

Weather Prediction Using Recurrent Neural Networks

Camden Alpert
Computer Science
University of Texas at
Dallas
Richardson, United States
cra200002@utdallas.edu

Qasim Bhutta
Computer Science
University of Texas at
Dallas
Richardson, United States
qhb210000@utdallas.edu

Omar Hussain
Computer Science
University of Texas at
Dallas
Richardson, United States
omh190000@utdallas.edu

Mauricio Rodriguez Rios
Computer Science
University of Texas at
Dallas
Richardson, United States
mrx210005@utdallas.edu

Abstract — Machine Learning and Artificial Intelligence are taking more and more importance in fields of science such as Meteorology. This implementation provides an approach to creating models that are useful to this field using a Recurrent Neural Network. This study focuses specifically on the performance of the “hyperbolic tangent” (or “tanh”) activation function and how it can be used for weather prediction.

Keywords — Neural Networks, Recurrent Neural Networks, Activation Functions, Parameters, Hyperparameters

I. INTRODUCTION AND BACKGROUND

Neural Networks are a specific architecture of Machine Learning models typically used to predict regression data. The general structure of any Neural Network consists of 3 layers. An input, hidden and output layer. Each of these layers consists of nodes (or neurons), where each node is passed a parameter that would correspond to the weight of a feature. Initially, these weights are generated randomly before the iteration of the network. Each of these nodes also creates an output from an activation function. Typical activation functions used are Sigmoid, Rectified Linear Unit (ReLU), and Hyperbolic Tangent(tanh). There is only one input layer and output layer, but it is possible to have

multiple hidden layers. This number of hidden layers is a hyperparameter, which includes other parameters that dictate the flow of data through the network and its structure. A Recurrent Neural Network adds on to the structure of a Neural Network.

This implementation of a Recurrent Neural Network focuses on predicting temperature using a dataset with daily weather data from the Austin, Texas KATT station. It contains data from 02-21-13 to 07-31-2017. The dataset contains data highs, lows and averages for Temperature (Fahrenheit), Dew Point (Fahrenheit), Humidity (as a Percentage), Wind (in MPH), Visibility (in Miles), Sea Level Pressure (in millibars) as well as values for amount of Precipitation (in inches). It should be noted that this dataset contains only one value of the mentioned features per day. Meaning the data average of any piece of data is the average of its low and high. There is no hourly data included with the set.

Our model was implemented using a Jupyter Notebook using Python as the main language. Several libraries were employed for preprocessing and data analysis. None were used in implementing the model itself, however. The scikit-learn library was used to perform the preprocessing of data, we used MinMaxScaler and StandardScaler tools in particular. The pandas library was used to manipulate

our data's handling. Numpy was used while preprocessing and inside the model to perform the correct calculations. Finally, matplotlib was used to create graphs for our data analysis.

The preprocessing of our data took place in several steps. First came The dataset contains a column named "Events", but this was dropped as we considered it was irrelevant towards predicting temperature. The feature "PrecipitationSumInches", which contains the amount of rain for the given days, needed to be modified so that all of the instances of "T" within the column were replaced to 0.001. The "T" represented trace amounts of rain, and we considered it inconsequential to replace this value to 0.001 throughout. The dataset also contained "-" to denote data that was not able to be captured. These were replaced with the value NaN within the data frame. Using the pandas corr function, the correlation of the dataset within its parts was output using matplotlib. The correlation is especially useful to help us determine which features should be passed through the model. Since the highest correlations were noticed on the "TempAvgF", "TempLowF" and "TempHighF" columns, these were selected to possibly be passed through the model. They were then scaled using the Sci-kit Learn library. The dataframe then had to be reshaped using the "sliding window" technique so that it included 7-day sequences, as these 7 days are used to predict the next. The sliding window approach uses a one-day shift, which can be visualized as the first sequence containing {Day 1 – Day 7}, and the second sequence containing {Day 2 – Day 8}, and so on. A total of 1312 such sequences were generated, with the target for the first sequence (Day 1-7), being the temperature on Day 8. The weather data is then split into training, testing and validation sets using a 80/10/10 split.

II. THEORETICAL AND CONCEPTUAL STUDY OF ALGORITHM TO BE IMPLEMENTED

Recurrent Neural Networks (RNNS) are a specialized neural network meant to process sequential data. RNNs retain information about prior inputs through the use of feedback loops which allow them to learn and capture dependencies of the sequential data. As such, RNNs are capable of tasks such as those involving language or time-series predictions, as is the case for our goal of weather prediction using weather series data.

The ability to model time or sequential-based relationships is a key advantage of the RNN model, which is carried out through a recursive update method. At each time step t within the hidden state H_t is calculated as:

$$H_t = \phi_h(X_t W_{xh} + H_{t-1} W_{hh} + b_h)$$

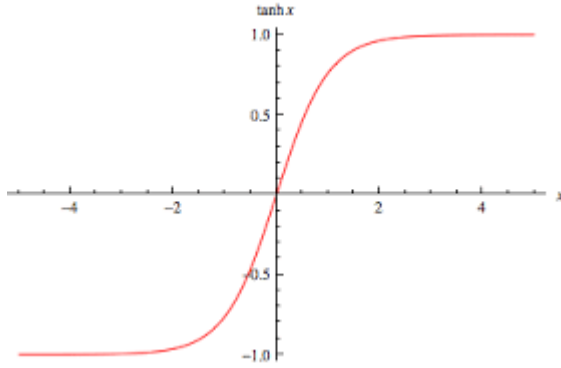
with X_t as the input, W_{xh} and W_{hh} as the weight matrices, b_h as the bias, and ϕ_h as a non-linear activation function, typically tanh or sigmoid. In our case, we made use of the tanh activation function.

While RNNs are capable models of making time-series predictions, RNNs do have their downsides brought by the sequential data. The major issues RNNs suffer from are the vanishing and exploding gradient problems when backpropagating through time over sequences, especially longer form sequences. There exists more advanced algorithms or architectures that address these issues, such as Long Short-Term Memory networks, but our focus remains on RNNs for the course of this project.

As mentioned in the Introduction of this report, RNN's rely on an activation function to properly come to predictions. In this case, the Hyperbolic Tangent (tanh) function (and its derivative) will be used. The tanh function is known for being a better alternative to Sigmoid and ReLU activations due to its more vast use cases. The main characteristic of tanh that separates it from the others is its range

residing in between -1 and 1, giving the model the possibility to cancel out biases in some cases, something which the Sigmoid and ReLU functions can't perform due to being strictly positive functions. As a result, the gradients passed through backpropagation are a lot more balanced, especially when longer dependencies are a quality of the model. It is worth noting that both the Sigmoid and tanh functions are both susceptible to vanishing gradients (when the networks weight become very small, possibly slowing down learning), but tanh is generally not as affected by this phenomenon as frequently as Sigmoid is. ReLU does not suffer from vanishing gradients, but instead from exploding gradients (the opposite of vanishing gradients). This is since the ReLU function is not bounded. Now, this does not mean that the tanh function cannot suffer from exploding gradients as well. Simply, tanh is a more balanced function compared to the other alternatives.

Graph of tanh



Derivative of tanh

$$\begin{aligned}
 \frac{d}{dz} \left(\frac{e^z - e^{-z}}{e^z + e^{-z}} \right) &= \frac{e^z + e^{-z}}{(e^z + e^{-z})^2} d(e^z - e^{-z}) - \frac{e^z - e^{-z}}{(e^z + e^{-z})^2} d(e^z + e^{-z}) \\
 &= \frac{(e^z + e^{-z})(e^z + e^{-z})}{(e^z + e^{-z})^2} - \frac{(e^z - e^{-z})(e^z - e^{-z})}{(e^z + e^{-z})^2} \\
 &= \frac{(e^z + e^{-z})^2 - (e^z - e^{-z})^2}{(e^z + e^{-z})^2} \\
 &= 1 - \left(\frac{e^z - e^{-z}}{e^z + e^{-z}} \right)^2 \\
 &= 1 - \tanh(z)^2
 \end{aligned}$$

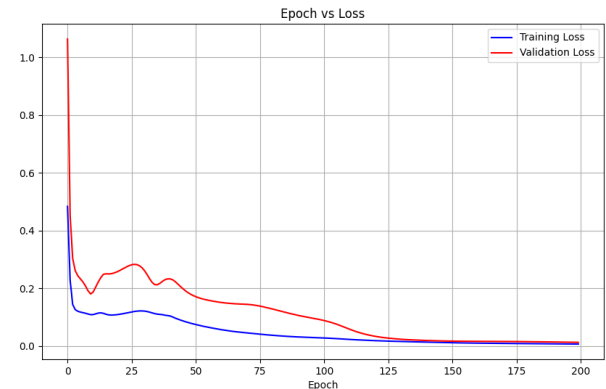
III. RESULTS AND ANALYSIS

When choosing the appropriate hyperparameters for a Recurrent Neural Network, there are several choices to consider. These include the number of epochs, the number of hidden units in the hidden layer, and the learning rate. Selecting optimal hyperparameters is essential to ensure the most accurate model possible.

To get the best hyperparameters for the 3 categories listed above, we used a grid search approach, looping through 27 combinations. For our learning rate, we used the choices: 0.01, 0.001, and 0.0001. For our number of epochs, we used the choices: 50, 100, and 200. For our number of hidden units, we used the choices: 10, 50, and 100.

When testing with a learning rate of 0.01, 50 or 100 hidden units, and epochs 50, 100, and 200, we came across a matrix overflow problem, leading to the model's results becoming flawed. This can be attributed to the high learning rate, which is directly used when calculating new weights and biases. A high learning rate causes excessive weight updates which can lead to matrix overflow as the calculated number may exceed the limit throughout the various matrix multiplications that occur.

After 27 iterations, we concluded that the best combination of hyperparameters is: Learning Rate = 0.001, 200 Epochs, and 100 Hidden Units. This resulted in the lowest test error at a minimal 0.006.

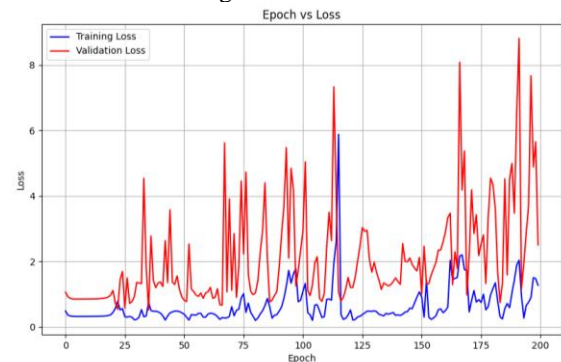


This combination of hyperparameters did not suffer from overfitting, even though it consisted of 200 epochs. As can be seen in the image above, this combination of hyperparameters resulted in a stable loss curve, leveling out around 100 epochs and staying relatively stable, with minute, incremental decreases in loss as it reached 200 epochs. Furthermore, this data had the lowest validation and test error as well, which shows that this hyperparameter combination allows the model to perform well on unseen data. The low loss with this combination can be attributed to the increased number of hidden units which allow the model to effectively learn trends from the data. Likewise, the learning rate of 0.001, is not too low or too high, which allows it to come to an optimal minimum by the 200 epoch mark, while not overshooting the minima due to a high learning rate, and not taking forever to reach the minima due to a very low learning rate.

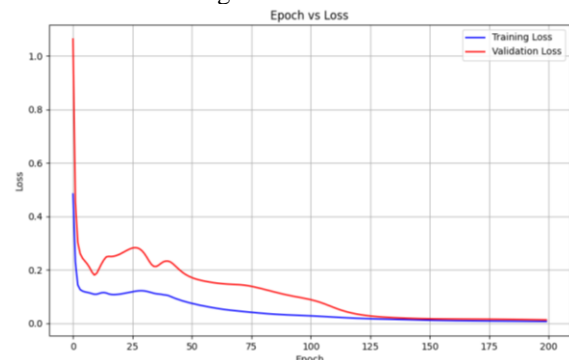
Another point that we noted is the difference in performance when persisting hidden state values (hidden layer unit values) across sequences versus resetting with each new sequence. In the approach that we finalized, we are resetting the hidden state values with each new sequence that is passed in, even though our sequences are closely related, since we are using a sliding window approach, and because one week's weather affects the next week's weather. In theory, it would make sense to persist hidden state values across sequences since the data is so closely related. However, we discovered that resetting the hidden state values led to higher performance across the board with each combination of hyperparameters. Initially confused about this, we discovered that the reason is due to the issue of vanishing/exploding gradients that are common in Recurrent Neural Networks, especially ones that use BackPropagation Through Time (BPTT), instead of Truncated-BackPropagation Through Time (TBPTT). Our model uses BPTT, and our training data consists of 1312 sequences of

7 time-steps each. If we persist hidden state, in one epoch, BPTT would pass through $1312 * 7 = 9,184$ "time-steps". However, when we reset, we only perform BPTT on 7 "time-steps", which prevents the exploding/vanishing gradient problem, leading to higher performance overall.

Persisting Hidden State Values



Resetting Hidden State Values



Both models were evaluated using 0.001 Learning Rate, 100 hidden units, and 200 epochs

IV. CONCLUSION AND FUTURE WORK

We believe that we have done exceptional work in implementing our RNN for weather forecasting. After thorough evaluation, the results that were acquired are very promising for the model's predictive potential. As we continue to work on it, we would like to add more features to predict Average Temperature, such as Dew Point, Humidity, etc. Although our model may not be perfect, we believe that there are several areas of growth which can help fortify the model's performance. One area that can be researched further is how the geographic scope could be

expanded. Our primary focus for weather forecasting is centered on Austin, however, any efforts to advance the model's application in various other locations with distinguished climate patterns would prove to be immensely beneficial.

Additionally, if any further tuning can be done for the model so that it can focus on multiple different geographical regions, then that will enhance its overall performance. Improvements in the model's ability to predict can also be encompassed by using regional climate models and satellite imagery. Upon integrating these data sources, the model will be capable of capturing any sophisticated weather patterns. There can be a more resilient way to monitor the model's performance moving forward. This involves conducting longitudinal studies. Longitudinal studies will help assess the model's accuracy and reliability over a long period of time when it comes to generated weather forecasts. The impact of the model's performance and these studies can be evaluated on particular areas such as agriculture and emergency preparedness. The model will remain highly reliable if it can predict what it takes to make any environment thrive in various scenarios. By embracing these avenues for future success, we believe that the model's forecasting capabilities will be optimized and its effectiveness can be proven in numerous real-world applications such as disaster management and agricultural planning. The efforts of these avenues will provide valuable insights into future weather conditions in the long run. Overall, we believe that this project

doesn't only serve as a building block towards accurate weather forecasting, but it also enforces the essence of continuous improvement in the ever-changing field of meteorology.

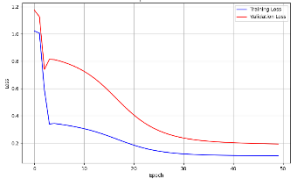
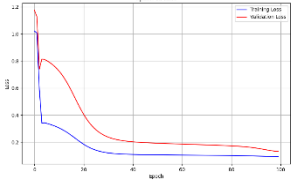
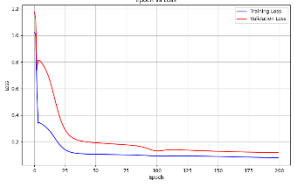
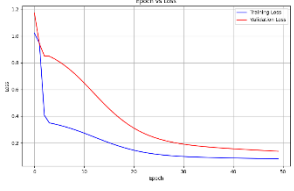
REFERENCES

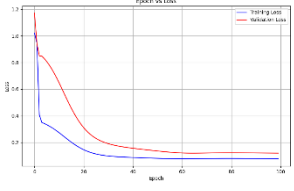
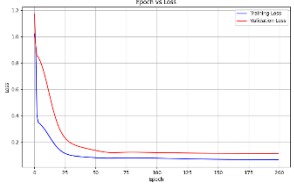
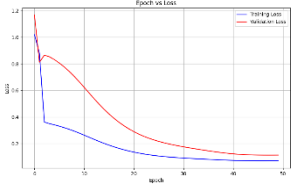
- [1] Dataquest, "RNN From Scratch In Python," *YouTube*, Feb. 06, 2023. <https://www.youtube.com/watch?v=4wuIOcD1LLI> (accessed Nov. 22, 2024).
- [2] R. M. Schmidt, "Recurrent Neural Networks (RNNs): A gentle Introduction and Overview," *arXiv:1912.05911 [cs, stat]*, Nov. 2019. Available: <https://arxiv.org/abs/1912.05911>
- [3] "Weather Prediction using RNN," *kaggle.com*. <https://www.kaggle.com/code/syedali110/weather-prediction-using-rnn>
- [4] S. A. Marhon, C. J. F. Cameron, and S. C. Kremer, "Recurrent Neural Networks," *Intelligent Systems Reference Library*, pp. 29–65, 2013, doi: https://doi.org/10.1007/978-3-642-36657-4_2.
- [5] X. Zhang, C. Zhong, J. Zhang, T. Wang, and W. W. Y. Ng, "Robust recurrent neural networks for time series forecasting," *Neurocomputing*, vol. 526, pp. 143–157, Mar. 2023, doi: <https://doi.org/10.1016/j.neucom.2023.01.037>.
- [6] "Austin Weather," *www.kaggle.com*. <https://www.kaggle.com/datasets/grubenm/austin-weather/data>
- [7] Z. Pei "Derivative of Tanh function," https://blogs.cuit.columbia.edu/zp2130/derivative_of_tanh_function/
- [8] Weisstein, Eric W. "Hyperbolic Tangent." *From MathWorld--A Wolfram Web Resource*. <https://mathworld.wolfram.com/HyperbolicTangent.html>

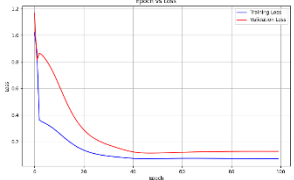
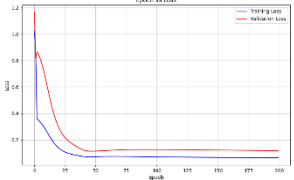
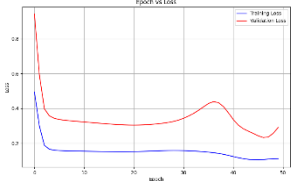
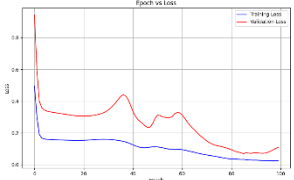
All experiments use 3 Input Units, 1 Hidden Layer, and 1 Output Unit

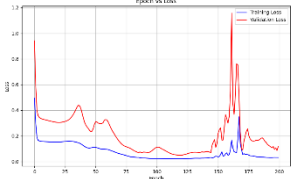
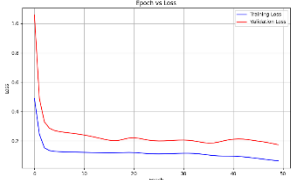
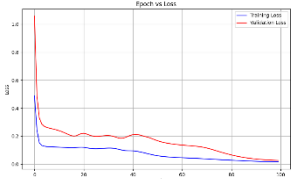
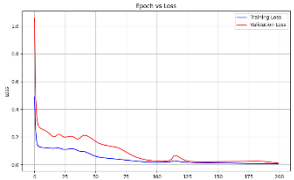
All experiments use TanH Activation and MSE for Loss

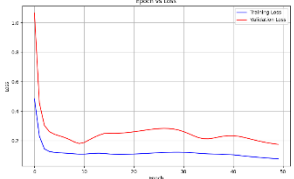
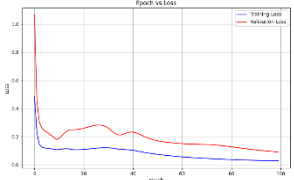
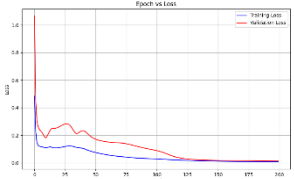
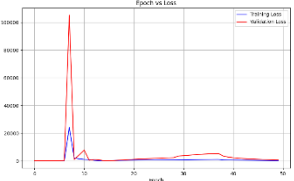
Top 3 Experiment Results: #18, #14, #17

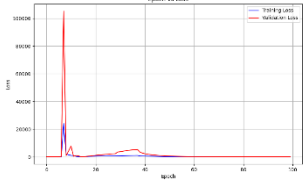
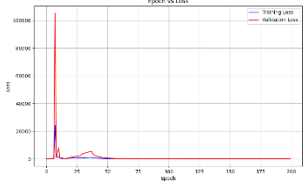
Experiment Number	Parameters Chosen	Results	Train/Validation vs Epoch Graph
1	Recurrent Neural Network: Epochs = 50 Error Function = MSE Activation Function = Tanh Learning Rate = .0001 Hidden Units = 10	Train/Valid/Test Split = 80/10/10 Size of dataset = 1,318 days Training MSE = 0.106 Validation MSE = 0.193 Test MSE = 0.053	 A line graph titled 'Epoch vs Loss' showing Training Loss (blue line) and Validation Loss (red line) over 50 epochs. The Training Loss starts at approximately 1.8 and drops sharply to about 0.3 by epoch 10, then continues to decrease slowly to 0.106 at epoch 50. The Validation Loss starts at approximately 1.8, drops to about 0.8 by epoch 10, and then gradually decreases to 0.193 at epoch 50.
2	Recurrent Neural Network Epochs = 100 Error Function = MSE Activation Function = Tanh Learning Rate = 0.0001 Hidden Units = 10	Train/Valid/Test Split = 80/10/10 Size of dataset = 1,318 days Training MSE = 0.092 Validation MSE = 0.131 Test MSE = 0.044	 A line graph titled 'Epoch vs Loss' showing Training Loss (blue line) and Validation Loss (red line) over 100 epochs. The Training Loss starts at approximately 1.8 and drops sharply to about 0.3 by epoch 10, then continues to decrease slowly to 0.092 at epoch 100. The Validation Loss starts at approximately 1.8, drops to about 0.8 by epoch 10, and then gradually decreases to 0.131 at epoch 100.
3	Recurrent Neural Network Epochs = 200 Error Function = MSE Activation Function = Tanh Learning Rate = 0.0001 Hidden Units = 10	Train/Valid/Test Split = 80/10/10 Size of dataset = 1,318 days Training MSE = 0.079 Validation MSE = 0.118 Test MSE = 0.045	 A line graph titled 'Epoch vs Loss' showing Training Loss (blue line) and Validation Loss (red line) over 200 epochs. The Training Loss starts at approximately 1.8 and drops sharply to about 0.3 by epoch 10, then continues to decrease slowly to 0.079 at epoch 200. The Validation Loss starts at approximately 1.8, drops to about 0.8 by epoch 10, and then gradually decreases to 0.118 at epoch 200.
4	Recurrent Neural Network Epochs = 50 Error Function = MSE Activation Function = Tanh	Train/Valid/Test Split = 80/10/10 Size of dataset = 1,318 days Training MSE = 0.081 Validation MSE = 0.138	 A line graph titled 'Epoch vs Loss' showing Training Loss (blue line) and Validation Loss (red line) over 50 epochs. The Training Loss starts at approximately 1.8 and drops sharply to about 0.3 by epoch 10, then continues to decrease slowly to 0.081 at epoch 50. The Validation Loss starts at approximately 1.8, drops to about 0.8 by epoch 10, and then gradually decreases to 0.138 at epoch 50.

	Learning Rate = 0.0001 Hidden Units = 50	Test MSE = 0.038	
5	Recurrent Neural Network Epochs = 100 Error Function = MSE Activation Function = Tanh Learning Rate = 0.0001 Hidden Units = 50	Train/Valid/Test Split = 80/10/10 Size of dataset = 1,318 days Training MSE = 0.077 Validation MSE = 0.119 Test MSE = 0.044	
6	Recurrent Neural Network Epochs = 200 Error Function = MSE Activation Function = Tanh Learning Rate = 0.0001 Hidden Units = 50	Train/Valid/Test Split = 80/10/10 Size of dataset = 1,318 days Training MSE = 0.064 Validation MSE = 0.112 Test MSE = 0.035	
7	Recurrent Neural Network Epochs = 50 Error Function = MSE Activation Function = Tanh Learning Rate = 0.0001 Hidden Units = 100	Train/Valid/Test Split = 80/10/10 Size of dataset = 1,318 days Training MSE = 0.071 Validation MSE = 0.113 Test MSE = 0.035	

8	<p>Recurrent Neural Network</p> <p>Epochs = 100</p> <p>Error Function = MSE</p> <p>Activation Function = Tanh</p> <p>Learning Rate = 0.0001</p> <p>Hidden Units = 100</p>	<p>Train/Valid/Test Split = 80/10/10</p> <p>Size of dataset = 1,318 days</p> <p>Training MSE = 0.070</p> <p>Validation MSE = 0.126</p> <p>Test MSE = 0.039</p>	
9	<p>Recurrent Neural Network</p> <p>Epochs = 200</p> <p>Error Function = MSE</p> <p>Activation Function = Tanh</p> <p>Learning Rate = 0.0001</p> <p>Hidden Units = 100</p>	<p>Train/Valid/Test Split = 80/10/10</p> <p>Size of dataset = 1,318 days</p> <p>Training MSE = 0.063</p> <p>Validation MSE = 0.117</p> <p>Test MSE = 0.036</p>	
10	<p>Recurrent Neural Network</p> <p>Epochs = 50</p> <p>Error Function = MSE</p> <p>Activation Function = Tanh</p> <p>Learning Rate = 0.001</p> <p>Hidden Units = 10</p>	<p>Train/Valid/Test Split = 80/10/10</p> <p>Size of dataset = 1,318 days</p> <p>Training MSE = 0.112</p> <p>Validation MSE = 0.291</p> <p>Test MSE = 0.078</p>	
11	<p>Recurrent Neural Network</p> <p>Epochs = 100</p> <p>Error Function = MSE</p> <p>Activation Function = Tanh</p> <p>Learning Rate = 0.001</p> <p>Hidden Units = 10</p>	<p>Train/Valid/Test Split = 80/10/10</p> <p>Size of dataset = 1,318 days</p> <p>Training MSE = 0.023</p> <p>Validation MSE = 0.108</p> <p>Test MSE = 0.106</p>	

12	<p>Recurrent Neural Network</p> <p>Epochs = 200</p> <p>Error Function = MSE</p> <p>Activation Function = Tanh</p> <p>Learning Rate = 0.001</p> <p>Hidden Units = 10</p>	<p>Train/Valid/Test Split = 80/10/10</p> <p>Size of dataset = 1,318 days</p> <p>Training MSE = 0.031</p> <p>Validation MSE = 0.117</p> <p>Test MSE = 0.401</p>	
13	<p>Recurrent Neural Network</p> <p>Epochs = 50</p> <p>Error Function = MSE</p> <p>Activation Function = Tanh</p> <p>Learning Rate = 0.001</p> <p>Hidden Units = 50</p>	<p>Train/Valid/Test Split = 80/10/10</p> <p>Size of dataset = 1,318 days</p> <p>Training MSE = 0.063</p> <p>Validation MSE = 0.174</p> <p>Test MSE = 0.049</p>	
14	<p>Recurrent Neural Network</p> <p>Epochs = 100</p> <p>Error Function = MSE</p> <p>Activation Function = Tanh</p> <p>Learning Rate = 0.001</p> <p>Hidden Units = 50</p>	<p>Train/Valid/Test Split = 80/10/10</p> <p>Size of dataset = 1,318 days</p> <p>Training MSE = 0.017</p> <p>Validation MSE = 0.027</p> <p>Test MSE = 0.011</p>	
15	<p>Recurrent Neural Network</p> <p>Epochs = 200</p> <p>Error Function = MSE</p> <p>Activation Function = Tanh</p> <p>Learning Rate = 0.001</p> <p>Hidden Units = 50</p>	<p>Train/Valid/Test Split = 80/10/10</p> <p>Size of dataset = 1,318 days</p> <p>Training MSE = 0.006</p> <p>Validation MSE = 0.012</p> <p>Test MSE = 0.013</p>	

16	<p>Recurrent Neural Network</p> <p>Epochs = 50</p> <p>Error Function = MSE</p> <p>Activation Function = Tanh</p> <p>Learning Rate = 0.001</p> <p>Hidden Units = 100</p>	<p>Train/Valid/Test Split = 80/10/10</p> <p>Size of dataset = 1,318 days</p> <p>Training MSE = 0.076</p> <p>Validation MSE = 0.174</p> <p>Test MSE = 0.062</p>	
17	<p>Recurrent Neural Network</p> <p>Epochs = 100</p> <p>Error Function = MSE</p> <p>Activation Function = Tanh</p> <p>Learning Rate = 0.001</p> <p>Hidden Units = 100</p>	<p>Train/Valid/Test Split = 80/10/10</p> <p>Size of dataset = 1,318 days</p> <p>Training MSE = 0.028</p> <p>Validation MSE = 0.089</p> <p>Test MSE = 0.033</p>	
18	<p>Recurrent Neural Network</p> <p>Epochs = 200</p> <p>Error Function = MSE</p> <p>Activation Function = Tanh</p> <p>Learning Rate = 0.001</p> <p>Hidden Units = 100</p>	<p>Train/Valid/Test Split = 80/10/10</p> <p>Size of dataset = 1,318 days</p> <p>Training MSE = 0.006</p> <p>Validation MSE = 0.012</p> <p>Test MSE = 0.006</p>	
19	<p>Recurrent Neural Network</p> <p>Epochs = 50</p> <p>Error Function = MSE</p> <p>Activation Function = Tanh</p> <p>Learning Rate = 0.01</p> <p>Hidden Units = 10</p>	<p>Train/Valid/Test Split = 80/10/10</p> <p>Size of dataset = 1,318 days</p> <p>Training MSE = 214.490</p> <p>Validation MSE = 664.119</p> <p>Test MSE = 247.987</p>	

20	<p>Recurrent Neural Network</p> <p>Epochs = 100</p> <p>Error Function = MSE</p> <p>Activation Function = Tanh</p> <p>Learning Rate = 0.01</p> <p>Hidden Units = 10</p>	<p>Train/Valid/Test Split = 80/10/10</p> <p>Size of dataset = 1,318 days</p> <p>Training MSE = 1.197</p> <p>Validation MSE = 2.523</p> <p>Test MSE = 1.901</p>	
21	<p>Recurrent Neural Network</p> <p>Epochs = 200</p> <p>Error Function = MSE</p> <p>Activation Function = Tanh</p> <p>Learning Rate = 0.01</p> <p>Hidden Units = 10</p>	<p>Train/Valid/Test Split = 80/10/10</p> <p>Size of dataset = 1,318 days</p> <p>Training MSE = 0.354</p> <p>Validation MSE = 1.025</p> <p>Test MSE = 0.735</p>	
22	<p>Recurrent Neural Network</p> <p>Epochs = 50</p> <p>Error Function = MSE</p> <p>Activation Function = Tanh</p> <p>Learning Rate = 0.01</p> <p>Hidden Units = 50</p>	<p>OVERFLOW IN MATRICES DUE TO HIGH LEARNING RATE</p>	<p>OVERFLOW IN MATRICES DUE TO HIGH LEARNING RATE</p>
23	<p>Recurrent Neural Network</p> <p>Epochs = 100</p> <p>Error Function = MSE</p> <p>Activation Function = Tanh</p> <p>Learning Rate = 0.01</p> <p>Hidden Units = 50</p>	<p>OVERFLOW IN MATRICES DUE TO HIGH LEARNING RATE</p>	<p>OVERFLOW IN MATRICES DUE TO HIGH LEARNING RATE</p>
24	<p>Recurrent Neural Network</p> <p>Epochs = 200</p> <p>Error Function = MSE</p> <p>Activation Function = Tanh</p> <p>Learning Rate = 0.01</p> <p>Hidden Units = 50</p>	<p>OVERFLOW IN MATRICES DUE TO HIGH LEARNING RATE</p>	<p>OVERFLOW IN MATRICES DUE TO HIGH LEARNING RATE</p>

25	Recurrent Neural Network Epochs = 50 Error Function = MSE Activation Function = Tanh@mau Learning Rate = 0.01 Hidden Units = 100	OVERFLOW IN MATRICES DUE TO HIGH LEARNING RATE	OVERFLOW IN MATRICES DUE TO HIGH LEARNING RATE
26	Recurrent Neural Network Epochs = 100 Error Function = MSE Activation Function = Tanh Learning Rate = 0.01 Hidden Units = 100	OVERFLOW IN MATRICES DUE TO HIGH LEARNING RATE	OVERFLOW IN MATRICES DUE TO HIGH LEARNING RATE
27	Recurrent Neural Network Epochs = 200 Error Function = MSE Activation Function = Tanh Learning Rate = 0.01 Hidden Units = 100	OVERFLOW IN MATRICES DUE TO HIGH LEARNING RATE	OVERFLOW IN MATRICES DUE TO HIGH LEARNING RATE