## **Summary**

Our primary goal of this project is to predict the sentiment accurately from reviews using Deep Learning. We have used the Recurrent Neural Network for building and training our custom model. We have collected our dataset from a kaggle competition which contains 50000 Labeled data and 50000 Unlabeled data. At first we have inspected the dataset and removed the noise from the data. Then we have preprocessed dataset using embedding method and tried to train the model. We have stopped the training if the validation loss does not decrease after 3 epochs. Here we have chosen to save only the best possible iteration of model. Finally We have compared the results of the best three models, based on the validation data.Then we average the predictions of these three models, which produce 93.5% accuracy.

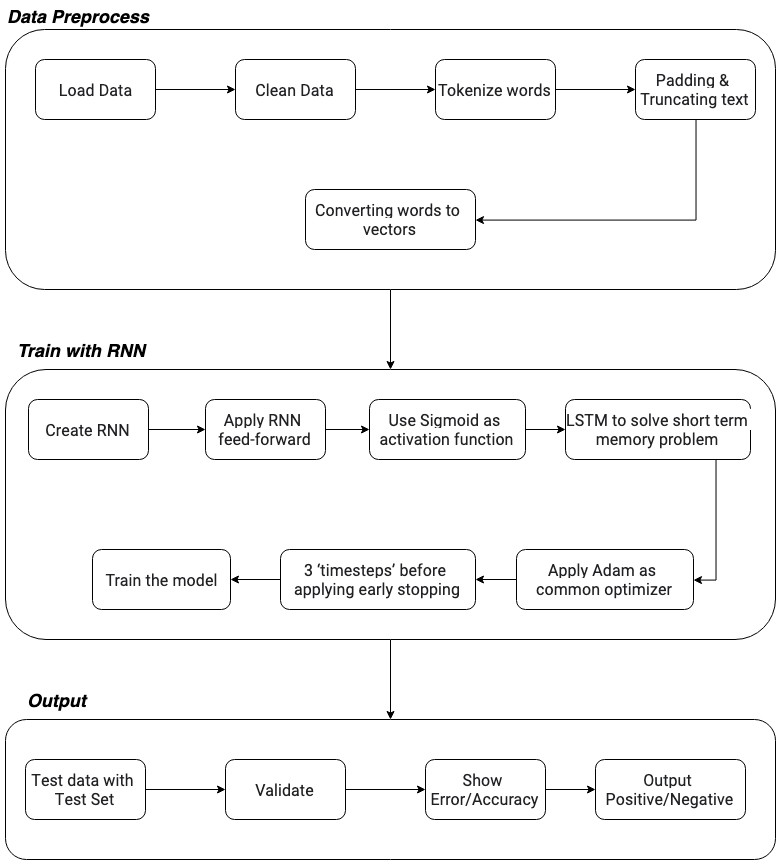
## **Introduction**

Humans don’t start their thinking from scratch every second. As we read any article, we understand each word based on our understanding of previous words. We don’t throw everything away and start thinking from scratch again. Our thoughts have persistence. Traditional methods and the traditional neural network can’t remember the previous state findings. But recurrent neural network (RNN) can solve this kind of problem because RNN takes sequential text as input (a single-line sentence or multiple sentences) and analyze the whole sentence or multiple sentences together as we humans do.

Movie review talks about the viewpoint of the story of the movie, the characters, their particular roles, backgrounds, scripts and so on. As a result, it gives a definite picture to the viewers. Our primary goal of this project to find out whether a review conveys a positive or negative sentiment. For example, we say a review contains negative feedback if it has a racist or sexist sentiment associated with it.

Here we have tried to classify positive vs negative sentiment from IMDB movie reviews. This type of solving helps to identify the user feedback statistics that how much acceptance or popularity of a movie has just having a look into millions of user feedback which is impossible to analyze manually.

## **Block diagram of the Project**



***Figure 1: Block diagram of the workflow of the Sentiment Analysis process***

## 

## **Description of the Dataset**

We have collected data set for our project from a Kaggle competition: [**https://www.kaggle.com/c/word2vec-nlp-tutorial**](https://www.kaggle.com/c/word2vec-nlp-tutorial)

Total Number of instances in Data set: **50,000** IMDB movie reviews (**Labeled**)

Total Number of instances in Data set: **50,000** IMDB movie reviews (**Unlabeled**)

The sentiment of reviews is **binary** (**0 for negative review and 1 for positive review**)

**Preprocessing**

To improve the performance of our model we have cleaned the data set by removing unnecessary HTML tags, Special characters, and Extra spaces. We also removed stop words from the review because Stop words are words that provide a little context to a sentence (a, the, just…) and produce unnecessary noise in the data set. We have used NLTK Corpora to get the list of stop words for English. Here’s a link to the list of stop words that we are using **https://www.nltk.org/nltk\_data/**

## **Input procession / Feature Embedding**

We have used words tokenizing for input processing. Tokenizing is the process of converting words into numbers. Each word in the sentence gets a unique number after tokenizing. We have used tokenizing for input processing because Deep Learning algorithm works better with numbers instead of text. We have used Keras library for tokenizing words.

To Train our model faster we are limiting the length of review up to a maximum of 200. Reviews with more than 200 words will have those extra words removed. Reviews with less than 200 words will have padding tokens added until it reaches a length of 200. We have used Keras sequence library for padding.

We have used a TensorFlow random\_uniform distribution for word embeddings before feeding into our model. Because Embedding is a technique which converts words to vectors with a specified embedding size of dimensions.

## 

## **Neural Network Architecture**

We have used LSTM (Recurrent Neural Network) for building our model.

The main building blocks of our RNN Model are:

1. **RNN Layers**: This is the heart of our recurrent model. By default it is 2 layer network with 50% dropout by default.
2. **RNN Feed Forward**: This is the feed forward part of our model. we could use different maximum review length for each batch.
3. **Fully Connected Layer**: This is where we add our first, and possibly second fully connected layers. Their weights and biases are initialized using tensorflow initializer.
4. **Output Layer**: We only have 1 output because we are predicting the sentiment on a scale of 0 to 1. Sigmoid is our activation function because it maps the output of our final fully connected layer to this range.

The default parameters for our model:

n\_words = len(word\_index)

embed\_size = 300

batch\_size = 250

lstm\_size = 128

num\_layers = 2

dropout = 0.5

learning\_rate = 0.001

epochs = 100

multiple\_fc = False

fc\_units = 256

The training process of model starts with the default parameters if it is not specified.

## **Results and Discussion**

The default parameters of our model are the great thing to improve the performance of the model. Because we can change the default values for tuning the performance of our model. We have set default batch\_size to 250 so that data will be divided into equal batches. Here we have tried to stop the training based on early stopping rather than the number of epochs. Because early stopping can save a lot of time when training.Here we have chosen to save only the best iteration of model. We have stopped the training if the validation loss does not decrease after 3 epochs. If the default number of epoch is set to 100, then we can say that our model is fully trained. But in that case training process would be more complex and time consuming. So in order to save training time we have used 13 epochs for training our model.

After training our model we have to test our prediction for testing data. If we do not change the default parameters, the prediction will be made with the default parameters, which are likely to be less optimal than some of the values with which we tuned.

We have compared the results of the best three models, based on the validation data. Then we average the predictions of these three models, which should produce an even better set of predictions.

The results of the predictions that we got after using the default settings are as follows :

* Predictions1: 0.919
* Predictions2: 0.914
* Predictions3: 0.916
* Combined Predictions: 0.935

That means our model can predict the sentiment of a review 93.5% accurately.

## **Conclusion & Future Work**

Here we have tried to analyze sentiment from user comments or feedback. Although we have tried and have been successful to solve this problem only with 2 labels (Positive vs Negative), and we are also pleased by how this analysis has finished, but not satisfied with our work because of the feeling of incompleteness. There are some limitations that we want to fulfill in the near future. This result could have been improved by using a larger model, using pre-trained vectors (such as GloVe), and using an ensemble of more predictions. Although it would be nice to carry out these efforts and improve our results, we feel that would not be the best use of our time. Rather we can work on supporting multiple types of sentiment analysis, we should not be stuck only with these two labels.

## **Data and Code Availability**

All of our Python code, Colab file, and Dataset files are uploaded at Github repository, which can be found here- <https://github.com/tanvirehsan/sentiment-analysis-tensorflow>