# Recurrent Neural Networks (RNN) for time series prediction

Amit Kumar
*Department of Computer science*
*University of Texas at Dallas*
Richardson, USA
Net Id: axk210047

Bharath Tej Chinimilli
*Department of Computer science*
*University of Texas at Dallas*
Richardson, USA
Net Id: bxc200031

Raj Mishra
*Department of Computer science*
*University of Texas at Dallas*
Richardson, USA
Net Id: rxm190093

***Abstract*—Classical Machine learning algorithms give equal importance to the all previous seen data for prediction. But, in the problems like stock market prediction, weather prediction or in sentence completion a emphasis on the recently seen data should be given and classical machine learning algorithms seems don't give any emphasis on recent data. This paper is about the implementation of Long short term memory (LSTM) neural network from scratch to predict weather in Jena, Germany.We implemented LSTM in python using libraries like pandas for data manipulation, numpy for faster mathematical calculations and matplotlib and seaborn for data visualization.**

***Index Terms*—LSTM, Time series prediction, Jena-weather prediction, neural network**

## I. Introduction and background work

For centuries people have tried to predict the weather. Ancient greeks, Indians, Babylonians etc. tried to predict the weather by recording temperatures and seeing the cloud pattern[1]. But their predictions used to have some inaccuracies due to the fact that there are many variables that have an effect on the temperature and it is impossible to find all these variables and use them to solve the temperature prediction problem[1].

With the advancement in machine learning and deep learning with the availability of lots of data and the computational power to learn from these data. Neural networks are able to learn the patterns within the data to predict temperature.

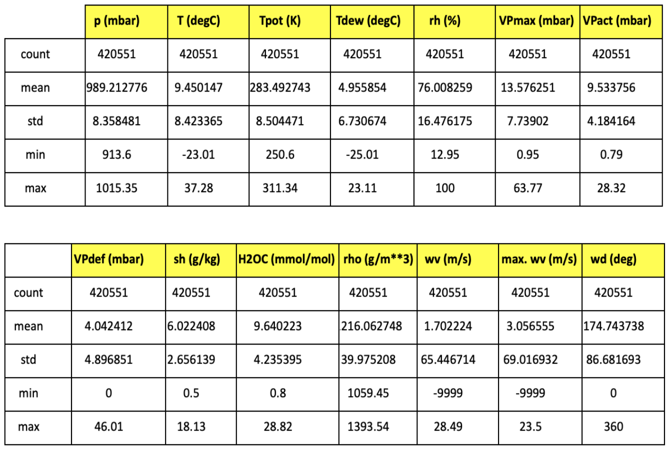
That's why for the Machine learning course project we decided to work on temperature forecasting based on Recurrent Neural Network(RNN) based architecture which has a history of working well for time series prediction problems such as stock market prediction, temperature prediction etc.[2].

In particular we used Long Short term Memory[4] which is RNN based architecture for the creation of our model.This is because in the long-term remembering RNN leads to vanishing gradients which makes it impossible for the model to learn and LSTM seems to overcome this issue[4].

## II. Dataset

The dataset we used for our project is called Jena climate dataset. It is a time-series dataset which is recorded at Max planck Institute for Biogeochemistry in Jena, Germany. The dataset consists of weather data starting from January 1$^{st}$ 2009 to December 31$^{st}$ 2016. It has recorded data for every 10 minutes between these dates. There are 15 feature in the dataset and 420,551 instances. Here is the data distribution of each dataset in Fig. 1. As you can see in Fig. 1 all the features are having different range of values. so, Min_Max scaler is used to scale the feature values between 0 and 1 so that the model converge faster.

.



| | p (mbar) | T (degC) | Tpot (K) | Tdew (degC) | rh (%) | VPmax (mbar) | VPact (mbar) |
|---|---|---|---|---|---|---|---|
| count | 420551 | 420551 | 420551 | 420551 | 420551 | 420551 | 420551 |
| mean | 989.212776 | 9.450147 | 283.492743 | 4.955854 | 76.008259 | 13.576251 | 9.533756 |
| std | 8.358481 | 8.423365 | 8.504471 | 6.730674 | 16.476175 | 7.73902 | 4.184164 |
| min | 913.6 | -23.01 | 250.6 | -25.01 | 12.95 | 0.95 | 0.79 |
| max | 1015.35 | 37.28 | 311.34 | 23.11 | 100 | 63.77 | 28.32 |

| | VPdef (mbar) | sh (g/kg) | H2OC (mmol/mol) | rho (g/m**3) | wv (m/s) | max. wv (m/s) | wd (deg) |
|---|---|---|---|---|---|---|---|
| count | 420551 | 420551 | 420551 | 420551 | 420551 | 420551 | 420551 |
| mean | 4.042412 | 6.022408 | 9.640223 | 216.062748 | 1.702224 | 3.056555 | 174.743738 |
| std | 4.896851 | 2.656139 | 4.235395 | 39.975208 | 65.446714 | 69.016932 | 86.681693 |
| min | 0 | 0.5 | 0.8 | 1059.45 | -9999 | -9999 | 0 |
| max | 46.01 | 18.13 | 28.82 | 1393.54 | 28.49 | 23.5 | 360 |

Fig. 1. Data Distribution of the features

## III. Theoretical and conceptual study of the algorithm we would like to implement

### A. *Recurrent Neural Network(RNN)*

RNN are powerful neural networks because of their internal memory. Like other neural networks RNN was also developed in the second part of the 20th century. But, nobody has seen their true potential until the last decade with the rise of computational power and the availability of large amount of data.Because they have an internal memory they can remember recent input and they can do good predictions about what comes next. That's why they are preferred in the time-series prediction, Natural Language Processing(NLP) problems like sentence completion, and in video processing problems in which the model has to predict what happens in the next frame etc. They tend to do better compared to other Deep learning architectures[5].

RNN and feed-forward neural networks are almost similar. In both neural networks the data moves only forward. Feed-forward neural networks don't have any internal memory, so they don't remember anything about the past inputs except for the training which leads to them not having any notion of time. In contrast, RNN has internal memory which means the output for the current instance has some weightage along with the next input in deciding the output of the next instance. You can see the difference of both networks in the Fig. 2 and Fig.3

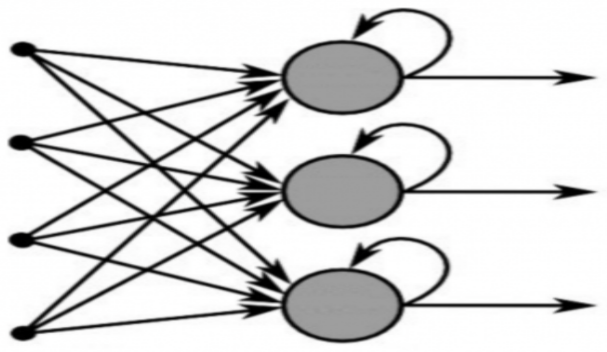


Fig. 2. Example of RNN [5]

Even though it seems like they are perfect for time-series predictions, they have two major issues to tackle. That is exploding gradients and vanishing gradients. Exploding gradients are when an algorithm keeps on giving higher weightage to a particular neuron and at a point it decides the final output irrespective of the other neuron's values. This problem will be solved by squashing the gradients. Vanishing gradients are when an algorithm becomes too small that the model stops learning or learning takes a lot of time. This problem cannot be solved as easily as exploding gradients. But, Long short-term memory which is derivative of RNN seems to tackle it compared to RNN[5].

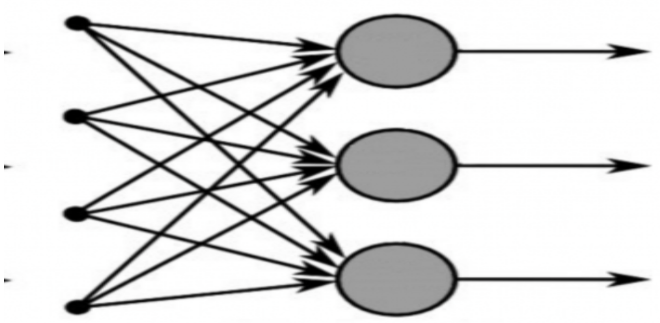


Fig. 3. Example of Feed-forward neural network [5]

### B. *Long short term memory(LSTM)*

LSTM is developed to solve the short term memory problem by Hochreiter, Sepp, and Jürgen Schmidhuber and first mentioned this architecture in their paper "Long short-term memory" [4].There are three gates in LSTM which makes its ability to remember important things for long periods of time. They are the cell gate, forget gate input gate and output gate.

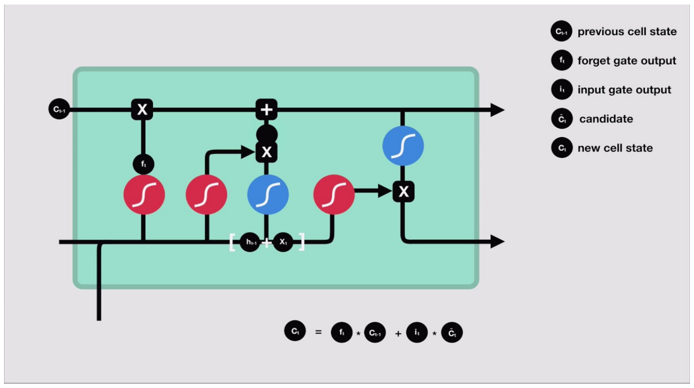


Fig. 4. LSTM [7]

**Forget gate:** this gate decides whether an information should be remembered or forgotten. The previous state output and current state input is passed through this gate which has a sigmoid function and multiplies the output with the cell state. If the output of the gate is closer to 0 then the cell state information is forgotten or if the output of the gate is near to 1 then the cell state information is remembered.

**Input gate:** This gate decides what kind of new information is important and not important. First output of the previous state and the input of the current state are passed into a sigmoid function which squashes the values between 0 and 1 to decide which part of the information and which is not. Again combine data of previous state output and current state input through a tanh function which squashes the values between -1 and 1. Now the output of the tanh function and the sigmoid function are multiplied to decide what new information should be sent to the cell state.

**Cell state:** This state is important because this state gives LSTM the ability to remember important things for long periods of time. The values in the cell state are multiplied to the output of the forget gate to forget unnecessary information. Now, the cell state is added with the output of the input gate to add new information to its memory.

**Output gate**: This gate decides what will be the output of the LSTM. First, the combined input of the current state and the output of the previous state goes through a sigmoid function. The cell state goes through a tanh function. Now, the output of the sigmoid function and the tanh function are multiplied to give the output.

## *C. Forward Propagation of LSTM:*

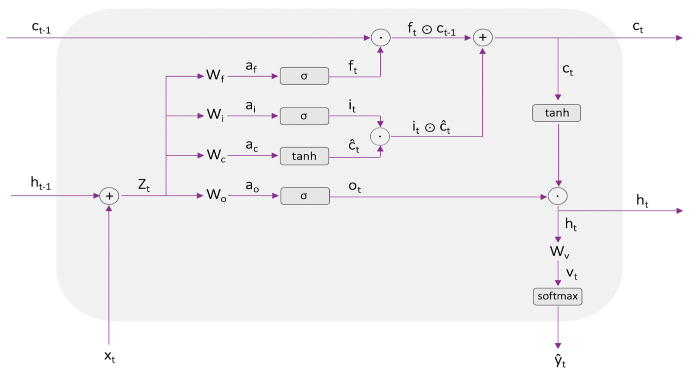


Fig. 5. Forward Propagation of LSTM [8]

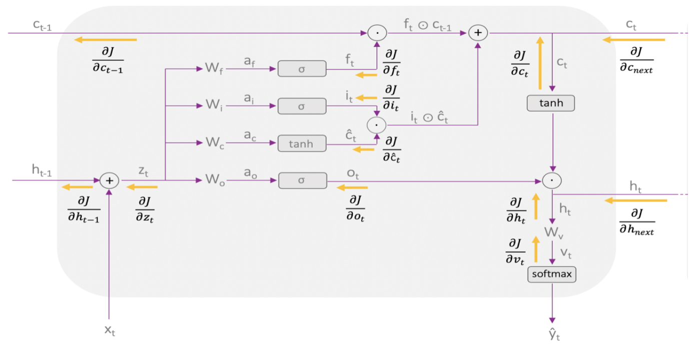
$$h_{t-1} \in \mathbb{R}^{n_h}, \qquad x_t \in \mathbb{R}^{n_x}$$
$$z_t = [h_{t-1}, x_t]$$

$$a_f = W_f \cdot z_t + b_f, \qquad f_t = \sigma(a_f)$$
$$a_i = W_i \cdot z_t + b_i, \qquad i_t = \sigma(a_i)$$
$$a_o = W_o \cdot z_t + b_o, \qquad o_t = \sigma(a_o)$$
$$a_c = W_c \cdot z_t + b_c, \qquad \hat{c}_t = tanh(a_c)$$

$$c_t = i_t \odot \hat{c}_t + f_t \odot c_{t-1}$$
$$h_t = o_t \odot tanh(c_t)$$

$$v_t = W_v \cdot h_t + b_v$$
$$\hat{y}_t = softmax(v_t)$$

## *D. Backward Propagation of LSTM:*



$c_{t-1}$
$f_t \odot c_{t-1}$
$c_t$
$\frac{\partial J}{\partial c_{t-1}}$
$\frac{\partial J}{\partial f_t}$
$\frac{\partial J}{\partial c_t}$
$\frac{\partial J}{\partial c_{next}}$
$W_f$
$a_f$
σ
$f_t$
$W_i$
$a_i$
$i_t$
$\frac{\partial J}{\partial i_t}$
$i_t \odot \hat{c}_t$
tanh
$W_c$
$a_c$
$\hat{c}_t$
$\frac{\partial J}{\partial \hat{c}_t}$
$h_{t-1}$
$z_t$
$W_o$
$a_o$
$o_t$
$h_t$
$\frac{\partial J}{\partial h_{t-1}}$
$\frac{\partial J}{\partial z_t}$
$\frac{\partial J}{\partial o_t}$
$\frac{\partial J}{\partial h_t}$
$\frac{\partial J}{\partial h_{next}}$
$W_v$
$v_t$
$\frac{\partial J}{\partial v_t}$
softmax
$x_t$
$\hat{y}_t$

Fig. 6. Backward Propagation of LSTM [8]

*output*

$$\frac{\partial J}{\partial v_t} = \hat{y}_t - y_t$$

$$\frac{\partial J}{\partial W_v} = \frac{\partial J}{\partial v_t} \cdot \frac{\partial v_t}{\partial W_v} \Rightarrow \frac{\partial J}{\partial W_v} = \frac{\partial J}{\partial v_t} \cdot h_t^T$$

$$\frac{\partial J}{\partial b_v} = \frac{\partial J}{\partial v_t} \cdot \frac{\partial v_t}{\partial b_v} \Rightarrow \frac{\partial J}{\partial b_v} = \frac{\partial J}{\partial v_t}$$

*hidden state*

$$\frac{\partial J}{\partial h_t} = \frac{\partial J}{\partial v_t} \cdot \frac{\partial v_t}{\partial h_t} \Rightarrow \frac{\partial J}{\partial h_t} = W_v^T \cdot \frac{\partial J}{\partial v_t}$$

$$\frac{\partial J}{\partial h_t} += \frac{\partial J}{\partial h_{next}}$$

*output gate*

$$\frac{\partial J}{\partial o_t} = \frac{\partial J}{\partial h_t} \cdot \frac{\partial h_t}{\partial o_t} \Rightarrow \frac{\partial J}{\partial o_t} = \frac{\partial J}{\partial h_t} \odot tanh(c_t)$$

$$\frac{\partial J}{\partial a_o} = \frac{\partial J}{\partial o_t} \cdot \frac{\partial o_t}{\partial a_o} \Rightarrow \frac{\partial J}{\partial a_o} = \frac{\partial J}{\partial h_t} \odot tanh(c_t) \odot \frac{d(\sigma(a_o))}{da_o}$$
$$\Rightarrow \frac{\partial J}{\partial a_o} = \frac{\partial J}{\partial h_t} \odot tanh(c_t) \odot \sigma(a_o)(1 - \sigma(a_o))$$
$$\Rightarrow \frac{\partial J}{\partial a_o} = \frac{\partial J}{\partial h_t} \odot tanh(c_t) \odot o_t(1 - o_t)$$

$$\frac{\partial J}{\partial W_o} = \frac{\partial J}{\partial a_o} \cdot \frac{\partial a_o}{\partial W_o} \Rightarrow \frac{\partial J}{\partial W_o} = \frac{\partial J}{\partial a_o} \cdot z_t^T$$

$$\frac{\partial J}{\partial b_o} = \frac{\partial J}{\partial a_o} \cdot \frac{\partial a_o}{\partial b_o} \Rightarrow \frac{\partial J}{\partial b_o} = \frac{\partial J}{\partial a_o}$$

*input*

$$\frac{\partial J}{\partial z_t} = \frac{\partial J}{\partial a_f}\cdot\frac{\partial a_f}{\partial z_t} + \frac{\partial J}{\partial a_i}\cdot\frac{\partial a_i}{\partial z_t} + \frac{\partial J}{\partial a_o}\cdot\frac{\partial a_o}{\partial z_t} + \frac{\partial J}{\partial a_c}\cdot\frac{\partial a_c}{\partial z_t}$$
$$\Rightarrow \frac{\partial J}{\partial z_t} = W_f^T\cdot\frac{\partial J}{\partial a_f} + W_i^T\cdot\frac{\partial J}{\partial a_i} + W_o^T\cdot\frac{\partial J}{\partial a_o} + W_c^T\cdot\frac{\partial J}{\partial a_c}$$

$$\frac{\partial J}{\partial h_{t-1}} = \frac{\partial J}{\partial z_t}[: n_h, :]$$

$$\frac{\partial J}{\partial c_{t-1}} = \frac{\partial J}{\partial c_t}\cdot\frac{\partial c_t}{\partial c_{t-1}} \Rightarrow \frac{\partial J}{\partial c_{t-1}} = \frac{\partial J}{\partial c_t}\odot f_t$$

*cell state*

$$\frac{\partial J}{\partial c_t} = \frac{\partial J}{\partial h_t}\cdot\frac{\partial h_t}{\partial c_t} \Rightarrow \frac{\partial J}{\partial c_t} = \frac{\partial J}{\partial h_t}\odot o_t\odot(1 - tanh(c_t)^2)$$

$$\frac{\partial J}{\partial c_t} + = \frac{\partial J}{\partial c_{next}}$$

$$\frac{\partial J}{\partial \hat{c}_t} = \frac{\partial J}{\partial c_t}\cdot\frac{\partial c_t}{\partial \hat{c}_t} \Rightarrow \frac{\partial J}{\partial \hat{c}_t} = \frac{\partial J}{\partial c_t}\odot i_t$$

$$\frac{\partial J}{\partial a_c} = \frac{\partial J}{\partial \hat{c}_t}\cdot\frac{\partial \hat{c}_t}{\partial a_c} \Rightarrow \frac{\partial J}{\partial a_c} = \frac{\partial J}{\partial c_t}\odot i_t\odot\frac{d(tanh(a_c))}{da_c}$$
$$\Rightarrow \frac{\partial J}{\partial a_c} = \frac{\partial J}{\partial c_t}\odot i_t\odot(1 - tanh(a_c)^2)$$
$$\Rightarrow \frac{\partial J}{\partial a_c} = \frac{\partial J}{\partial c_t}\odot i_t\odot(1 - \hat{c}_t^2)$$

$$\frac{\partial J}{\partial W_c} = \frac{\partial J}{\partial a_c}\cdot\frac{\partial a_c}{\partial W_c} \Rightarrow \frac{\partial J}{\partial W_c} = \frac{\partial J}{\partial a_c}\cdot z_t^T$$

$$\frac{\partial J}{\partial b_c} = \frac{\partial J}{\partial a_c}\cdot\frac{\partial a_c}{\partial b_c} \Rightarrow \frac{\partial J}{\partial b_c} = \frac{\partial J}{\partial a_c}$$

*input gate*

$$\frac{\partial J}{\partial i_t} = \frac{\partial J}{\partial c_t}\cdot\frac{\partial c_t}{\partial i_t} \Rightarrow \frac{\partial J}{\partial i_t} = \frac{\partial J}{\partial c_t}\odot\hat{c}_t$$

$$\frac{\partial J}{\partial a_i} = \frac{\partial J}{\partial i_t}\cdot\frac{\partial i_t}{\partial a_i} \Rightarrow \frac{\partial J}{\partial a_i} = \frac{\partial J}{\partial c_t}\odot\hat{c}_t\odot\frac{d(\sigma(a_i))}{da_i}$$
$$\Rightarrow \frac{\partial J}{\partial a_i} = \frac{\partial J}{\partial c_t}\odot\hat{c}_t\odot\sigma(a_i)(1 - \sigma(a_i))$$
$$\Rightarrow \frac{\partial J}{\partial a_i} = \frac{\partial J}{\partial c_t}\odot\hat{c}_t\odot i_t(1 - i_t)$$

$$\frac{\partial J}{\partial W_i} = \frac{\partial J}{\partial a_i}\cdot\frac{\partial a_i}{\partial W_i} \Rightarrow \frac{\partial J}{\partial W_i} = \frac{\partial J}{\partial a_i}\cdot z_t^T$$

$$\frac{\partial J}{\partial b_i} = \frac{\partial J}{\partial a_i}\cdot\frac{\partial a_i}{\partial b_i} \Rightarrow \frac{\partial J}{\partial b_i} = \frac{\partial J}{\partial a_i}$$

*forget gate*

$$\frac{\partial J}{\partial f_t} = \frac{\partial J}{\partial c_t}\cdot\frac{\partial c_t}{\partial f_t} \Rightarrow \frac{\partial J}{\partial f_t} = \frac{\partial J}{\partial c_t}\odot c_{t-1}$$

$$\frac{\partial J}{\partial a_f} = \frac{\partial J}{\partial f_t}\cdot\frac{\partial f_t}{\partial a_f} \Rightarrow \frac{\partial J}{\partial a_f} = \frac{\partial J}{\partial c_t}\odot c_{t-1}\odot\frac{d(\sigma(a_f))}{da_f}$$
$$\Rightarrow \frac{\partial J}{\partial a_f} = \frac{\partial J}{\partial c_t}\odot c_{t-1}\odot\sigma(a_f)(1 - \sigma(a_f))$$
$$\Rightarrow \frac{\partial J}{\partial a_f} = \frac{\partial J}{\partial c_t}\odot c_{t-1}\odot f_t(1 - f_t)$$

$$\frac{\partial J}{\partial W_f} = \frac{\partial J}{\partial a_f}\cdot\frac{\partial a_f}{\partial W_f} \Rightarrow \frac{\partial J}{\partial W_f} = \frac{\partial J}{\partial a_f}\cdot z_t^T$$

$$\frac{\partial J}{\partial b_f} = \frac{\partial J}{\partial a_f}\cdot\frac{\partial a_f}{\partial b_f} \Rightarrow \frac{\partial J}{\partial b_f} = \frac{\partial J}{\partial a_f}$$

### *E. Updation of weights*

Forward and backward propagation are calculated for the number of lookback times for each iteration and the weights are updated by calculating gradients of that iteration.

$$\frac{\partial J}{\partial W_f} = \sum_t^T \frac{\partial J}{\partial W_f^t}, \qquad W_f + = \alpha * \frac{\partial J}{\partial W_f}$$

$$\frac{\partial J}{\partial W_i} = \sum_t^T \frac{\partial J}{\partial W_i^t}, \qquad W_i + = \alpha * \frac{\partial J}{\partial W_i}$$

$$\frac{\partial J}{\partial W_o} = \sum_t^T \frac{\partial J}{\partial W_o^t}, \qquad W_o + = \alpha * \frac{\partial J}{\partial W_o}$$

$$\frac{\partial J}{\partial W_c} = \sum_t^T \frac{\partial J}{\partial W_c^t}, \qquad W_c + = \alpha * \frac{\partial J}{\partial W_c}$$

$$\frac{\partial J}{\partial W_v} = \sum_t^T \frac{\partial J}{\partial W_v^t}, \qquad W_v + = \alpha * \frac{\partial J}{\partial W_v}$$

## IV. Results and Analysis

Out of all the experiments we have done on our model, for experiment 5 we got lower MSE for both training and test data. We visualized the MSE of both training and test data. If you see MSE for test data in Fig.8 Sometimes MSE becomes higher than the average, this may be because of the season change, or chaotic weather conditions like cyclones near the region where we recorded the temperature might have effect.

TABLE I
Log table

| Experiment Number | Parameters Chosen | Result |
|---|---|---|
| 1 | Learning Rate: 0.1<br>Iteration: 1<br>Regularization Paramter 1: 0.9<br>Error Function :MSE<br>Regularization Paramter 2: 0.99 | Train/Test Split = 80:20<br><br>Dataset Size: 420124<br>MSE after training: 0.17608317<br>MSE after Testing: 0.151413901 |
| 2 | Learning Rate: 0.1<br>Iteration: 5<br>Regularization Paramter 1: 0.9<br>Error Function :MSE<br>Regularization Paramter 2: 0.99 | Train/Test Split = 80:20<br><br>Dataset Size: 210062<br>MSE after training: 0.28841110<br>MSE after Testing: 0.25883431 |
| 3 | Learning Rate: 0.05<br>Iteration: 10<br>Regularization Paramter 1: 0.9<br>Error Function :MSE<br>Regularization Paramter 2: 0.99 | Train/Test Split = 80:20<br><br>Dataset Size: 84024<br>MSE after training: 0.11638339<br>MSE after Testing: 0.088898 |
| 4 | Learning Rate: 0.1<br>Iteration: 3<br>Regularization Paramter 1: 0.9<br>Error Function :MSE<br>Regularization Paramter 2: 0.99 | Train/Test Split = 80:20<br><br>Dataset Size: 252074<br>MSE after training: 0.227143<br>MSE after Testing: 0.171210 |
| 5 | Learning Rate: 0.07<br>Iteration: 1<br>Regularization Paramter 1: 0.9<br>Error Function :MSE<br>Regularization Paramter 2: 0.99 | Train/Test Split = 80:20<br><br>Dataset Size: 420124<br>MSE after training: 0.10365<br>MSE after Testing: 0.08635 |

## V. Conclusion and future work

Even though we got significantly low MSE from the trials we did with the LSTM, more trials have to be done to check whether we can get MSE lower than what we get now. Also we have to try other neural network architectures like GRU, derivatives of RNN like Vanilla RNN which seems to

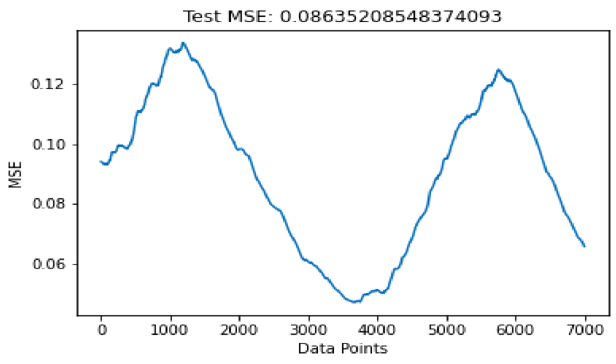


Fig. 7. MSE of test data for experiment 5

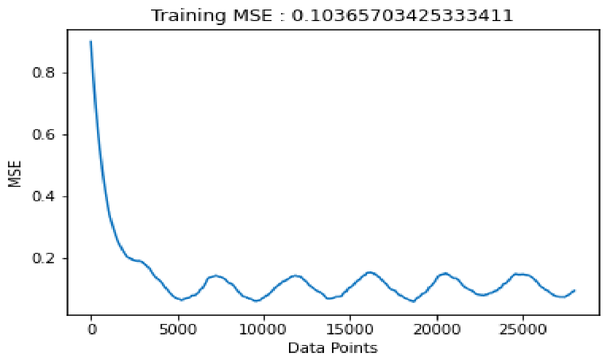


Fig. 8. MSE of train data for experiment 5

minimize the issues with the classical RNN[10], the use of Kalmar filters[9] and and ABBA LSTM [11] which seems to improve LSTM performance to see if even lower MSE achievable.

## References

[1] Weather forcasting <https://en.wikipedia.org/wiki/Weather_forecasting>
[2] The Promise of Recurrent Neural Networks for Time Series Forecasting <https://machinelearningmastery.com/promise-recurrent-neural-networks-time-series-forecasting/>
[3] Hochreiter, Sepp. "The vanishing gradient problem during learning recurrent neural nets and problem solutions." International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6.02 (1998): 107-116.
[4] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.
[5] A Guide to RNN: Understanding Recurrent Neural Networks and LSTM Networks <https://builtin.com/data-science/recurrent-neural-networks-and-lstm>
[6] Understanding LSTMs <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
[7] Illustrated Guide to LSTM and GRU: A step by step explanation <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>
[8] Deriving the backpropagation equations for a LSTM <https://christinakouridi.blog/2019/06/19/backpropagation-lstm/>
[9] Juan Antonio Pérez-Ortiz, Felix A. Gers, Douglas Eck, Jürgen Schmidhuber, "Kalman filters improve LSTM network performance in problems unsolvable by traditional recurrent nets, Neural Networks"
[10] Anyone Can Learn To Code an LSTM-RNN in Python (Part 1:RNN) <https://iamtrask.github.io/2015/11/15/anyone-can-code-lstm/>
[11] Elsworth, Gu ttel. "Time Series Forecasting Using LSTM Networks: A Symbolic Approach"