Journal of Theoretical and Applied Information Technology
15th June 2023. Vol.101. No 11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

EMPIRICAL INVESTIGATIONS TO SENTIMENT ANALYSIS OF MOVIE REVIEWS USING LSTM

[1]DR. HABIBULLA MOHAMMAD, [2]DR. J. RAVINDRA BABU, [3]DR. SURYA PRASADA RAO BORRA. [4]DR.SRINU PYLA [5]MR.TATA BALAJI [6]MRS. B. MOUNIKA [7]KOTESWARA RAO KODEPOGU

[1]Sr.Assistant Professor, Dept of ECE, PVP Siddhartha Institute of Technology,
[2]Associate Professor, Dept of ECE, PVP Siddhartha Institute of Technology:
[3]Associate Professor, Dept of ECE, PVP Siddhartha Institute of Technology:
[4]Dept. of ECE, Gayatri Vidya Parishad College of Engineering Visakhapatnam,
[5]Asst. Professor, Dept of ECE, PVP Siddhartha Institute of Technology:
[6]CSE Department, SRKR Engineering College, Bhimavaram, Andhra Pradesh,
[7]Associate Professor, Dept of CSE, PVP Siddhartha Institute of Technology:

E-mail: honeyhabeeb@gmail.com, jrb0009@gmail.com, suryaborra1679@gmail.com, srinupyla@gvpce.ac.in, balu170882@gmail.com, bmounika@srkrec.ac.in, koteswara2003@yahoo.co.in

ABSTRACT

Sentiment analysis or opinion mining is the computational study of people's opinions, sentiments, attitudes and emotions expressed in written language. It is one of the most active research areas in Natural Language Processing in the recent years. Sentiment analysis aids corporations in making decisions and changes in their business or service models based on the feedback of the customers regarding the current models.

Most sentiment analysis problems are classification problems (positive/neutral/negative) and not regression problems. It comes under Sequential problems which are a class of problem in machine learning where the order of the features presented to the model is important for making predictions.

In this project, we study the existing classification model based on Recurrent Neural Network (RRN), build a machine learning (Classification) model using long short-term memory **(**LSTM**)** network to overcome the Vanishing Gradient problem faced in RRN. The model takes an IMDB movie review dataset with 50,000 reviews as input; trains on 25,000 and uses the experienced acquired so far to classify another 25,000 reviews into positive and negative categories. We animate the results of the model using graphs.
Keywords: *LSTM, RNN, Sequential Problems, Sentiment, Movie Reviews*

1. INTRODUCTION

Sentiment analysis is an area of Natural Language Processing that benefited from the resurgence of deep learning. Sentiment analysis is defined as the prediction of the positivity of a text. Most sentiment analysis problems are classification problems (positive/neutral/negative) and not regression problems. It comes under Sequential problems which are a class of problem in machine learning where the order of the features presented to the model is important for making predictions.[1]

There are many practical applications of sentiment analysis. For example, modern customer service centers use sentiment analysis to predict the satisfaction of customers through the reviews they provide on platforms such as Yelp or Facebook. This allows businesses to step in immediately whenever customers are dissatisfied, allowing the problem to be addressed as soon as possible, and preventing customer churn.

Sentiment analysis has also been applied in the domain of stocks trading. In 2010, scientists showed that by sampling the sentiment in Twitter (positive versus negative tweets), we can predict whether the stock market will rise. Similarly, high-frequency trading firms use sentiment analysis to sample the sentiment of news related to certain companies, and execute trades automatically, based on the positivity of the news. The problem is sentiment analysis i.e. classifying the polarity of the movie reviews, i.e. whether the expressed opinion in the review or an entity feature/aspect is positive or negative.

1.1Sequential problems: These are a class of problem in machine learning in which the order of the features presented to the model is important for making predictions. Sequential problems are commonly encountered in the following scenarios:

NLP, including sentiment analysis, language translation, and text prediction Time series predictions

For example, let's consider the text prediction problem, as shown in the following screenshot, which falls under NLP:[2]

Human beings have an innate ability for this, and it is trivial for us to know that the word in the blank is probably the word *Japanese*. The reason for this is that as we read the sentence, we process the words as a sequence. The sequence of the words captures the information required to make the prediction. By contrast, if we discard the sequential information and only consider the words individually, we get a *bag of words,* as shown in the following diagram. We can see that our ability to predict the word in the blank is now severely impacted. Without knowing the sequence of words, it is impossible to predict the word in the blank. Besides text predictions, sentiment analysis and language translation are also sequential problems. In fact, many NLP problems are sequential problems, because the languages that we speak are sequential in nature, and the sequence conveys context and other subtle nuances. Sequential problems also occur naturally in time series problems. Time series problems are common in stock markets. The stock prediction problem is accurately defined as a time series problem, because knowing the movement of the stocks in the preceding hours or minutes is often crucial to predicting whether the stock will rise or fall. Today, machine learning methods are being heavily applied in this domain, with algorithmic trading strategies driving the buying and selling of stocks.

1.2NLP and sentiment analysis:

NLP is a subfield in **artificial intelligence** (**AI**) that is concerned with the interaction of computers and human languages. As early as the 1950s, scientists were interested in designing intelligent machines that could understand human languages. Early efforts to create a language translator focused on the rule-based approach, where a group of linguistic experts handcrafted a set of rules to be encoded in machines. However, this rule-based approach produced results that were sub-optimal, and, often, it was impossible to convert these rules from one language to another, which meant that scaling up was difficult. For many decades, not much progress was made in NLP, and human language was a goal that AI couldn't reach—until the resurgence of deep learning.

With the proliferation of deep learning and neural networks in the image classification domain, scientists began to wonder whether the powers of neural networks could be applied to NLP. In the late '00s, tech giants, including Apple, Amazon, and Google, applied LSTM networks to NLP problems, and the results were astonishing. The ability of AI assistants, such as Siri and Alexa, to understand multiple languages spoken in different accents was the result of deep learning and LSTM networks. In recent years, we have also seen a massive improvement in the abilities of text translation software, such as Google Translate, which is capable of producing translations as good as human language experts.

1.3Sentiment analysis is also an area of NLP that benefited from the resurgence of deep learning. Sentiment analysis is defined as the prediction of the positivity of a text. Most sentiment analysis problems are classification problems (positive/neutral/negative) and not regression problems. There are many practical applications of sentiment analysis. For example, modern customer service centers use sentiment analysis to predict the satisfaction of customers through the reviews they provide on platforms such as Yelp or Facebook. This allows businesses to step in immediately whenever customers are dissatisfied, allowing the problem to be addressed as soon as possible, and preventing customer churn. Sentiment analysis has also been applied in the domain of stocks trading. High-frequency trading firms use sentiment analysis to sample the sentiment of news related to certain companies, and execute trades automatically, based on the positivity of the news.[3]

Journal of Theoretical and Applied Information Technology
15th June 2023. Vol.101. No 11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

1.4Why sentiment analysis is difficult:

Early efforts in sentiment analysis faced many hurdles, due to the presence of subtle nuances in human languages. The same word can often covey a different meaning, depending on the context. Take for example the following two sentences:

We know that the sentiment of the first sentence is negative, as it probably means that the building is literally on fire. On the other hand, we know that the sentiment of the second sentence is positive, since it is unlikely that the person is literally on fire. Instead, it probably means that the person is on a *hot streak*, and this is positive. The rule-based approach toward sentiment analysis suffers because of these subtle nuances, and it is incredibly complex to encode this knowledge in a rule-based manner. Another reason sentiment analysis is difficult is because of sarcasm. Sarcasm is commonly used in many cultures, especially in an online medium. Sarcasm is difficult for computers to understand. In fact, even humans fail to detect sarcasm at times. Take for example the following sentence:

We can probably detect sarcasm in the preceding sentence, and come to the conclusion that the sentiment is negative. However, it is not easy for a program to understand that.

2. Existing System-RNN: To work with sequential data, the neural network needs to take in specific bits of the data at each time step, in the sequence that it appears. This provides the idea for an RNN. An RNN has high-level architecture, as shown in the following diagram:

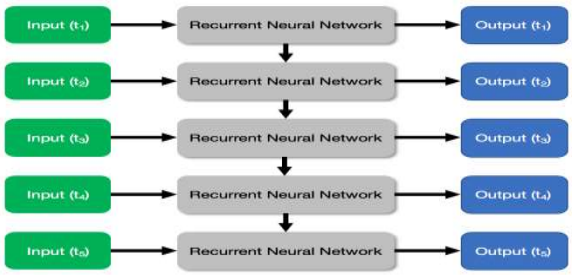
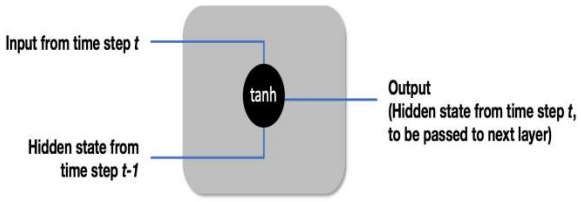


Figure1. RNN Architecture

From the previous diagram, we can see that an RNN is a multi-layered neural network. We can break up the raw input, splitting it into time steps. For example, if the raw input is a sentence, we can break up the sentence into individual words (in this case, every word represents a time step). Each word will then be provided in the corresponding layer in the RNN as **Input**. More importantly, each layer in an RNN passes its output to the next layer. The intermediate output passed from layer to layer is known as the hidden state. Essentially, the hidden state allows an RNN to maintain a memory of the intermediate states from the sequential data.

2.1Inside RNN:

The following diagram depicts the mathematical function inside each layer of an RNN:



The mathematical function of an RNN is simple. Each layer *t* within an RNN has two inputs:

- The input from the time step *t*
- The hidden state passed from the previous layer *t-1*

Each layer in an RNN simply sums up the two inputs and applies a *tanh* function to the sum. It then outputs the result, to be passed as a hidden state to the next layer. More formally, the output hidden state of layer *t* is this:

$$s_t = tanh(s_{t-1} + x_t)$$

Journal of Theoretical and Applied Information Technology
15th June 2023. Vol.101. No 11

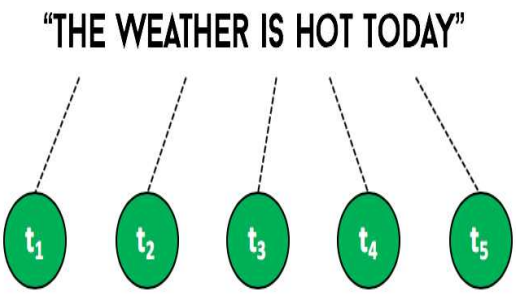
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

In the previous equation, *n* is the index of the last layer in the RNN. Recall from previous chapters that the *sigmoid* function produces an output between 0 and 1, hence providing the probabilities for each class as a prediction. We can see that if we stack these layers together, the final output from an RNN depends on the non-linear combination of the Understanding the approach:[4]

The architecture of an RNN makes it ideal for handling sequential data. Let's take a look at some concrete examples, to understand how an RNN handles different lengths of sequential data. Let's first take a look at a short piece of text as our sequential data:

"THE WEATHER IS HOT TODAY"

We can treat this short sentence as sequential data by breaking it down into five different inputs, with each word at each time step. This is illustrated in the following diagram:



Now, suppose that we are building a simple RNN to predict whether it snowing is based on this sequential data. The RNN would look something as follows:

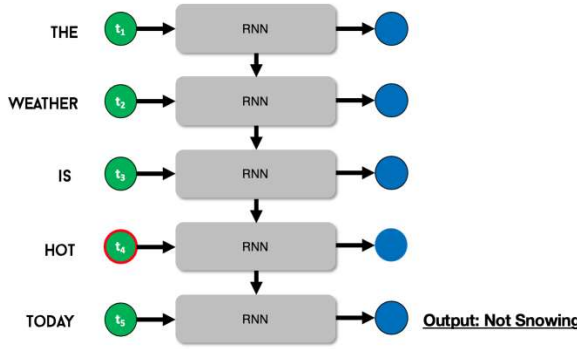


Figure2.RNN look

The critical piece of information in the sequence is the word **HOT**, at time step 4 (t_4, circled in red). With this piece of information, the RNN is able to easily predict that it is not snowing today. Notice that the critical piece of information came just shortly before the final output. In other words, we would say that there is a short-term dependency in this sequence.

2.2Drawbacks:

Clearly, RNNs have no problems with short-term dependencies. But what about long-term dependencies? Let's take a look now at a longer sequence of text. Let's use the following paragraph as an example:

"I really liked the movie but I was disappointed in the service and cleanliness of the cinema. The cinema should be better maintained in order to provide a better experience for customers."

Our goal is to predict whether the customer liked the movie. Clearly, the customer liked the movie but not the cinema, which was the main complaint in the paragraph. Let's break up the paragraph into a sequence of inputs, with each word at each time step (32 time steps for 32 words in the paragraph). The RNN would look this:

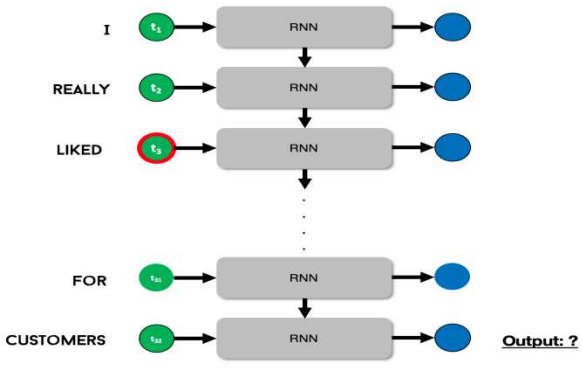


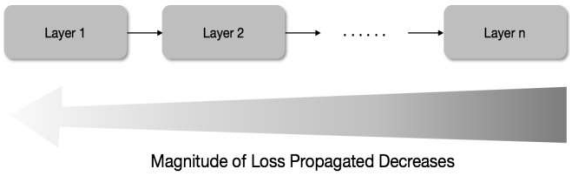
Figure3.RNN look

The critical words **liked the movie** appeared between time steps 3 and 5. Notice that there is a significant gap between the critical time steps and the output time step, as the rest of the text was largely irrelevant to the prediction problem (whether the customer liked the movie). In other words, we say that there is a long-term dependency in this sequence. Unfortunately, RNNs do not work

well with long-term dependency sequences. RNNs have a good short-term memory, but a bad long-term memory. To understand why this is so, we need to understand the **vanishing gradient problem** when training neural networks.[5]

2.3Vanishing gradient problem:

The vanishing gradient problem is a problem when training deep neural networks using gradient-based methods such as back propagation. Recall in previous chapters, we discussed the back propagation algorithm in training neural networks. In particular, the loss function provides information on the accuracy of our predictions, and allows us to adjust the weights in each layer, to reduce the loss. So far, we have assumed that back propagation works perfectly. Unfortunately, that is not true. When the loss is propagated backward, the loss tends to decrease with each successive layer:



As a result, by the time the loss is propagated back toward the first few layers, the loss has already diminished so much that the weights do not change much at all. With such a small loss being propagated backward, it is impossible to adjust and train the weights of the first few layers. This phenomenon is known as the vanishing gradient problem in machine learning.[6]

Interestingly, the vanishing gradient problem does not affect CNNs in computer vision problems. However, when it comes to sequential data and RNNs, the vanishing gradient can have a significant impact. The vanishing gradient problem means that RNNs are unable to learn from early layers (early time steps), which causes it to have poor long-term memory.

2.4Proposed System-LSTM:

LSTMs are a variation of RNNs, and they solve the long-term dependency problem faced by conventional RNNs. Before we dive into the technicalities of LSTMs, it is useful to understand the intuition behind them.

LSTMs – the intuition

As we explained in the previous section, LSTMs were designed to overcome the problem with long-term dependencies. Let's assume we have this movie review:

"I loved this movie! The action sequences were on point and the acting was terrific. Highly recommended!"

The task is to predict whether the reviewer liked the movie. As we read this review, we immediately understand that this review is positive. In particular, the following words (highlighted) are the most important:

"I loved this movie! The action sequences were on point and the acting was terrific. Highly recommended!"

If we think about it, only the highlighted words are important, and we can ignore the rest of the words. This is an important strategy. By selectively remembering certain words, we can ensure that our neural network does not get bogged down by too many unnecessary words that do not provide much predictive power. This is an important distinction of LSTMs over conventional RNNs. Conventional RNNs have a tendency to remember everything (even unnecessary inputs) that results in the inability to learn from long sequences. By contrast, LSTMs selectively remember important inputs (such as the preceding highlighted text), and this allows them to handle both short- and long-term dependencies. [7]

The ability of LSTMs to learn from both short- and long-term dependencies gives it its name, **long short-term memory** (**LSTM**).

Inside an LSTM network:

LSTMs have the same repeating structure of RNNs that we have seen previously. However, LSTMs differ in their internal structure. The following

Journal of Theoretical and Applied Information Technology
15th June 2023. Vol.101. No 11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

diagram shows a high-level overview of the repeating unit of an LSTM:[8]

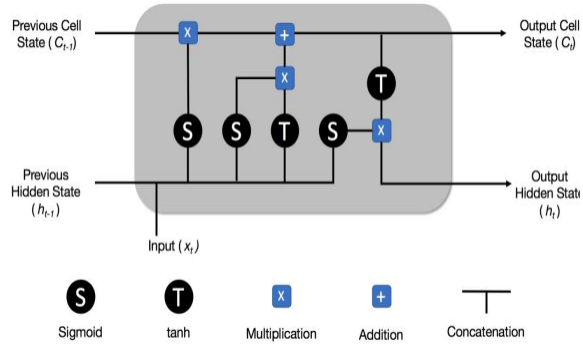


Figure 4. Repeating Unit Of An LSTM:

LSTMs have the ability to selectively remember important inputs and to forget the rest. The internal structure of an LSTM allows it to do that. [9]

An LSTM differs from a conventional RNN in that it has a cell state, in addition to the hidden state. You can think of the cell state as the current memory of the LSTM. It flows from one repeating structure to the next, conveying important information that has to be retained at the moment. In contrast, the hidden state is the overall memory of the entire LSTM. It contains everything that we have seen so far, both important and unimportant information. [9]

The LSTM releases information between the hidden state and the cell state via three important gates:

- Forget gate
- Input gate
- Output gate

Like physical gates, the three gates restrict the flow of information from the hidden state to cell state.

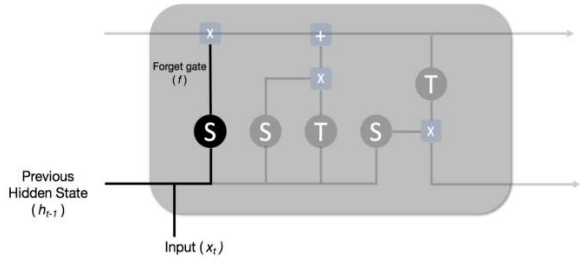


Figure 5. Repeating Unit Of An LSTM-2

The **Forget gate (f)** forms the first part of the LSTM repeating unit, and its role is to decide how much data we should forget or remember from the previous cell state. It does so by first concatenating the **Previous Hidden State (h_{t-1})** and the current **Input (x_t)**, then passing the concatenated vector through a sigmoid function. Recall that the sigmoid function outputs a vector with values between 0 and 1. A value of 0 means to stop the information from passing through (forget), and a value of 1 means to pass the information through (remember).

The output of the forget gate, *f,* is as follows:

$$f = \sigma(concatenate(h_{t-1}, x_t))$$

2.5Input gate:

The next gate is the **Input gate (i)**. The **Input gate (i)** controls how much information to pass to the current cell state. The input gate of an LSTM is highlighted in the following diagram

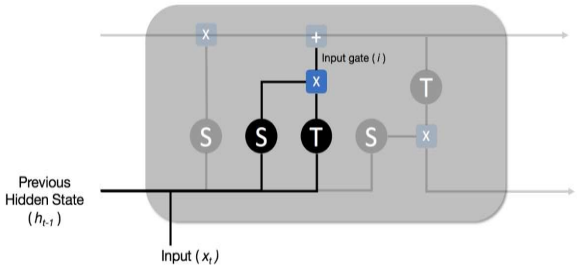


Figure 6. Input Gate

Just like the forget gate, the **Input gate (i)** takes as input the concatenation of the **Previous Hidden State (h_{t-1})** and the current **Input (x_t)**. It then passes two copies of the concatenated vector through a sigmoid function and a tanh function, before multiplying them together.

The output of the input gate, *i,* is as follows:

$$i = \sigma(concatenate(h_{t-1}, x_t)) * tanh(concatenate(h_{t-1}, x_t))$$

Now we have what is required to compute the current cell state (C_t) to be output. This is illustrated in the following diagram:

Journal of Theoretical and Applied Information Technology
15th June 2023. Vol.101. No 11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2.6 Output gate:

Finally, the output gate controls how much information is to be retained in the hidden state. The output gate is highlighted in the following diagram:

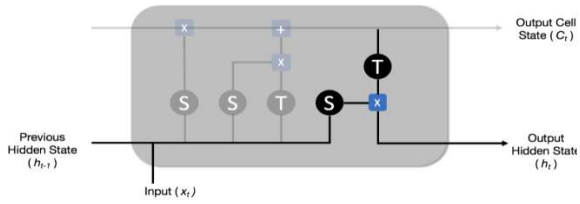


Figure7: Output Gate:

First, we concatenate the **Previous Hidden State (h_{t-1})** and the current **Input (x_t)**, and pass it through a sigmoid function. Then, we take the current cell state (C_t) and pass it through a tanh function. Finally, we take the multiplication of the two, which is passed to the next repeating unit as the hidden state (h_t). This process is summarized by the following equation:

$$h_t = \sigma(concatenate(h_{t-1}, x_t)) * tanh(C_t)$$

Input: The input to our neural network shall be IMDb movie reviews. The reviews will be in the form of English sentences. The dataset provided in Keras has already encoded the English words into numbers, as neural networks require numerical inputs.

Zero Padding:

Movie reviews have different lengths, and therefore the input vectors have different sizes. This is an issue, as neural networks only accept fixed-size vectors. To address this issue, we define a *maxlen* parameter. The *maxlen* parameter shall be the maximum length of each movie review. Reviews that are longer than *maxlen* will be truncated, and reviews that are shorter than *maxlen* will be padded with zeros.

The following diagram illustrates the zero padding process:

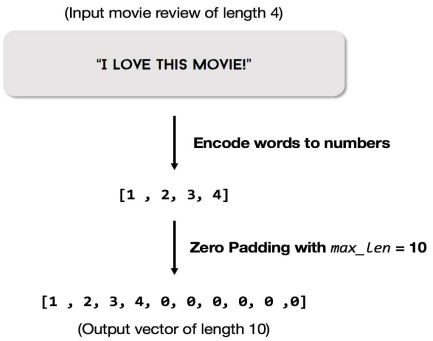


Figure 8: Zero Padding Process:

2.8Word Embedding:

The first layer in our neural network is the word embedding layer. Word embeddings are a learned form of vector representation for words. The word embedding layer takes in words as input, and then outputs a vector representation of these words. The vector representation should place similar words close to one another, and dissimilar words distant from one another. The word embedding layer learns this vector representation during training.[9]

Representing Words as Vectors:

We need to represent words as input data for our neural network i.e. we need to represent the words as vectors.

One-hot Encoding: A one hot encoding is a representation of categorical variables as binary vectors.

This first requires that the categorical values be mapped to integer values. Then, each integer value is represented as a binary vector that is all zero values except the index of the integer, which is marked with a 1.

Let's consider phrases such as the following:

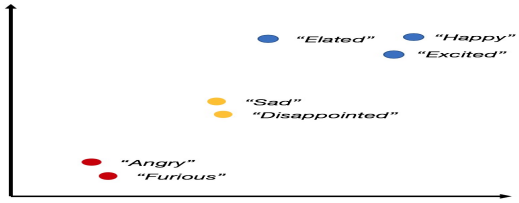
- Happy, excited
- Happy
- Excited

There are several problems with this one-hot encoded representation. Firstly, the number of axes depends on the number of unique words in our dataset. As we can imagine, there are

tens of thousands of unique words in the English dictionary. If we were to create an axis for each word, then the size of our vector would quickly grow out of hand. Secondly, such a vector representation would be extremely sparse (full of zeros). This is because most words appear only once in each sentence/paragraph. It is difficult to train a neural network on such a sparse vector.[9]

Word Embeddings: It is learned form of vector representation for words. The main advantage of word embeddings is that they have fewer dimensions than the one-hot encoded representation, and they place similar words close to one another.

The following diagram shows an example of a word embedding:



3. LSTM Layer:

The LSTM layer takes as input the vector representation of the words from the word embedding layer, and learns how to classify the vector representation as positive or negative. As we've seen earlier, LSTMs are a variation of RNNs, which we can think of as multiple neural networks stacked on top of one another.

3.1Dense Layer:

The next layer is the dense layer (fully connected layer). The dense layer takes as input the output from the LSTM layer, and transforms it into a fully connected manner. Then, we apply a sigmoid activation on the dense layer, so that the final output is between 0 and 1.

3.2Output:

The output is a probability between 0 and 1, representing the probability that the movie review is positive or negative. A probability near to 1 means that the movie review is positive, while a probability near to 0 means that the movie review is negative.

Note:

There are certain parameters we need to decide when we compile our model. They are as follows:

3.3Loss function:

It's a method of evaluating how well specific algorithm models the given data. If predictions deviate too much from actual results, loss function would give a very large number.
We use a *binary_crossentropy* loss function when the target output is binary and a *categorical_crossentropy* loss function when the target output is multi-class. Since the sentiment of movie reviews in this project is **binary** (that is, positive or negative), we ll use a *binary_crossentropy* loss function.

Optimizer:

order to minimize the error observed using loss nction, we use optimizer.
The choice of optimizer is an interesting problem in LSTMs. Certain optimizers may not work for certain datasets, due to the vanishing gradient and the exploding gradient problem .It is often impossible to know beforehand which optimizer works better for the dataset.
Therefore, the best way to know is to train different models using different optimizers, and to use the optimizer that gives the best results. We try the *SGD, RMSprop*, and the *ADAM* optimizer.

3.5Implementation:

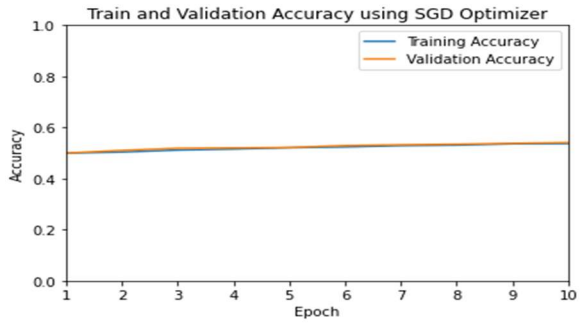
- Importing necessary packages and classes:
- Import IMDB Dataset:
- Zero Padding:
- Model Building:
- Training the model:
- Plotting the accuracy per epoch:
- Plotting the confusion matrix:

Journal of Theoretical and Applied Information Technology
15th June 2023. Vol.101. No 11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4. RESULTS:

SGD Optimizer:



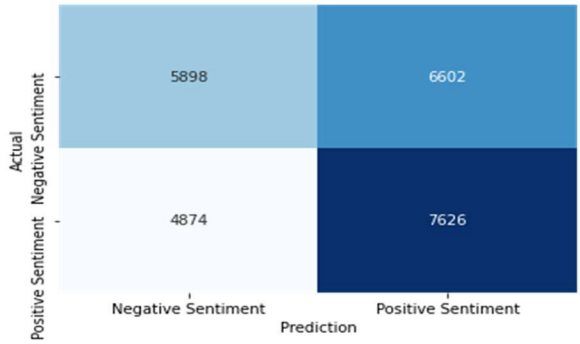
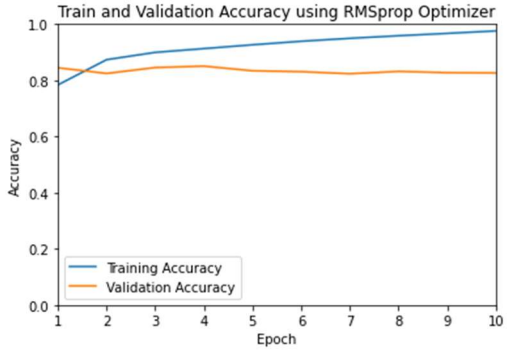


Figure 9: training and validation accuracy

The training and validation accuracy is stuck at 50%. Essentially, this shows that the training has failed and our neural network performs no better than a random coin toss for this binary classification task. Clearly, the SGD optimizer is not suitable for this dataset and this LSTM network

RMSprop Optimizer



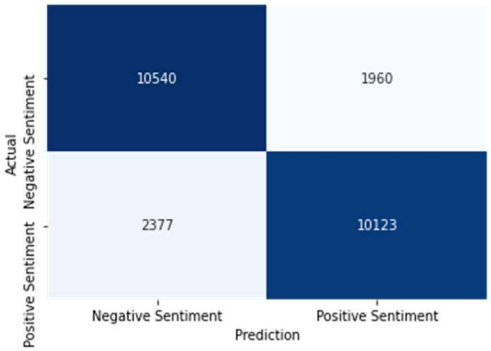
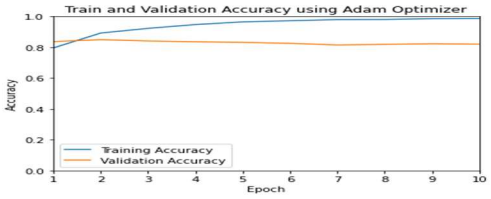


Figure10: RMSprop Optimizer

Within 10 epochs, our model is able to achieve a training accuracy of more than 95% and a validation accuracy of around 85%. Clearly, the RMSprop optimizer performs better than the sgd optimizer for this task

ADAM Optimizer:



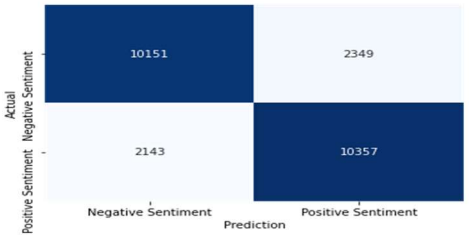


Figure11.ADAM Optimizer:

The *ADAM* optimizer does pretty well. From the preceding graph, we can see that the Training Accuracy is almost 100% after 10 epochs, while the Validation Accuracy is around 80%. This gap of 20% suggests that overfitting is happening when the *ADAM* optimizer is used.

By contrast, the gap between training and validation accuracy is smaller for the *RMSprop* optimizer. Hence, we conclude that the *RMSprop* optimizer is the most optimal for this dataset and the LSTM network, and we shall use

the model built using the *RMSprop* optimizer from this point onward.

4. Conclusion: In this project, we have built a model to classify movie reviews using LSTM network in combination with *RMSprop* optimizer. The model overcomes the Long-Term dependency and vanishing gradient problems of RRN and classifies the reviews in the test set with accuracy 85%. LSTM-based neural networks fail to detect sarcasm and other subtleties in our language. NLP is an extremely challenging subfield of machine learning that researchers are still working on today

REFERENCES:

[1]. James Loy, "Neural Network Projects with Python", Packt Publishing, ISBN:9781789138900

[2]. 2.Stone, Philip J., Dexter C. Dunphy, and Marshall S. Smith. "The general inquirer: A computer approach to content analysis." MIT Press, Cambridge, MA (1966).

[3]. 3.Gottschalk, Louis August, and Goldine C. Gleser. The measurement of psychological states through the content analysis of verbal behavior. Univ of California Press, 1969.

[4]. 4.Cataldi, Mario; Ballatore, Andrea; Tiddi, Ilaria; Aufaure, Marie-Aude (2013-06-22). "Good location, terrible food: detecting feature sentiment in user-generated reviews". Social Network Analysis and Mining. 3 (4): 1149–1163. CiteSeerX 10.1.1.396.9313. doi:10.1007/s13278-013-0119-7. ISSN 1869-5450. S2CID 5025282.

[5]. 5.Thelen, Michael; Riloff, Ellen (2002-07-06). "A bootstrapping method for learning semantic lexicons using extraction pattern contexts". Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10. EMNLP '02. USA: Association for Computational Linguistics.

[6]. 6.Yu, Hong; Hatzivassiloglou, Vasileios (2003-07-11). "Towards answering opinion questions: separating facts from opinions and identifying the polarity of opinion sentences". Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing. EMNLP '03. USA: Association for Computational Linguistics: 129–136.

[7]. 7.Sepp Hochreiter; Jürgen Schmidhuber (1997). "Long short-term memory". Neural Computation.

[8]. 8.Li, Xiangang; Wu, Xihong (2014-10-15). "Constructing Long Short-Term Memory based Deep Recurrent Neural Networks for Large Vocabulary Speech Recognition". arXiv:1410.4281 [cs.CL].

[9]. 9.Ma, Yukun; et al. (2018). "Targeted aspect-based sentiment analysis via embedding commonsense knowledge into an attentive LSTM". Proceedings of AAAI. pp. 5876–5883.