**Machine Learning Tutorial**

**Recurrent Neural Network (RNN)**

Student Name – Upputuri Praveen

Student Number – 23111387

GitHubLink –

Dataset Link - <https://www.kaggle.com/datasets/vijayvvenkitesh/microsoft-stock-time-series-analysis?resource=download>

**Introduction**

Recurrent Neural Network (RNN) is a deep neural network which has the ability to predict the output. This ability comes from the “remembering” property of recurrent neural network, it can store the past information in its hidden cells which then can be used to predict the future outputs or sequences. For example, if the input fed to RNN is “Once upon a time”, then the RNN will generate text like “there lived a king”, “there was a kingdom” this is called as text generation, there are many applications of RNN like whether forecasting, language translation, speech recognition, time series prediction, etc.

RNN differ from other neural networks is by its ability to store the information from its past inputs, RNNs are trained on the sequential data to predict the future outputs.

Recurrent Neural Network

The above figure shows a RNN which shows the output being fed back into the network this helps the network to store the previous inputs and use this information whenever is needed for making decisions, this feature separates the RNN from other neural networks where inputs and outputs are assumed to be independent but in RNN the loop which feedback output into network gives it the ability to remember, unlike in other networks where the weights are not shared equally, RNN weights are shared equally throughout the network, these weights are updated using backpropagation and since RNN works on sequential data we uses updated backpropagation called Backpropagation Through Time (BPTT).

**RNN Components**

W

h

V

U

Recurrent Neuron

The basic building block of RNN is Recurrent Neuron which consists of hidden state that stores the previous input while training in sequential data. Here U, V and W are wights, h is hidden state. X is input.

W

W

W

….. …..

V

V

V

U

U

U

Unrolled RNN

The above shown figure is unrolled RNN. When we have sequential data we train RNN present and Past inputs to make predictions in that case we will be having more than 1 input and more than 1 output which results into above shown networks called unrolled RNN.

RNN shares equal weights through out the network that means in each layer V, W, U are same they do not change.

Hidden state is calculated: = (U \* X + W \* + B) - Equ(1)

Output calculation: Y = activation function(V \* + C) - Equ(2)

Activation functions are sigmoid, tanh, ReLU, is current hidden state, is previous hidden state, U, V and W are weights, B and C are biases.

These RNN will not be used widely because of Vanishing/Exploding Gradients problem.

**Vanishing/Exploding Gradients**

In the below figure only consider input and W be the weight, and we have unfolded RNN four times.

input

W

W

W

By keeping the weights and biases as constant in Equ(2) as constant since they don’t change through out the network. Then, Output Y α V \* - Equ(3)

i.e, output is directly proportional to V that means if V increases output Y increase, if V decreases output Y decreases. This is the basis for vanishing/exploding gradients problem. Let us assume weight V as W.

From Equ(1) if we keep weights and biases as constant then, α X

Therefore, from Equ(2) if we replace in Equ(3) by X

Then, output Y α W \* input (X) output Y is directly proportional to weight W and input X .since is directly proportional to input(X).

Input in the first layer will be multiplied by weight W and passed to second layer this will continue till last layer. Let us assume W = 2 then, input will be multiplied by

so, for fist layer input \* 2

for second layer input \* 2 \* 2

for third layer input \* 2 \* 2 \* 2

for last layer input \* 2 \* 2 \* 2 \* 2

by the time it reaches last layer input is multiplied by . i.e input is multiplied by . In real world cases its possible that input is sequence can contain more than 100 values so the input will be scaled which is a very huge number this is called Exploding Gradient.

Similarly, let us assume that W = 0.5 then, input will be multiplied by

so, for fist layer input \* 0.5

for second layer input \* 0.5\* 0.5

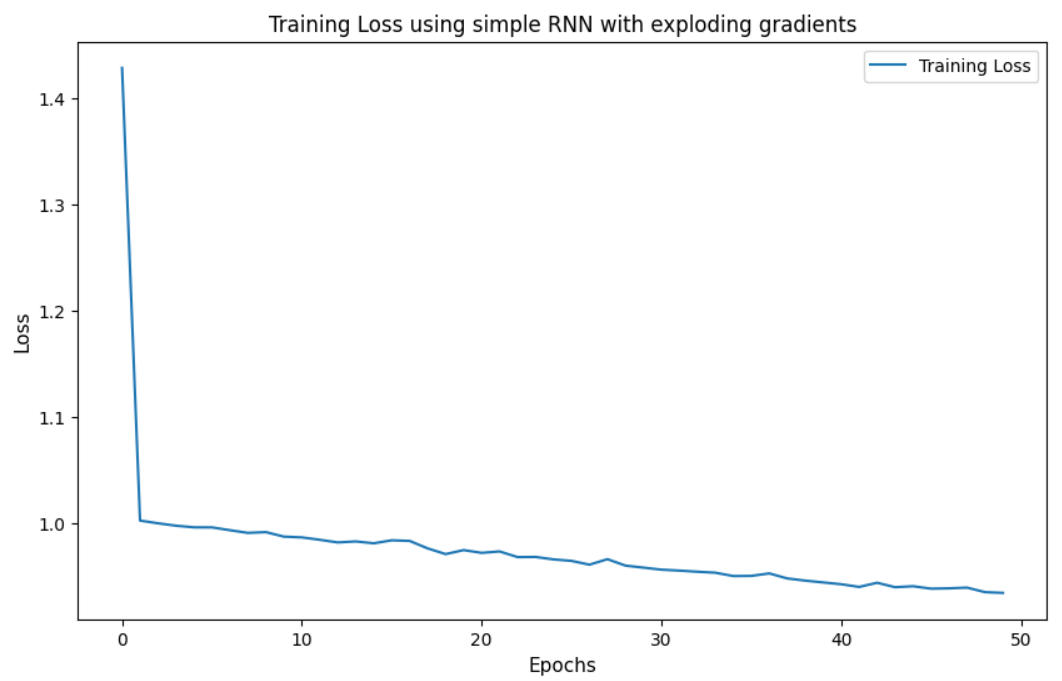
for third layer input \* 0.5\* 0.5\* 0.5

for last layer input \* 0.5\* 0.5\* 0.5\* 0.5

by time it reaches fourth layer input is scaled by , which is a very small number. In case of real world scenarios where the input contains huge amount of data to be trained this number will become very small this is called vanishing gradient. Due to this problems training becomes very difficult when training for long sequential data as it forgets after some time.

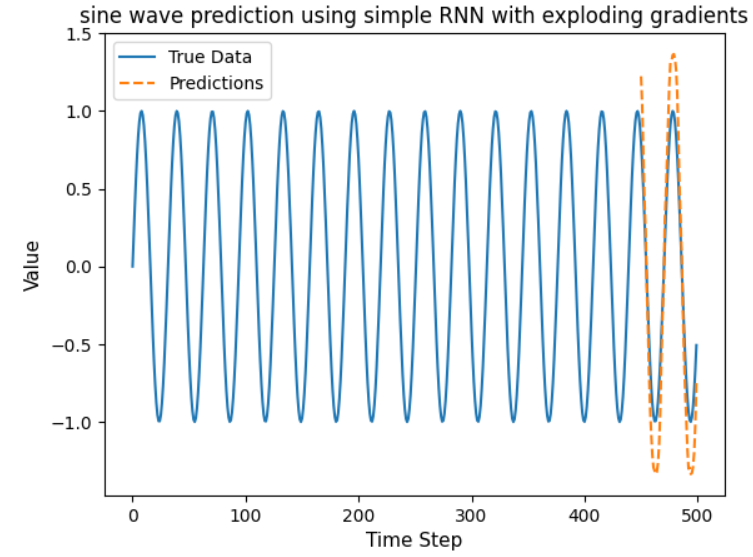
The dataset contains five columns Date and time, highest price of stock, lowest price of stock, closing value of stock and volume of stock on that day. This dataset is used to train the RNN to demonstrate the losses using vanishing/exploding gradients and characteristics.

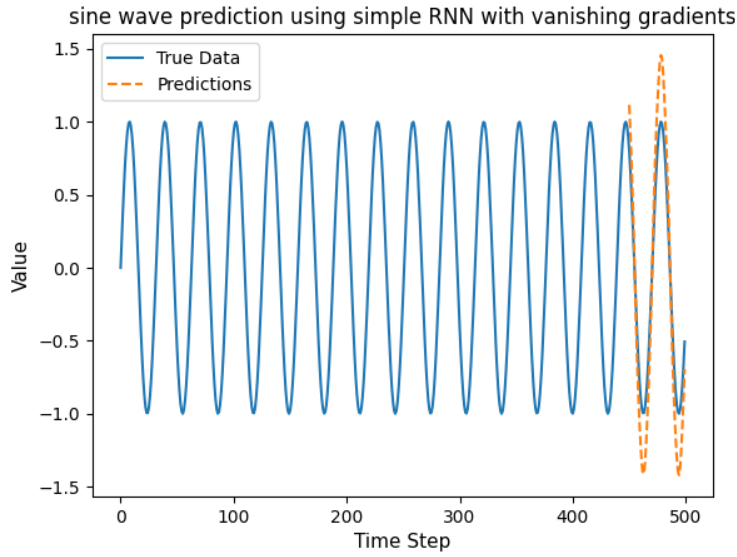


It can be observed that with vanishing gradients losses are oscillating.

Above figure shows the losses for RNN with exploding gradients.

**Predictions using RNN**

The below figures show the sine wave predictions using RNN with vanishing and exploding gradients.



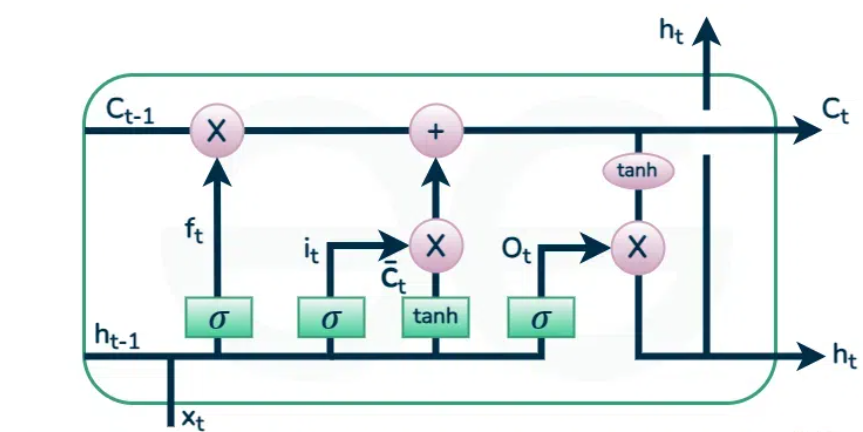
**Long Short Term Memory (LSTM)**

Long Short Term Memory (LSTM) solves the drawback of RNN by remembering some part of information from the input data that will be used for predicting output.

LSTM contains

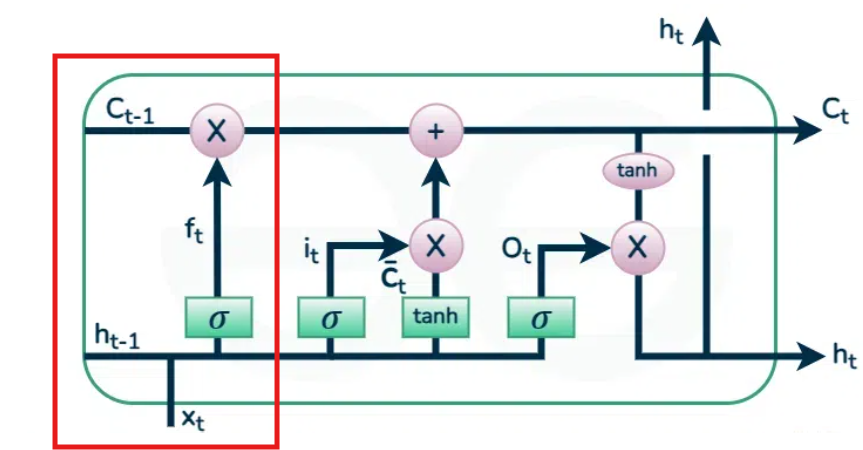
Forget Gate – It decides how much part of input to be removed from cell

Input Gate - It determines what part of input to be memorized by cell

****Output Gate – determines the output from cell

Ref: <https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/>

Above figure shows the LSTM and its components

**Forget Gate**

Forget gate

The Highlighted area shows the Forget gate

The equation for forget gate is = sigmoid activation function( \*( \* ) + )

Where,

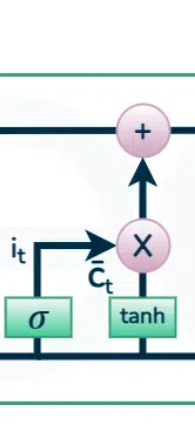
= weight of forget gate

= previous hidden state, = bias of forget gate and = input

In this gate input is multiplied by previous hidden state and passed to the sigmoid activation function where it is multiplied by weight of forget gate and final output will be added with forget gate bias. By this manner gate will decide how much input to be removed from cell.

**Input Gate**

The below figure shows the input gate



Input gate

Equations:

= sigmoid activation function( \*( \* ) + )

= tanh( \*( \* ) + )

= \* + \*

In the input gate input is multiplied by previous hidden state and multiplied by weight and added with bias and then passed to sigmoid function output from this function is called and a vector is created using tanh function which has values -1 to +1. Then the output from the input gate is calculated as the addition of part of removed input sequence from forgot gate and multiplied by

**Output Gate**

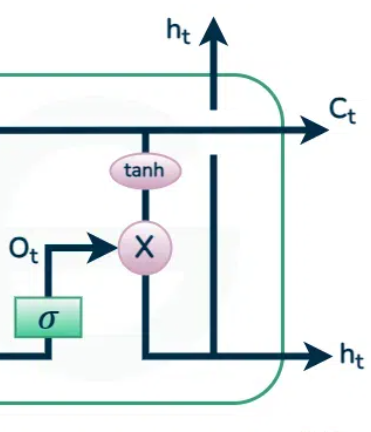
The Below figure shows the output gate which consists of a tanh activation function and a sigmoid activation function. A vector is generated by applying tanh activation and then sigmoid function is applied to \* which controls how much information to be remembered. In the last vector generated using tanh will be multiplied by the information obtained by sigmoid function and passed to next call as output and input.

Output gate equation is

= sigmoid activation function( \*( \* ) + )

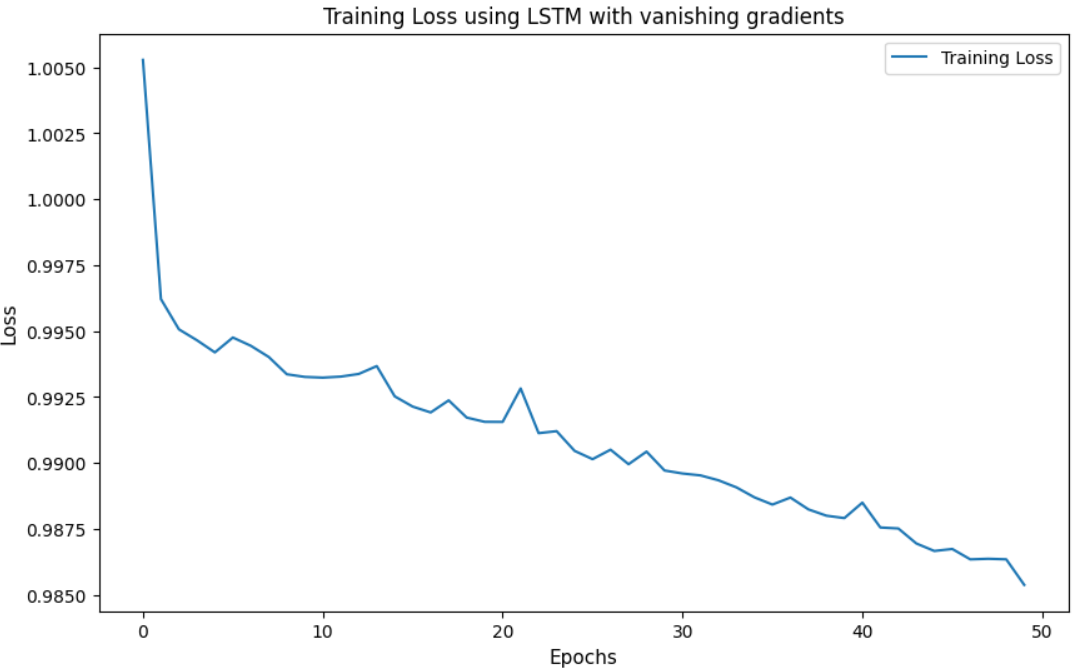
Where, is weight of output gate, is previous hidden state, is input and is bias of output gate.

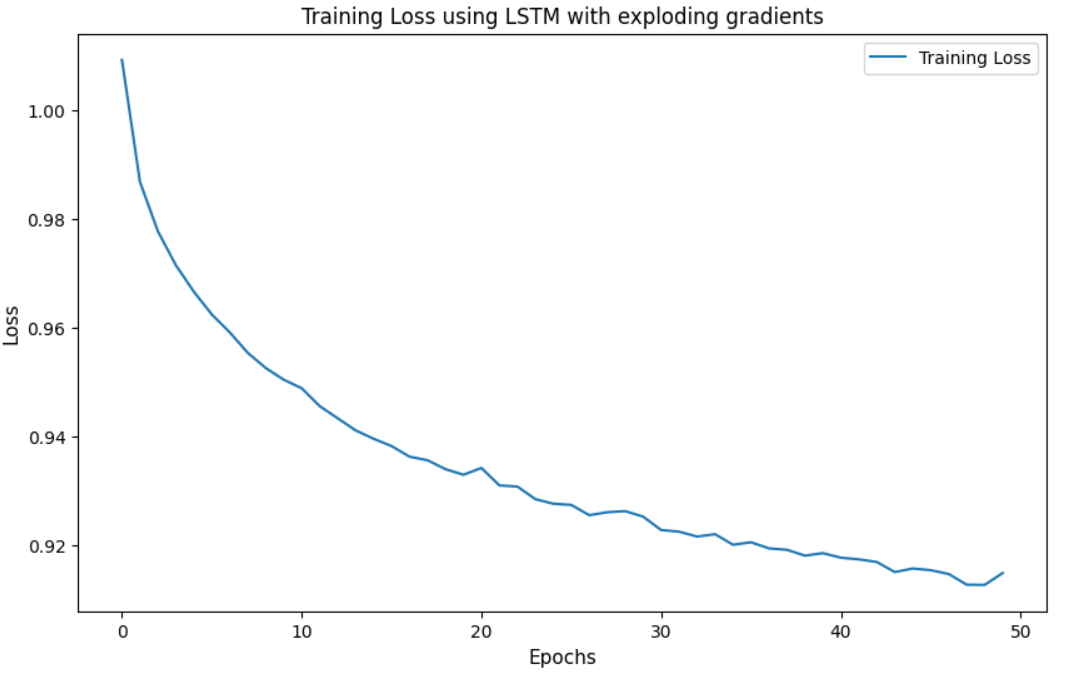
The output gate is mainly responsible for getting useful information from input.



Output gate

The below figures describes the losses which are obtained using LSTM on same dataset.

****

****

From the above two figures it can be observed that loss curves are smoother with LSTM compared with Simple RNN. LSTMs handles vanishing/exploding gradients problems very well as the oscillations in loss are decreasing.

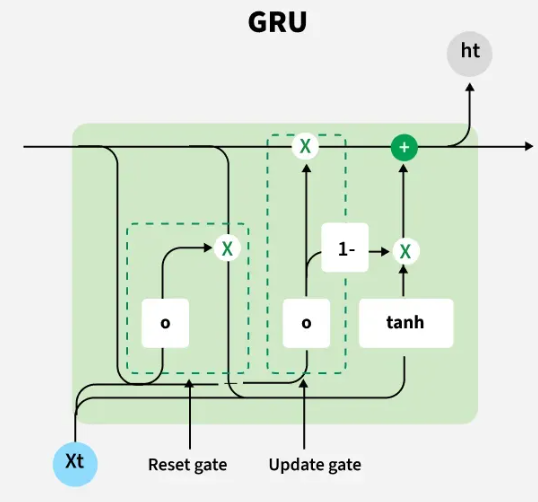
**Improvement Over Simple RNN**

Unlike in simple RNN where it fails to remember long sequence data. LSTM overcomes this disadvantage by means of controlling how much input to remember and how much input to discard due to this it remembers some parts of inputs which are then used to predict the output.

Although LSTM overcomes the Exploding/Vanishing Gradients problem it is computationally expensive, slow in training especially for long input sequences. To solve this problem GRUs are introduced

**Gated Recurrent Unit (GRU)**

Gated Recurrent Unit (GRU) has two gates Update Gate and Reset Gate which makes it less computational than LSTM, faster than LSTM.



Ref - <https://www.geeksforgeeks.org/rnn-vs-lstm-vs-gru-vs-transformers/>

The above diagram shows GRU which consists of Reset Gate and Update Gate.

Update Gate – Determines how much of information from past should be kept for future making GRU remembering details from past inputs.

Reset Gate – Determines how much past information to be removed if it is not needed.

Equations:

Reset Gate - ​=sigmoid activation function(​ \* [​ \* ​])

Where, ​ is weight of reset gate, ​ is previous hidden state, ​ is input

Update Gate - ​=sigmoid activation function(​ \* [​ \* ​])

Where, is weight of update gate,​ ​ is previous hidden state, ​ is input



The above figure show the GRU performance on dataset it can be observed that the curve is more smoother than LSTM.

**Improvements over LSTM**

Gated Recurrent Units has less architecture demands because it is constructed with only two gates, it is faster than LSTM in training and equally accurate. Due to the introduction of gates which selectively store and discard the information from inputs it eliminates the vanishing/exploding gradients problem very well.

**Conclusion**

Recurrent Neural Networks (RNN) have capability to remember data and use this data to predict the future outputs/sequences but due to vanishing/exploding gradients problem they are not used widely, to solve this problem Long Short Term Memory (LSTM) is discovered which can remember sequences in input data to predict outputs and therefore solving vanishing/exploding gradients problem but LSTM has large architectural requirements and large computational power requirements and becomes slow while training for large input sequences. To solve this issue Gated Recurrent Unit (GRU) is introduced which is faster than LSTM in training, need less computational power and as accurate as LSTM.

**References**

1 <https://aws.amazon.com/what-is/recurrent-neural-network/#:~:text=A%20recurrent%20neural%20network%20(RNN,a%20specific%20sequential%20data%20output>.

2 <https://www.ibm.com/think/topics/recurrent-neural-networks>