Sentiment Analysis Using Simple RNN, LSTM, GRU, Bidirectional LSTM

***Abstract***— In this deep learning era, NLP is a thriving field that see lots of different advancements. One of them is sentiment analysis. In this paper, I have applied a simple and efficient Neural Language Model approach for text classification (Sentiment Analysis) that relies only on unsupervised word representation inputs. I have applied four models for sentiment analysis: -

1. **Model 1** that employs Embedding layer followed by a simple RNN layer followed by a fully connected layer.
2. **Model 2** that employs Embedding layer followed by an LSTM layer followed by a fully connected layer.
3. **Model 3** that employs Embedding layer followed by a GRU layer followed by a fully connected layer.
4. **Model 4** that employs Embedding layer followed by a bidirectional LSTM layer followed by a fully connected layer.

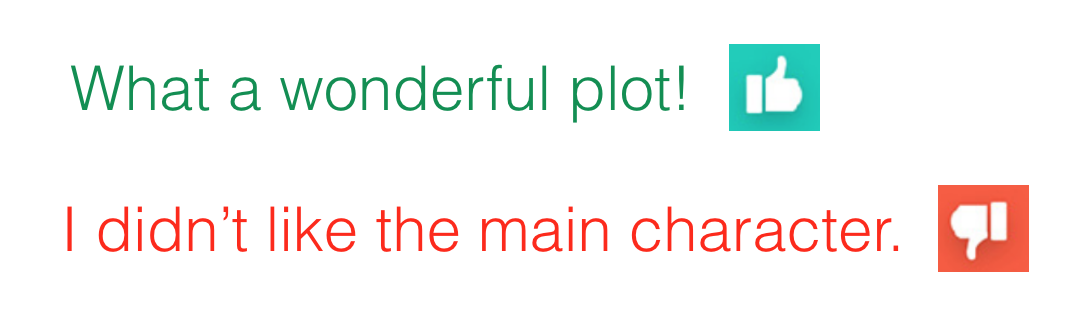
I am also applying word embedding technique, which has improved the model to capture syntactic and semantic word relationships. This project shows that applying the above three models achieves excellent result on the customer review dataset.

1. **INTRODUCTION**

Natural language processing is all about creating systems that process or “understand” language in order to perform certain tasks. These tasks could include:

* Question Answering – The main job of technologies like Siri, Alexa, and Cortana
* Sentiment Analysis – Determining the emotional tone behind a piece of text
* Image to Text Mappings – Generating a caption for an input image
* Machine Translation – Translating a paragraph of text to another language
* Speech Recognition – Having computers recognize spoken words

Sentiment Analysis: The process of computationally identifying and categorizing opinions expressed in a piece of text, especially in order to determine whether the writer's attitude towards a particular topic, product, etc. is positive, negative, or neutral.



When it comes to sentiment detection it has become a bit of a commodity. Especially the big 5 vendors offer their own sentiment detection as a service. Google offers an NLP API with sentiment detection. Microsoft offers sentiment detection through their Azure platform. IBM has come up with a solution called Tone Analyzer, that tries to get the "tone" of the message, which goes a bit beyond sentiment detection. Amazon offers a solution called comprehend that runs on AWS as a lambda.

1. **METHODOLOGY AND APPROACH**

My methodology for semantic analysis is as follow: -

* 1. **Getting Data**

I have used the IMDB movie review dataset for training my model. This dataset includes 25,000 movies reviews from IMDB, labelled by sentiment (positive/negative). Reviews have been pre-processed, and each review is encoded as a sequence of word indexes (integers). For convenience, words are indexed by overall frequency in the dataset, so that for instance the integer "3" encodes the 3rd most frequent word in the data. This allows for quick filtering operations such as: "only consider the top 10,000 most common words, but eliminate the top 20 most common words". As a convention, "0" does not stand for a specific word, but instead is used to encode any unknown word. [Link to Dataset](https://keras.io/datasets/#imdb-movie-reviews-sentiment-classification)

Syntax:

*from keras.datasets import imdb*

*(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data()*

We can also check for the dataset instances using *keras.datasets.imdb.get\_word\_index()* routine.

* 1. **Data Pre-processing**

The RNN will take sequences of constant length. This length is the *words\_limit* which is defined to be 100 in my code. Since the reviews differ heavily in terms of lengths, I will trim each review to its first 100 words. If reviews are shorter than 100 words, I will pad them with zeros. That is, if the review is ['best', 'movie', 'ever'], [117, 18, 128] as integers, the row will look like [0, 0, 0, ..., 0, 117, 18, 128]. These word encodings will be passed on to the RNN model as inputs.

* 1. **RNN Model**

The model architecture looks like:

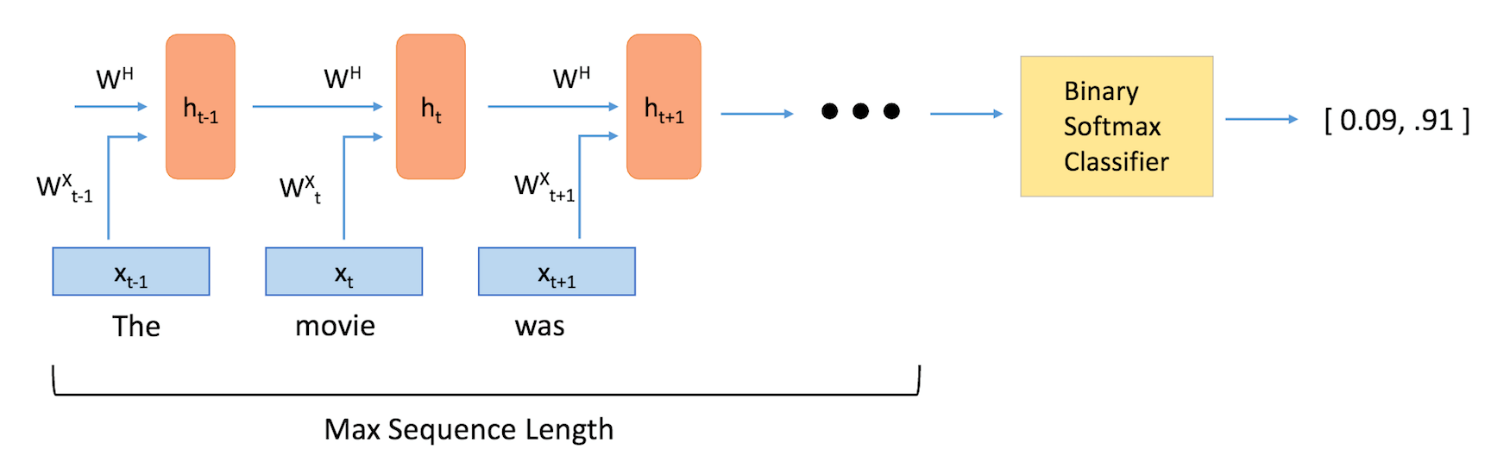


Figure 1 Many to One RNN Model Architecture

Where **ht**: RNN layer at time t.

**Xt**: Word embeddings which acts as input for RNN layer at time t.

**W:** Represent weights.

My RNN model includes the following layers:

* + 1. **Embedding layer**

The embedding layer will learn a word embedding for all the words in the dataset. It has three arguments. The input dimension in our case is 200 words. The output dimension aka the vector space in which words will be embedded. In our case we have chosen 128 dimensions so a vector of the length of 128 to hold our word coordinates. We want these vectors to be created in such a way that they somehow represent the word and its context, meaning, and semantics. For example, we’d like the vectors for the words “love” and “adore” to reside in relatively the same area in the vector space since they both have similar definitions and are both used in similar contexts. The vector representation of a word is also known as a word embedding.

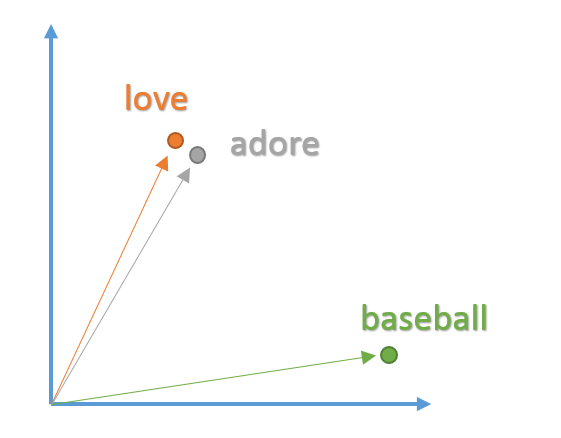
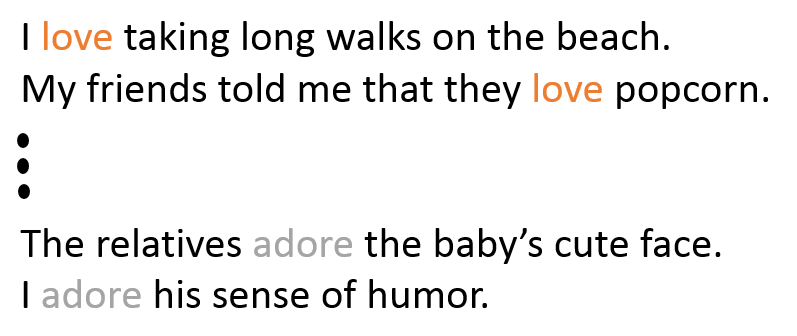


Figure 2 Word Embedding Vectors

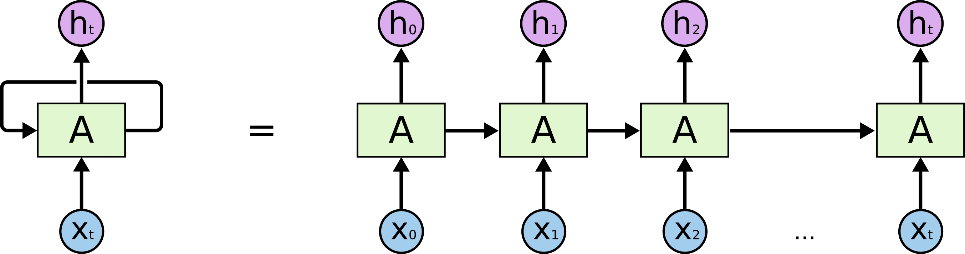
Words with similar contexts will be placed close together in the vector space. In natural language, the context of words can be very important when trying to determine their meanings. Taking our previous example of the words “adore” and “love”, consider the types of sentences we’d find these words in.



From the context of the sentences, we can see that both words are generally used in sentences with positive connotations and generally precede nouns or noun phrases. This is an indication that both words have something in common and can possibly be synonyms. Context is also very important when considering grammatical structure in sentences. Most sentences will follow traditional paradigms of having verbs follow nouns, adjectives precede nouns, and so on. For this reason, the model is more likely to position nouns in the same general area as other nouns. The layer takes in a large dataset of sentences (English Wikipedia for example) and outputs vectors for each unique word in the corpus.

* + 1. **Simple RNN layer**

The unique aspect of NLP data is that there is a temporal aspect to it. Each word in a sentence depends greatly on what came before and comes after it. In order to account for this dependency, we use a **many-to-one recurrent neural network**.



In the above diagram, a chunk of neural network, A, looks at some input xt and outputs a value ht. A loop allows information to be passed from one step of the network to the next. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.

**The state space model: -**

****

* Suppose inputs and outputs are the vectors 𝒙(𝒕)and 𝒚(𝒕)
* the three connection weight matrices are **WIH**, **WHH** and **WHO**, and
* the hidden and output unit activation functions are **fH** and **fO**, then the behaviour of the recurrent network can by the pair of non-linear matrix equations:

Here, the state is defined by the set of hidden unit activations h(t).

In my model the output is a binary variable with outputs 0 (Negative) and 1(Positive).

* + 1. **Fully Connected layer with dropouts**

Fully Connected layers are also known as dense layer. Here each hidden layer neuron is connected to every output neurons.

Dropout refers to dropping out units (both hidden and visible) in a neural network. Dropout is a form of regularization. It aims to help prevent overfitting by increasing testing accuracy, for each mini-batch in the training set, dropout layers, with probability p, randomly disconnect inputs from the preceding layer to the next layer in the network architecture. Here, I randomly disconnect with probability p=0.4. After the forward and backward pass are computed for the minibatch, we re-connect the dropped connections, and then sample another set of connections to drop.



* + 1. **Activation layer**

Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron. I have used the **softmax** activation function in my project.

The Softmax regression is a form of logistic regression that normalizes an input value into a vector of values that follows a probability distribution whose total sums up to 1. The output values are between the range [0,1] which is nice because we are able to avoid binary classification and accommodate as many classes or dimensions in our neural network model. This is why softmax is sometimes referred to as a multinomial logistic regression. The function is usually used to compute losses that can be expected when training a data set. Known use-cases of softmax regression are in discriminative models such as Cross-Entropy and Noise Contrastive Estimation. These are only two among various techniques that attempt to optimize the current training set to increase the likelihood of predicting the correct word or sentence.

1. **TRAINING MODEL**

For the training of the model I have considered the batch size to be 32 and number of epochs to be 3 and used the following optimizer and loss function: -

* 1. **Adam Optimizer**

Adam, is by far the most popular and widely used optimizer in DL. In most cases, you can blindly choose the Adam optimizer and forget about the optimization alternatives. Adam is different to classical stochastic gradient descent. Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training.

However, in ADAM, a learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds. The math representation can be simplified in the following way:

* 1. **Binary Cross Entropy loss**

It is the loss function used in training the deep learning models where the categorical outcome of the model is a binary variable (Say, Yes/No). The algorithm will try to minimize this loss by adjusting models’ parameters/weights. The binary cross entropy loss will be equated as: -

Where y = 1/0 (Positive/Negative), p : probability of positive outcome.

1. **RESULT**

**Current Model 1** have achieved an accuracy of 65.98% and has a loss score of 61.33%.

The loss and acuuracy graph:

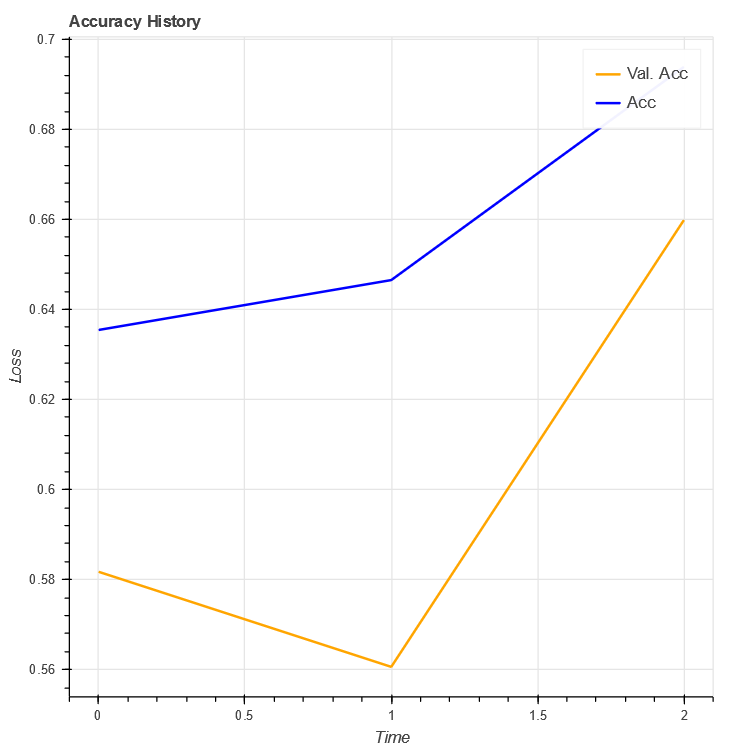
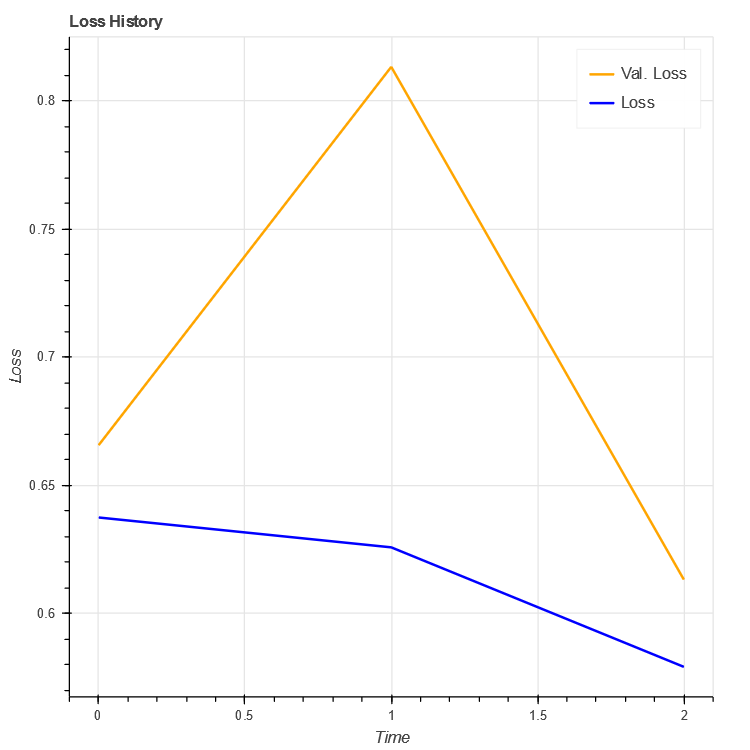


Figure 3 Loss Curve Figure 4 Accuracy Curve

1. **CONCLUSION:**

RNN and its kind are a great way to do sentiment analysis with minimum amount of workflow. The validation accuracy is going up and the model accuracy is going up. So, we can conclude that model is performing well.

1. **REFERENCES:**

[1] A. Hassan, “SENTIMENT ANALYSIS WITH RECURRENT NEURAL NETWORK AND UNSUPERVISED Ph . D . Candidate : Abdalraouf Hassan , Advisor : Ausif Mahmood Dep of Computer Science and Engineering , University of Bridgeport , CT , 06604 , USA,” no. March, pp. 2–4, 2017.