An Introduction to Recurrent Neural Networks and their Application in Sentiment Analysis

Akshat K.

August 2022

Abstract

Over the years Artificial Neural Networks have become an incredibly powerful tool, finding many varied applications in the modern world. Artificial Neural Networks, which are modeled on the structure of the human brain, have the ability to generate complex inferences from data that was previously impossible with traditional algorithms. With the exponential increase in the amount and complexity of data, Artificial Neural Networks are becoming integral to data analytics. However, traditional Neural Networks do not have the ability to effectively process sequential data, such as text or video. This is one of the main drawbacks of traditional Feed Forward Neural Networks. In this paper, we will discuss the Recurrent Neural Network architecture and how its design leads to its unique ability to deal with sequential data and address the drawback of traditional Feed-Forward Neural Networks. We will then perform a comparative analysis of many different types of Recurrent Neural Networks and display their effectiveness with a standard problem from Natural Language Processing: Sentiment analysis.

1 Introduction

In general, traditional Neural Networks contain a singular Input vector and singular output vector with data passing from input to output through the layers. While this structure has many useful applications, it is entirely unable to deal with sequential data, where there are multiple input vectors and in order to make inferences the structure of that data must be preserved. In order to deal with sequential data we must use Recurrent Neural Networks which use single units of data processing blocks strung together by a hidden state to store sequence specific data. The ability to vary the number of inputs and outputs allows for a flexible chain-like structure which makes it useful for many different applications. The main focus of this paper, Sentiment analysis, is an application of a many-to-one Network.

In this paper we will focus on the structure of multiple types of Neural Networks. We will be comparing them based on their structure and accuracy on a standard Natural Language Processing application: sentiment analysis. In this paper, we find that Traditional Feed Forward Neural Networks do not have the ability to deal with Recurrent data such as text. Vanilla Recurrent Neural Networks preform better but are ultimately hindered by the vanishing/exploding gradient problem. Long-Short Term Memory networks and Gated Recurrent Units preform roughly equally on testing data.

2 Feed Forward and Recurrent Neural Network Structure

We have already discussed that both Neural Network architectures have different ways of passing information from one end to another but we have not yet discussed how exactly these networks do so. In this section we will focus on the structure of both standard feed forward networks and Recurrent Neural Networks, as well as provide their forward propagation equations. This will also show how Recurrent Neural Networks are designed to handle sequential data.

2.1 Feed Forward Neural Network Unit

We start by looking at feed forward Neural Networks as described in [1].

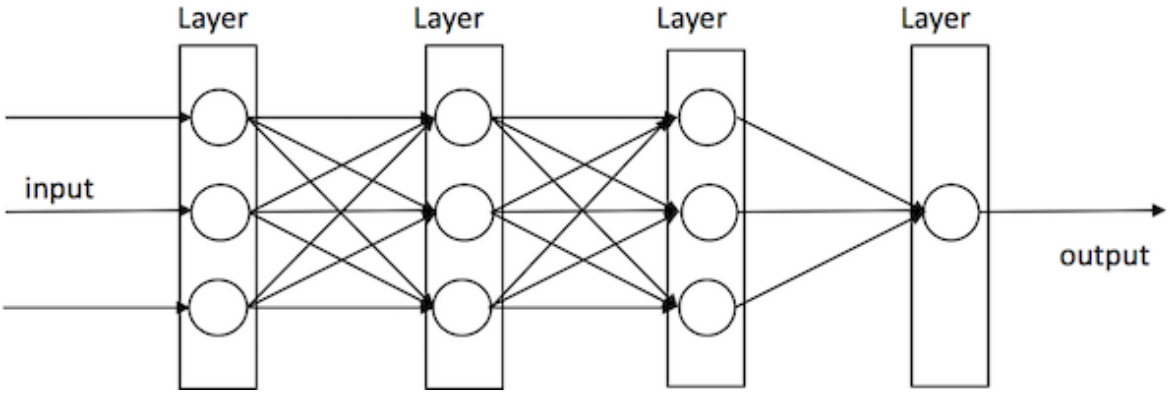

Figure 1: Structure of a standard Feed Forward Neural Network

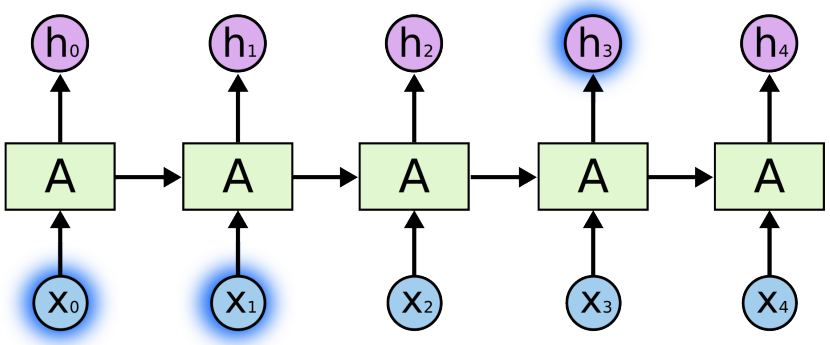

Figure 2: Structure of a Recurrent Neural Network

All Neural Networks can ultimately be reduced to a series of matrix operations. We call the input vector $\vec{x}$, the weight matrix W, the bias vector $\vec{b}$, and the output vector $\hat{y}$. Then we can represent a single dense layer with the forward propagation equation:

$$\hat{y} = \vec{x} \cdot W + \vec{b}$$

adding an activation function, $\sigma(x)$, gives us:

$$\hat{y} = \sigma(\vec{x} \cdot W + \vec{b})$$

This is the main forward propagation equation for the dense layer. Feed Forward Neural Networks work by connecting these Dense layers by making the output of one the input of the other.

2.2 Vanilla Recurrent Neural Network Unit

Now we take a look at Recurrent Neural Network as described in [2]. Let us take a look inside one of these units. For this example we will be looking at a Vanilla Recurrent Neural Network.

For the Recurrent Neural Network there are two inputs, the input at this time-step $x^{\langle t \rangle}$, and the hidden state at this time-step $h^{\langle t-1 \rangle}$. Note that now we must specify the time-step as the there are multiple inputs and the hidden state changes with each of them. In the Vanilla Recurrent Unit the forward propagation equations are:

$$h^{\langle t \rangle} = f_h \left(x^{\langle t \rangle} \cdot W_x + h^{\langle t-1 \rangle} + b_h \right)$$

$$\hat{y}^{\langle t \rangle} = f_y \left(h^{\langle t \rangle} \cdot W_y + b_y \right)$$

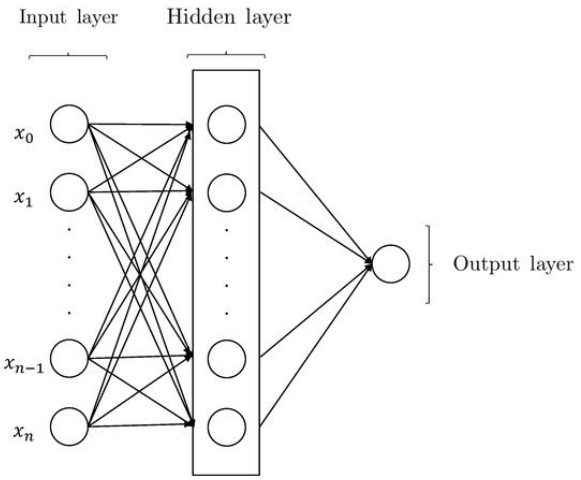

Figure 3: Internal structure of a dense layer

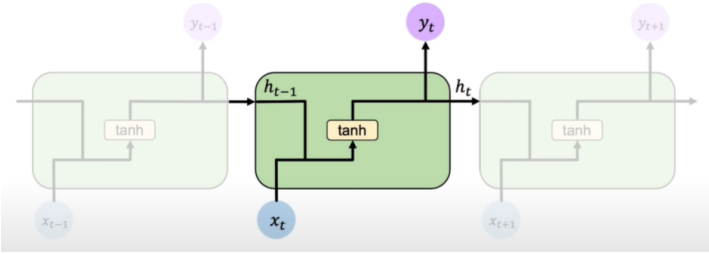

Figure 4: Structure of a Recurrent Neural Network

Note in these equations f_y and f_h are activation functions. Now we can see the hidden state is a combination of the current input and the previous hidden state. This is how the hidden state can carry sequential data. We can also see that the output at a given time-step is simply a dense layer applied on the hidden state. The fact that the hidden state acts as both an input of the unit and an output allows Recurrent layers to connect to each other and form a chain.

3 Training Recurrent Neural Networks

Regular Neural Networks train through a process known as back propagation. The main concept behind back propagation is that the cost of a Neural Network, denoted as $C(\hat{y}, y)$ (where $\hat{y}$ is the Neural Network output and y is the expected output), is a function of all the parameters of the Neural Network therefore the gradient descent vector $-\alpha\nabla C$ will give the optimal required changes for the network. For example in a network with the forward propagation equations:

$$z = x \cdot W + b$$

$$\hat{y} = \sigma(z)$$

Where x is the input vector, $\hat{y}$ is the Neural Network output, and W, b are the network parameters. Also note we use z as an intermediary variable to represent total sum before the use of the activation function. the network derivatives can be calculated as follows:

$$\frac{\partial C}{\partial W} = \frac{\partial C}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial W} = \nabla_{\hat{y}} C \sigma'(z) \cdot x^\top$$

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial b} = \nabla_{\hat{y}} C \sigma'(z)$$

3.1 Back Propagation Through Time

Recurrent Neural Networks are trained using an algorithm called Back Propagation Through Time (BPTT). It shares many similarities with standard back propagation and follows the same concept, being that optimizing a Neural Network simply comes down to expressing the cost function in terms of the network parameters and calculating the gradient of that cost function. The main difference between the two algorithms is, as the name suggests, that BPTT must also look at the time step while calculating its derivatives as the network parameters did not have one singular effect on the network but rather multiple at each time step. In this example we will be examining BPTT in Vanilla Recurrent Neural Networks. The total loss of a Recurrent Neural Network is generally calculated as the sum of all losses at each time step however since we will be working on Sentiment analysis, which is a many-to-one application of Recurrent Neural Networks, we will examine the algorithm with only one output and therefore one loss.

Let us start with an example network with only two inputs at two time-steps and one final output at the end. In this scenario the forward propagation equations will be:

$$
\begin{aligned}
z_{h^{\langle 1 \rangle}} &= x^{\langle 0 \rangle} \cdot W_x + h^{\langle 0 \rangle} \cdot W_h + b_h \\
h^{\langle 1 \rangle} &= f_h(z_{h^{\langle 1 \rangle}}) \\
z_{h^{\langle 2 \rangle}} &= x^{\langle 1 \rangle} \cdot W_x + h^{\langle 1 \rangle} \cdot W_h + b_h \\
h^{\langle 2 \rangle} &= f_h(z_{h^{\langle 2 \rangle}}) \\
z_{\hat{y}} &= h^{\langle 2 \rangle} \cdot W_y + b_y \\
\hat{y} &= f_y(z_y) \\
C(\hat{y}, y)
\end{aligned}
$$

Where $x^{\langle t \rangle}$ are the inputs at time step t, $h^{\langle t \rangle}$ are the hidden states at time step t and W_x, W_y, W_h, b_h, b_y are the network parameters. So now to calculate the derivatives:

$$
\frac{\partial C}{\partial b_y} = \frac{\partial C}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_{\hat{y}}} \frac{\partial z_{\hat{y}}}{\partial b_y} = \nabla_{\hat{y}} C f_y'(z_{\hat{y}})
$$

$$
\frac{\partial C}{\partial W_y} = \frac{\partial C}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_{\hat{y}}} \frac{\partial z_{\hat{y}}}{\partial W_y} = \nabla_{\hat{y}} C f_y'(z_{\hat{y}}) \cdot h^{\langle 2 \rangle \top}
$$

$$
\frac{\partial C}{\partial W_h} = \frac{\partial C}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_{\hat{y}}} \frac{\partial z_{\hat{y}}}{\partial h_2} \left(\frac{\partial h_2}{\partial W_h} + \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial W_h} \right) = \nabla_{\hat{y}} C f_y'(z_{\hat{y}}) W_y \left(f_h'(z_{h^{\langle 2 \rangle}}) h^{\langle 1 \rangle \top} + f_h'(z_{h^{\langle 2 \rangle}}) W_h f_h'(z_{h^{\langle 1 \rangle}}) h^{\langle 0 \rangle \top} \right)
$$

$$
\frac{\partial C}{\partial W_x} = \frac{\partial C}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_{\hat{y}}} \frac{\partial z_{\hat{y}}}{\partial h_2} \left(\frac{\partial h_2}{\partial W_x} + \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial W_x} \right) = \nabla_{\hat{y}} C f_y'(z_{\hat{y}}) W_y \left(f_h'(z_{h^{\langle 2 \rangle}}) x^{\langle 1 \rangle \top} + f_h'(z_{h^{\langle 2 \rangle}}) W_h f_h'(z_{h^{\langle 1 \rangle}}) x^{\langle 0 \rangle \top} \right)
$$

$$
\frac{\partial C}{\partial b_h} = \frac{\partial C}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_{\hat{y}}} \frac{\partial z_{\hat{y}}}{\partial h_2} \left(\frac{\partial h_2}{\partial b_h} + \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial b_h} \right) = \nabla_{\hat{y}} C f_y'(z_{\hat{y}}) W_y \left(f_h'(z_{h^{\langle 2 \rangle}}) + f_h'(z_{h^{\langle 2 \rangle}}) W_h f_h'(z_{h^{\langle 1 \rangle}}) \right)
$$

4 The Issue with Vanilla Recurrent Neural Networks

Vanilla Neural Networks preform modestly well on smaller inputs, however as the size of the sequence increases they lose the ability to make inferences off the data. This is due to something called the vanishing/exploding gradient problem. In this section we will present how this problem occurs and some of the most popular solutions.

4.1 What is the Vanishing/Exploding Gradient Problem

The vanishing/exploding gradient problem occurs during back propagation through time and causes there to be little or no change to layers that are further within the network from where the loss is currently being calculated. Sometimes the opposite problem also occurs where there are disproportionately large changes further into the network.

Lets start by looking at a general equation for one of the derivatives encountered in BPTT:

$$\frac{\partial C}{\partial W_h} = \frac{\partial C}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_{\hat{y}}} \frac{\partial z_{\hat{y}}}{\partial h_n} (\sum_{k=2}^{n} \frac{\partial h_{k-1}}{\partial W_h} \prod_{j=n}^{k} \frac{\partial h_j}{\partial h_{j-1}})$$

the main issue here is the $\frac{\partial h_{k-1}}{\partial W_h} \prod_{j=n}^{k} \frac{\partial h_j}{\partial h_{j-1}}$ term. This long product becomes very sensitive and even a minute change in the product would cause a massive change in the result. This can happen because of floating point approximation. So the sensitivity of this product combined with computer approximations can lead to massive changes. Many times even the repeated multiplication of an activation function between zero and one can also cause a decay as we propagate further back into the network. This is especially prevalent in our case as we have only one output at the end of the network.

5 Solutions to the Vanishing Gradient Problem

One of the main solutions to the vanishing/exploding gradient problem is developing a unit that avoids this issue. The main cause of the vanishing gradient problem is the lack of a filter between the input and the hidden state. To solve this we introduce another state called the cell state. The cell state also threads through the network but is very selective and careful about how it interacts with the input and hidden state. There are two main types of networks in this category that we will be going over in this paper, Long-Short Term Memory networks and gated recurrent Units.

5.1 Long Short Term Memory Networks

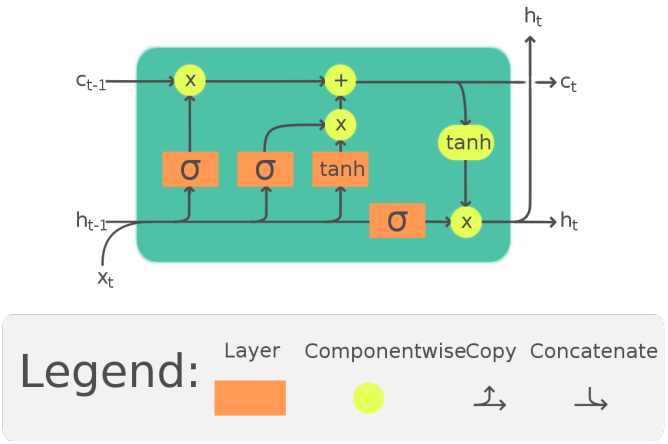

Figure 5: Internal structure of a LSTM unit

The main idea in the LSTM unit structure is the cell state. Just as the hidden state was the main novelty introduced in Vanilla Recurrent Neural Networks, the cell state is the main novelty of the LSTM. The cell state is designed to only retain the most important information from the data, whether recent or long term, and it does so using gates. These gates help control the flow of information into the cell state so that the network has the ability to forget unnecessary information but also has the ability to recall information from very far behind in the chain. The LSTM does this using the activation function. Since sigmoid applied on a vector will output a vector filled with numbers between 0 and 1 it can act as a way of gauging the importance of specific data, with 0 being unimportant and

1 being extremely important. LSTMs have three major gates, forget gate, input gate, and an output gate. These gates help change the cell state according to the data, but only make the cell state retain important information about the data. This allows it to take into account recent inputs, but also idea from much further back in the network, hence the name Long-Short Term Memory Network. Below are the feed-forward equations for a LSTM cell:

$$f^t = \sigma_g(W_f x^t + U_f h^{t-1} + b_f)$$
$$i^t = \sigma_g(W_i x^t + U_i h^{t-1} + b_i)$$
$$o^t = \sigma_g(W_o x^t + U_o h^{t-1} + b_o)$$
$$\tilde{c}^t = \sigma_c(W_c x^t + U_c h^{t-1} + b_c)$$
$$c^t = f^t \odot c^{t-1} + i^t \odot c^t$$
$$h^t = o^t \odot \sigma_h(c_t)$$

where σ represents the sigmoid activation function, W and U are matrices, and $\odot$ represents element-wise multiplication

5.2 Gated Recurrent Units

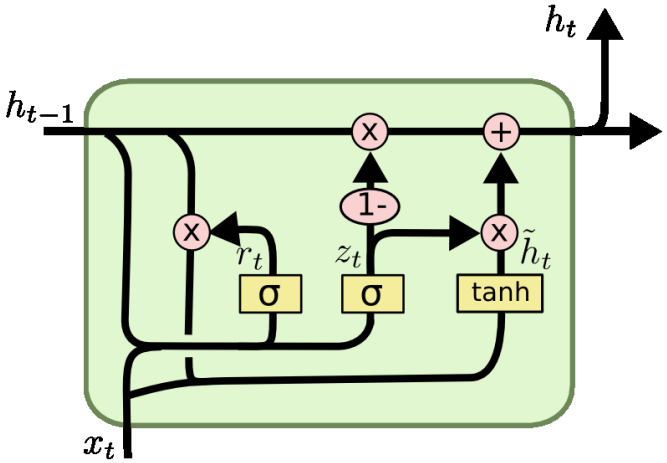

Figure 6: Internal structure of a GRU unit

Similarly to LSTM networks, Gated Recurrent Units(GRU for short) were designed to counteract the vanishing gradient problem. In LSTMs there is a hidden state and then a cell state, with the cell state being regulated by three gates, the input gate, output gate, and forget gate. In the gated recurrent unit there are only two gates, the reset and update gate. These gates serve a similar function to the gates in the LSTM and work to avoid the vanishing gradient problem in the hidden state and help the network retain important long term information while taking into account recent information. The Gated Recurrent Unit was developed before the LSTM and LSTMs find more use in modern day applications. Below are the feed-forward equations for a GRU cell:

$$z^t = \sigma_g(W_z x^t + U_z h^{t-1} + b_z)$$
$$r^t = \sigma_g(W_r x^t + U_r h^{t-1} + b_r)$$
$$\hat{h}^t = \phi_h(W_h x^t + U_h(r^t \odot h^{t-1}) + b_h)$$
$$h_t = z_t \odot \hat{h}^t + (1 - z_t) \odot h^{t-1}$$

6 Conclusion and an example

Feed Forward Neural Networks have many applications, however they are not very adept at handling sequential data. Recurrent Neural Networks were designed as a solution to this problem. Standard Vanilla Recurrent Neural Networks can handle smaller sequences of data and make important inferences off the sequential structure of the data, however the vanishing gradient problem sets in once the size of the data gets too large. In order to solve this issue we mainly have two different types of networks, Long Short Term Memory Networks and Gated Recurrent Units. These networks work to minimize the effects of the vanishing gradient problem by introducing gates into the flow of data which helps regulate which information should be kept and which information should be thrown out.

6.1 Short example of each type of Recurrent Neural Network

To illustrate the effectiveness of Recurrent Neural Networks, we will use sentiment analysis. The data set used is the IMDB sentiment analysis data set which consists of movie reviews and the output for each network is a number between 0 and 1 representing the sentiment of the review. Below is a table summarizing the results of each test and the training graphs for the respective networks.

Table 1: Accuracy of tested Networks

Type of network	Eval on training set	Eval on testing set
Feed Forward	50.57%	49.42%
Vanilla RNN	81.3%	81.96%
LSTM	93.86%	88.79%
GRU	96.53%	88.88%

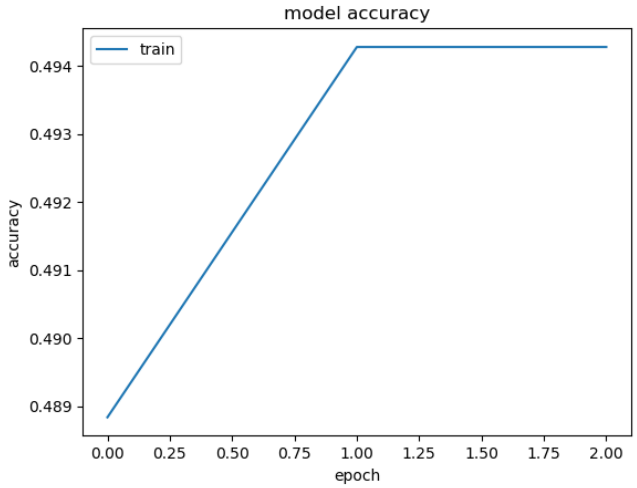

Figure 7: accuracy of Feed Forward Model at each epoch

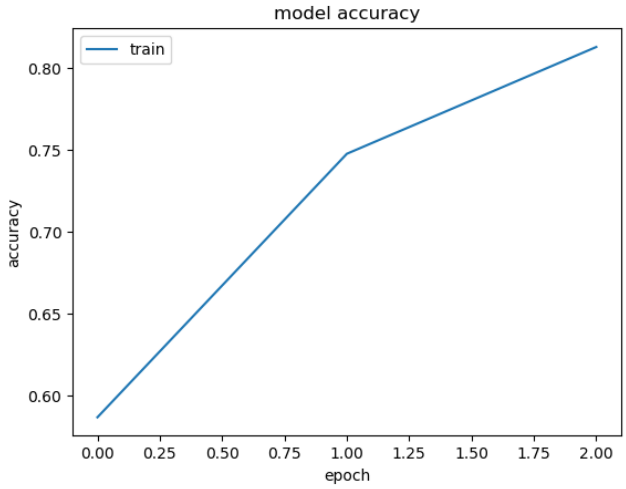

Figure 8: accuracy of Vanilla RNN Model at each epoch

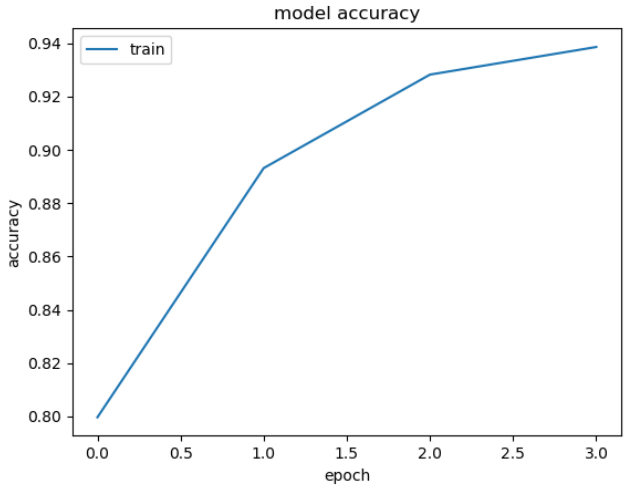

Figure 9: accuracy of LSTM Model at each epoch

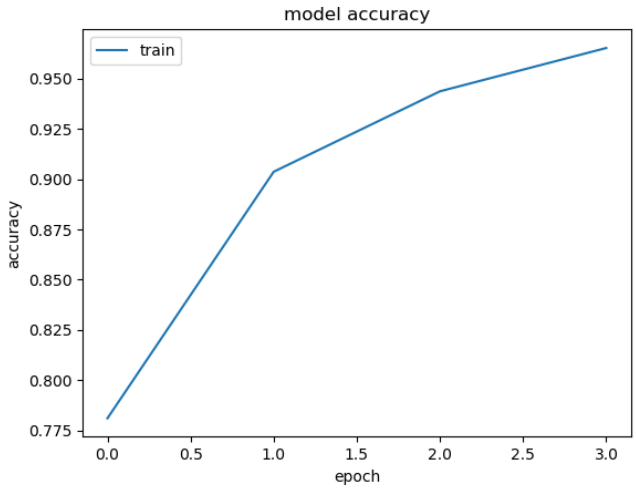

Figure 10: accuracy of GRU Model at each epoch

References

[1] G. Bebis and M. Georgiopoulos. *Feed-forward neural networks*, volume 13. IEEE, 1994.

[2] Jeffrey L. Elman. *Finding Structure in time*, volume 14. Cognitive Science, 1990.