We first tried these experiments on a 16,764 point subset of the training data, just to quickly see how turning the knobs on model configuration affects the performance, before running it on our full dataset.

Since this is a Yelp dataset, and reviews can come across informal, the reviews can be riddled with typos. Part of the task for the project was to find an easy way to pre-process this data so as to not introduce too many embeddings due to exact same words being counted twice due to the spelling differences. Examples include people who did not place spaces probably between punctuation (e.g. “food.I” should turn to “food” and “I”). We introduced various pre-processing rules to parse out the obvious duplicates, such as splitting by “.” or “!” in case spaces weren’t provided, or splitting by hyphens into vocabulary that was already encountered.



	Train Accuracy	Test Accuracy
Baseline	0.716	0.563
Baseline + Punctuation removed	0.795	0.612
Baseline + punctuation embeddings	0.814	0.625
Baseline + TF-IDF	0.786	0.613

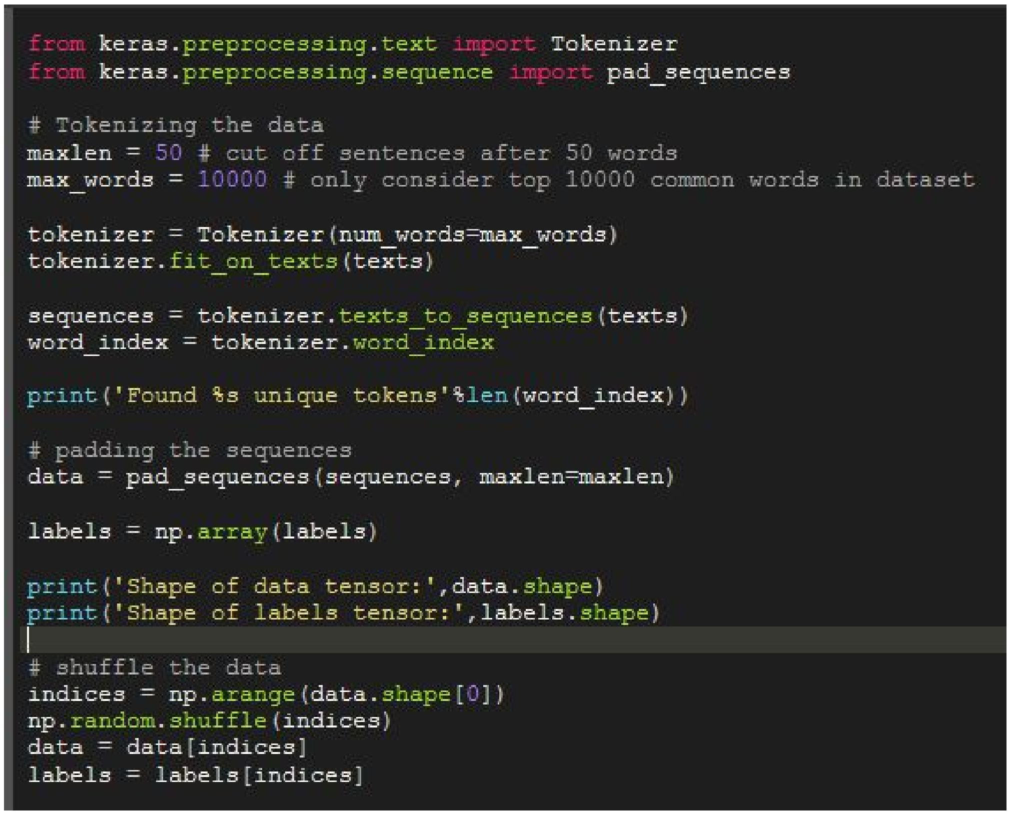
Table-1. Various training and test results on the Baseline model

With this part of the experiment done, we decided to try out the remaining models. We kept batch sizes, learning rates, and optimizer the same to provide some level playing field for each of the models and to see if we can properly distinguish with some set of fixed variables shared across the models. We first ran the FFN model where we used windows of up to 5 words. Next, we ran the RNN model to increase the weightings of some hidden states before doing a weighted sum and feeding into a fully connected layer. Lastly, we ran the LSTM model.

Approaches:

For our approach, we specifically wanted to explore 4 different models: a shallow neural network (baseline), a Feed-Forward Neural Network with a dense layer on the stacked word-embeddings, an RNN on top of the embedding layer, and finally an LSTM model.

Preprocessing the data: As with any textual content analytics problem, we utilize tokenization. Note that we don't tend to remove stopwords or punctuation, as some sequences of characters are quite expressive in the context of Yelp.

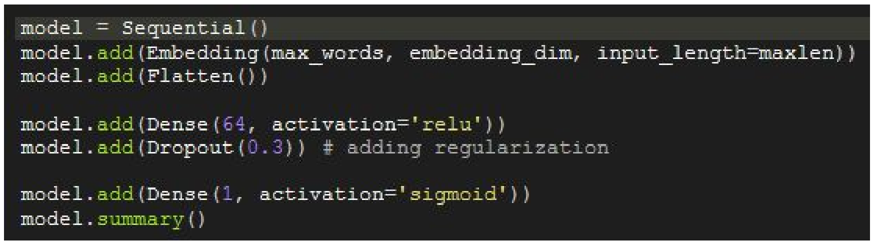


```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

# Tokenizing the data
maxlen = 50 # cut off sentences after 50 words
max_words = 10000 # only consider top 10000 common words in dataset

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)

sequences = tokenizer.texts_to_sequences(texts)
word_index = tokenizer.word_index

print('Found %s unique tokens'%len(word_index))

# padding the sequences
data = pad_sequences(sequences, maxlen=maxlen)

labels = np.array(labels)

print('Shape of data tensor:',data.shape)
print('Shape of labels tensor:',labels.shape)

# shuffle the data
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
```

Figure-1. Preprocessing of the dataset

Baseline: Our baseline is the basic shallow neural network. This network takes in a 256-dimensional input, which is the desired dimensionality of our word embeddings. To reduce a review of multiple word vectors to one “encoded” vector, we simply sum all of the word vectors we retrieve from the embedding layer. We expect this loss of order information to be insufficient for classification of humor, and that we primarily would really like to ascertain what proportion order-information helps the opposite models.

FFN: A FNN is a neural network where the connections between the nodes don't end in a cycle. This makes it different from its successor, RNNs. Our model will retrieve a matrix of word embeddings representing a review's original text before proceeding to perform predictions on this matrix.



```
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Flatten())

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3)) # adding regularization

model.add(Dense(1, activation='sigmoid'))
model.summary()
```

Figure-2. Structure of Feed Forward Neural Network

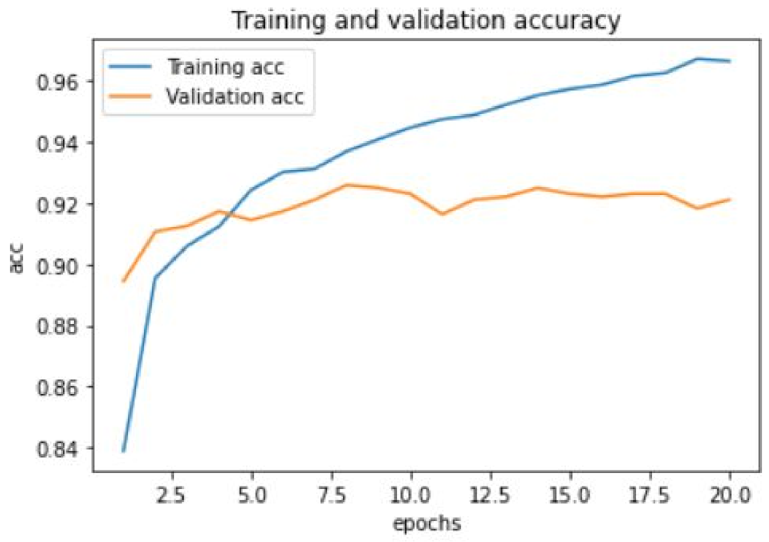


Figure-3. Accuracy on training and validation set (FNN)

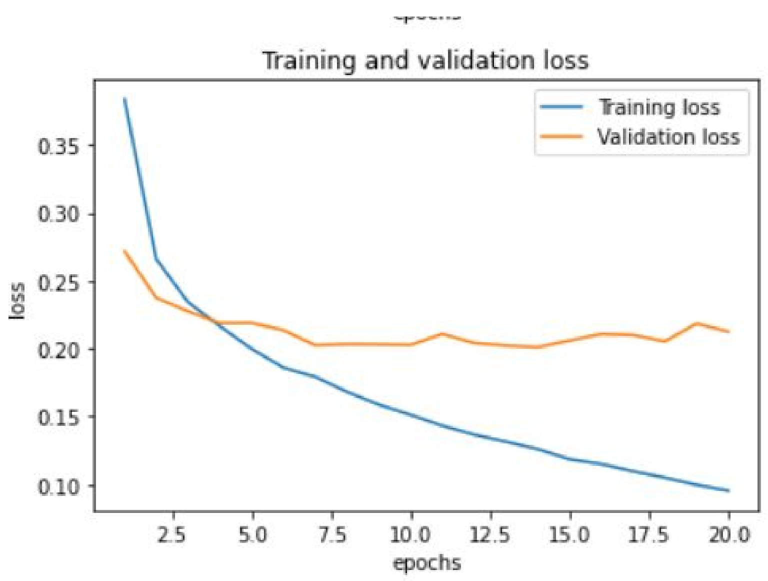
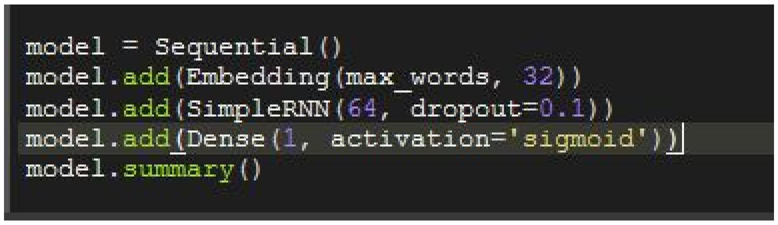


Figure-4. Accuracy on training and validation set (FNN)

RNN model on top of the Embedding layer: As a next step, we decided to use models that try to model sequences of word vectors. We can consider the reviews as a sequence where we only have one label at the end. This leads us to Recurrent Neural Networks. In general, jokes build to a punchline and are built around a specific subject for which they use a handful of keywords.



```
model = Sequential()
model.add(Embedding(max_words, 32))
model.add(SimpleRNN(64, dropout=0.1))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

Figure-5. Structure of Simple RNN

We're hoping that this model outperforms the previous two models based on the explicit retention of order-information. We feed the review into an embedding layer and pass each of the embeddings as inputs to the RNN.

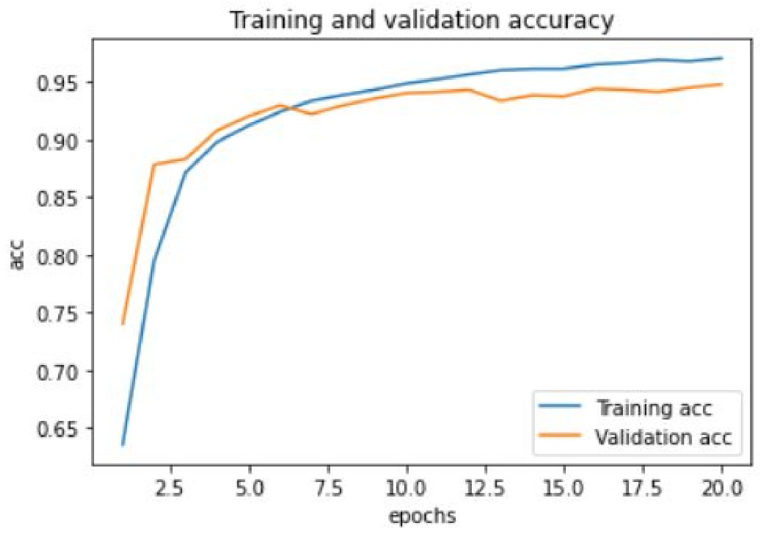


Figure-6. Accuracy on training and validation set (RNN)

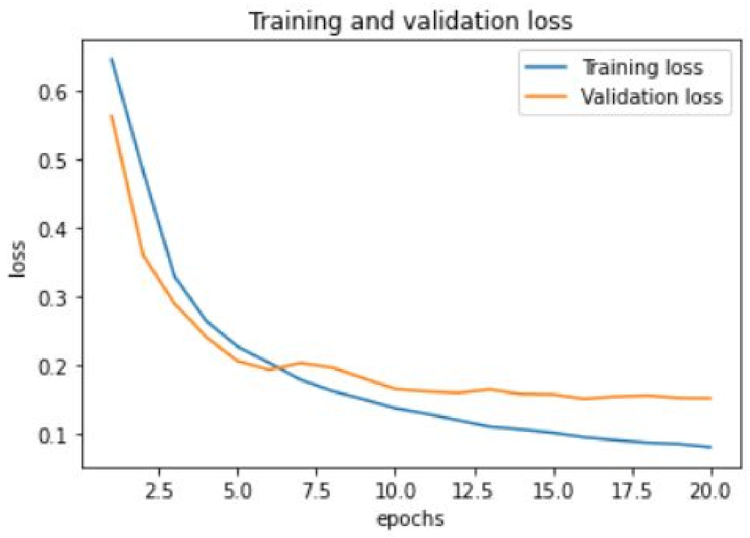


Figure-7. Accuracy on training and validation set (RNN)

LSTM (Long Short Term Memory):

- We thought of exploiting the idea of keeping memory units to try and capture long-distance dependencies in the text, and so we implemented LSTMs.Long Short Term Memory networks – usually just called "LSTMs" – are a type of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997) and were refined and popularized by many of us in the following work. They work tremendously well on an outsized kind of problem and are now widely used. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of your time is practically their default behavior, not something they struggle to learn! All recurrent neural networks have the shape of a sequence of repeating modules of neural networks. In standard RNNs, this repeating module will have an awfully simple structure, like one tanh layer.

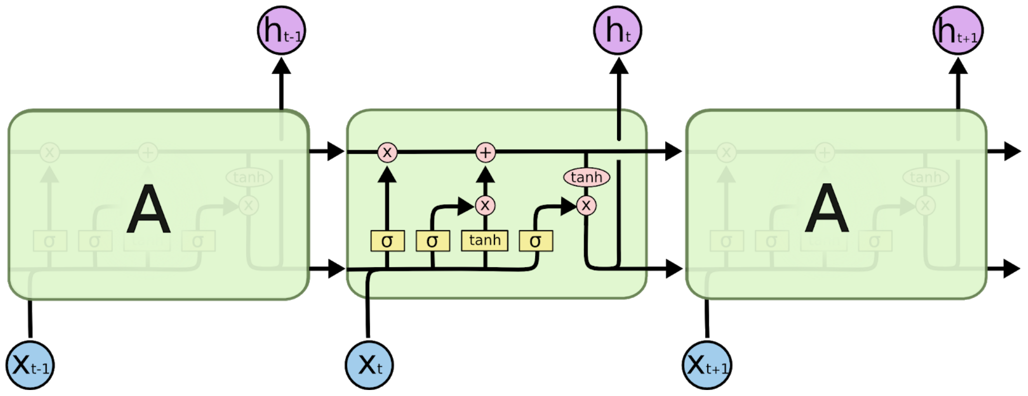
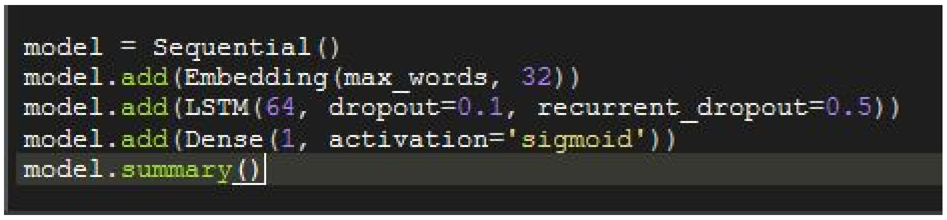


Figure-8. Basic working of LSTM

Here, We use LSTM such that it compares with the other two approaches described above. It is very simple to implement LSTM in every dataset just we have to add the LSTM layer between the Sequential and Dense layer just like shown below:



```
model = Sequential()
model.add(Embedding(max_words, 32))
model.add(LSTM(64, dropout=0.1, recurrent_dropout=0.5))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

Figure-9. Structure of LSTM

From this approach, we get a good amount of accuracy on the training and validation dataset and the minimum data loss.

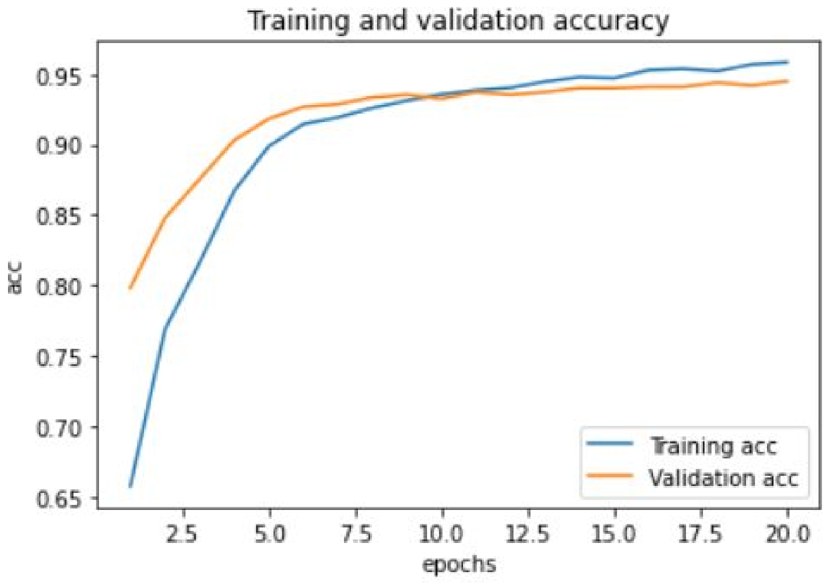


Figure-10. Accuracy on training and validation set (LSTM)

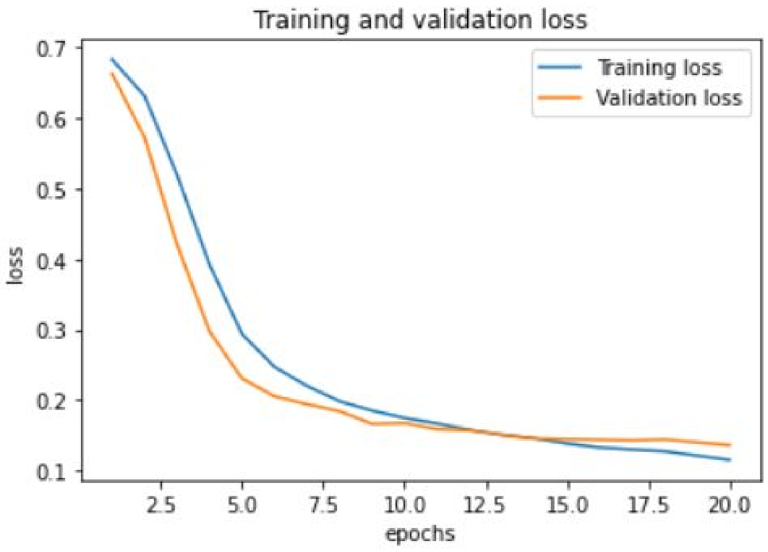


Figure-11. Loss on training and validation set (LSTM)

Conclusion and Future Work:

Rank	Name	Training Accuracy	Validation Accuracy
1	Feed Forward Network	0.9643	0.9284
2	Simple RNN	0.9518	0.9415
3	LSTM layer	0.9589	0.9480

Table-2. Compares the accuracy on all of the used models

As seen in the table, the Feed Forward network gave the best results in training while the LSTM model gave the most valid outputs. The RNN model gave mediocre result.

In the future, we will try to improve the accuracy of the model using the concept of Bidirectional LSTM and Google's Bidirectional Encoder Representations from Transformer (BERT) and ELMo pre-trained word embeddings.