Using a Custom RNN to Predict Dallas Temperature

Khoa Diep
The University of Texas at Dallas
Richardson, TX
ktd170030@utdallas.edu

Alejandro Urquieta
The University of Texas at Dallas
Richardson, TX
asu230001@utdallas.edu

Adithya Viswanathan
The University of Texas at Dallas
Richardson, TX
axv190040@utdallas.edu

Maviya Yaseen
The University of Texas at Dallas
Richardson, TX
mxy200016@utdallas.edu

Emily Chao
The University of Texas at Dallas
Richardson, TX
exc220012@utdallas.edu

Farrel Raja
The University of Texas at Dallas
Richardson, TX
fjr190000@utdallas.edu

***Abstract*— This paper explores the application of a recurrent neural network, or RNN, to predict temperatures in Dallas, Texas using historical hourly weather data. The dataset used includes humidity, pressure, and temperature measurements in Dallas from 2012 to 2017. The RNN architecture, consisting of a unique feedback loop in the hidden layer, was employed in this implementation to better analyze and output sequential data. In addition, this paper also delves into the architectural details of neural networks, the MSE and MAE loss functions, and the use of the backpropagation technique in RNNs. Both forward and backward propagation were used to train the model and it was finally tested on a testing dataset. After a thorough analysis of the RNN weather prediction model, we believe that this study demonstrates its effectiveness in handling time series data.**

I. Introduction

Due to its location between the Rocky Mountains and the Appalachian Mountains, Texas experiences a vast range of weather throughout the year. The cold air flows down from Canada and conflicts with the warm air flowing up from the Gulf of Mexico [1]. This leads to very diverse and ever-changing climates across the large state of Texas, making it challenging to accurately predict the weather on a daily or hourly basis.

A. Problem Definition

To address this issue, we decided to employ a simple recurrent neural network, or simple RNN, as our chosen machine learning technique. RNNs are effective in processing sequential and time series data because of their ability to remember past information. Time series data is chronologically ordered and indexed by time. This includes data like weather records, patient health evolution metrics, or economic indicators.

To predict sequential weather patterns, RNNs can use past data to forecast upcoming weather. Unlike other neural networks that use forward propagation to process their inputs, RNNs retain information about past inputs and apply this information to accurately determine the output, in this case, the weather prediction, of the network.

The RNN receives input weather data. The input is sent to the hidden state, which is where the network's memory of the sequential data is stored and processed. After it processes that data, it will be used to generate the forecast based on historical data which will then be outputted to the user in the form of the results.

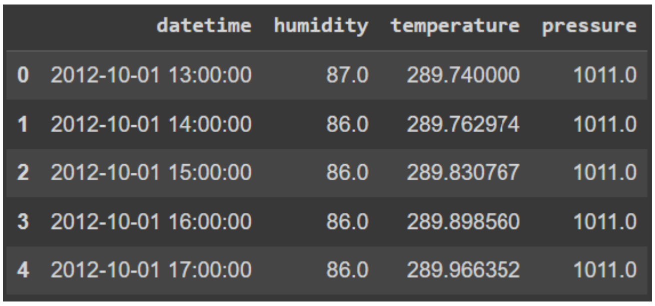
II. Dataset

A. Historical Hourly Weather Data 2012-2017

For this project, we decided to adapt the *Historical Hourly Weather Data 2012-2017* dataset, which is a set of different historical weather data collected between 2012-2017 for 30 US and Canadian Cities, and 6 Israeli Cities. This dataset consists of hourly measurements of humidity, pressure, temperature, weather description, wind direction, wind speed, and city-specific attributes. Each attribute was separated into its own CSV file [3].

B. Cleaned Dataset

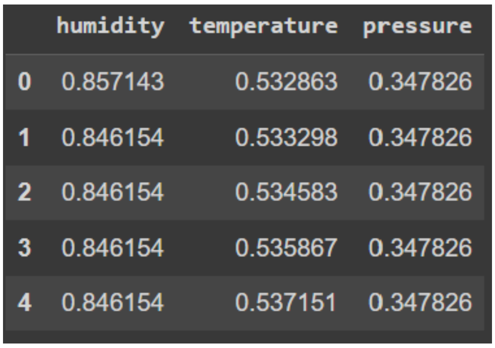
To create our cleaned dataset, we preprocessed the individual datasets of humidity, pressure, and temperature for Dallas. Empty values were interpolated by using the average between their previous and next values.



	datetime	humidity	temperature	pressure
0	2012-10-01 13:00:00	87.0	289.740000	1011.0
1	2012-10-01 14:00:00	86.0	289.762974	1011.0
2	2012-10-01 15:00:00	86.0	289.830767	1011.0
3	2012-10-01 16:00:00	86.0	289.898560	1011.0
4	2012-10-01 17:00:00	86.0	289.966352	1011.0

Fig. 1. Pre-Normalized Combined Data

Then, the three individual datasets were combined and normalized using min-max scaling to mitigate varying scales that could occur if used by our RNN model. The resulting normalized dataset contained 45,252 tuples of humidity, pressure, and temperature attributes.



	humidity	temperature	pressure
0	0.857143	0.532863	0.347826
1	0.846154	0.533298	0.347826
2	0.846154	0.534583	0.347826
3	0.846154	0.535867	0.347826
4	0.846154	0.537151	0.347826

Fig. 2. Post-normalized combined data

We then explored the frequency distribution of the three attributes to understand the shape and the general tendencies of the data.

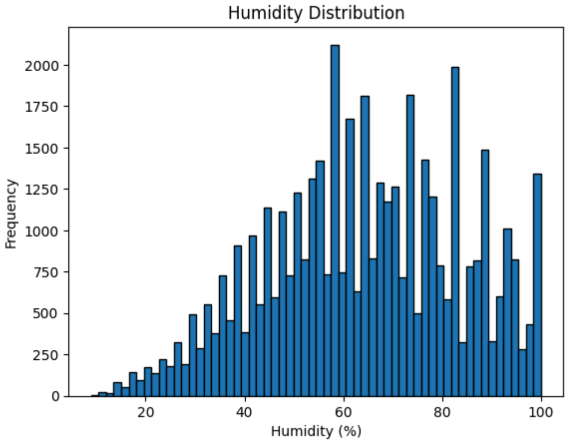


Fig. 3. Humidity Distribution

For humidity, the data follows a comb distribution, with high-frequency spikes occurring at varying levels of humidity. The highest spike occurs at around 58% humidity.

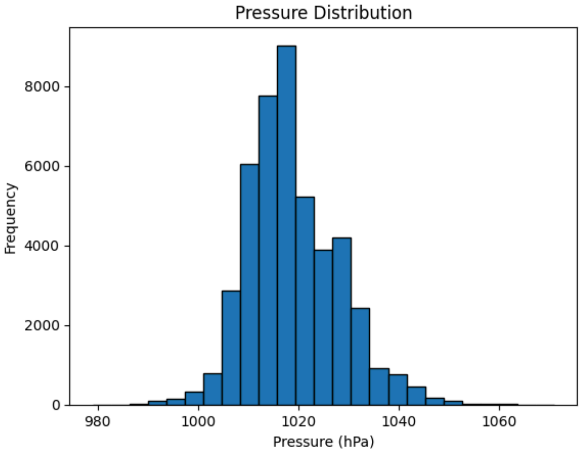


Fig. 4. Pressure Distribution

Pressure has the most normal distribution, with the data roughly following a general bell curve. The highest peak frequency occurs at around 1020 hPa, slightly off-center.

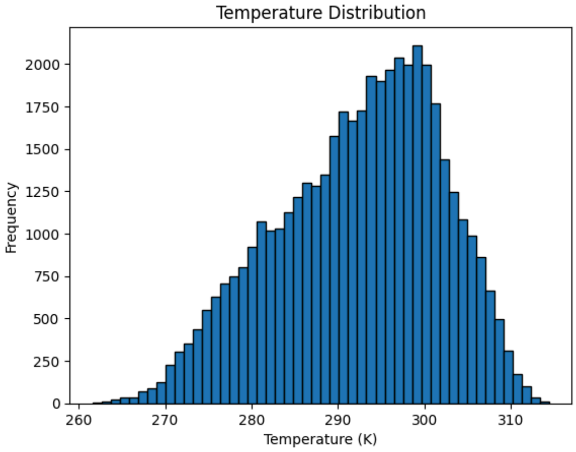


Fig. 5. Temperature Distribution

The temperature distribution is left-skewed, with 290 K to 305 K having the highest frequency peak.

C. Time Sequences

The data is then broken into time sequences of size N (default size of 10), thus each time sequence has N time steps. These time sequences store the last N tuples of information from humidity, temperature, and pressure, which will be fed into the RNN as a single training instance.

D. Training and Testing Splits

These time sequences are then split into training and testing sets. Specifically, the first 20% of the sequence will be the training instances (X_train), and the first 20% of the temperature from the cleaned data will be the training values (y_train). The last 80% will be the testing instances (X_test), and the first 20% of the temperature from the cleaned data will be the testing values (y_test). The exact percentages can be adjusted as a parameter in the driver.

III. Study of Recurrent Neural Networks

A. Structure of Artificial Neural Networks

An Artificial Neural Network (ANN) is a machine learning model consisting of individual nodes, also called neurons. These neurons are organized into different layers. Typically, ANNs consist of an input layer, one or more hidden layers, and a final output layer, in that order. Artificial neurons receive weighted signals from connected neurons from the previous layer.

In addition to the weights, each neuron usually has a bias term. These biases allow the model to capture the offsets of the input data.

To calculate the value of each neuron, an activation function is applied to the sum of its previous weighted signals and its bias. These activation functions introduce some nonlinearity into the network, which allows the model to capture more complicated relationships. The activation functions utilized are tanh, sigmoid, and ReLU.

$$tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{1}$$

$$sigmoid(x) = \sigma(x) = \frac{1}{1+e^{-x}} \tag{2}$$

$$ReLU(x) = max(0, x) \tag{3}$$

A forward pass through all neurons for each layer is called forward propagation. This results in an output per forward propagation.

B. Loss Functions

To evaluate the performance of a machine model, a loss function, also referred to as an error function, is applied. This function quantifies the difference between the predicted output of the model, denoted as ŷ, and the actual output from the data, denoted as y.

Some loss functions applied to a regression problem, such as our own, are Mean Square Error (MSE) and Mean Absolute Error (MAE). MAE is also known as L1 loss and MSE is also known as L2 loss [4].

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{4}$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{5}$$

C. Backpropagation of ANNs

To train a neural network, including RNNs, the backward propagation algorithm called backpropagation is applied to adjust the weights and biases of each layer. Gradient descent, an algorithm to find the local minimum of a differential function, is utilized to minimize the loss function of an ANN. The idea of gradient descent is to iteratively step in the opposite direction of the gradient until reaching and settling at the local minimum [5].

In the context of ANNs (and thus RNNs), if forward propagation goes through the model in one way, backward propagation proceeds through the model oppositely. The gradient of the last layer is calculated first, and the gradient of the first layer is calculated last. The backward flow is dictated by the chain rule from gradient calculus since some computations of gradients in a layer are reused in the previous layer.

The derivative of the applied activation function is needed to calculate the gradient.

$$\frac{d}{dx}\,tanh(x) = 1 - tanh^2(x) \tag{6}$$

$$\frac{d}{dx}\sigma(x) = 1 - \sigma(x) \tag{7}$$

$$\frac{d}{dx}\mathrm{ReLU}(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

During backward propagation, adjustments to the weights and biases are made proportional to its gradient descent and learning rate (a hyperparameter that linearly scales the gradient). The learning rate is denoted by η.

An epoch is the number of times a neural network will train over the entire training data. Forward propagation and backward propagation will be run across the entire data once per epoch. Thus, each sample from the training data influences the model.

D. Architecture of Recurrent Neural Networks

RNNs are a type of ANN, distinguished by a unique hidden layer that incorporates a feedback loop. This layer is called a recurrent hidden layer. The recurrent hidden layer's architecture enables it to retain information about its input sequences.

This is achieved by the hidden layer's nodes having their outputs influence its next subsequent inputs to the same nodes, creating a feedback loop.

Each node within the recurrent hidden layer uses its current weighted inputs, weighted memory, bias, and activation function to compute its neuron values. This process repeats for each tuple from the time sequence.

E. Unrolling an RNN

To incorporate forward propagation and backward propagation for an RNN, the networks need to be unrolled (or unfolded). Unrolling is a process of expanding the complete neural network for the entire sequence. Effectively, this creates a deep neural network with each layer representing a corresponding time step.

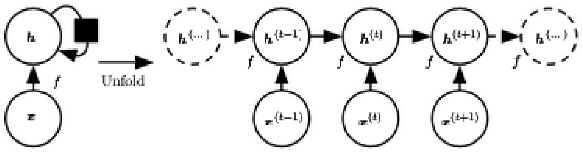


Fig. 6. Unrolling an RNN [6]

For each time step in forward propagation, the RNN processes the current input from the time step to calculate the output and update the hidden state and memory. This process is repeated for each time step in the sequence.

Backward propagation in RNNs involves calculating gradients, discussed extensively earlier in this section, in the same manner as ANNs. For RNNs, this process is called Backward Propagation Through Time (BPTT).

BPTT unfolds the network in the opposite direction as forward propagation. Similar to forward propagation, each time step corresponds to a cell of the RNN. The key difference is that the gradients are computed with respect to the loss function. These gradients are stepped backward in time from the final time step to the first time step [6][7].

IV. Implementing a Recurrent Neural Network

A. Application of RNNs to the Data

An RNN was created from scratch with an input layer the size of the input tuples. The next layer is a recurrent hidden layer with an adjustable size. The final output layer would be the predicted, normalized temperature.

For the time series sequence of size N, the network will be unrolled into an N+2 layered neural network, with one input layer, and N hidden layers, one layer for each time sequence.

An activation function (tanh, sigmoid, or ReLU) is applied to the recurrent layer. A linear activation function is applied to the output. The output layer will predict the temperature.

Training the custom recurrent neural network requires forward propagation to calculate loss and backward propagation on each sequence and the corresponding value of the training dataset to adjust the weights and biases.

To test the model, forward propagation is applied to the testing dataset to calculate the total loss.

B. *Model Parameters*

The three input parameters that we have chosen are temperature, pressure, and humidity. These values could help us predict the most accurate weather conditions. Temperature is important as a lot of weather conditions depend on it. For example, if it is below freezing, there could be a chance of snow, sleet, or black ice. Pressure can affect rainfall and cloud formations, as winds “blow towards the low pressure and the air rises in the atmosphere”. When the air rises, water vapor condenses, “forming clouds” and some sort of precipitation usually happens [8]. Humidity is a good indicator of whether or not it will rain. Places with high humidity have more precipitation than places that do not. We also decided to randomize the training data to prevent order bias, which could cause our RNN model to learn any unintentional patterns related to the order in which data was originally presented and improve generalization for a more robust model.

Our output parameter is the prediction variable, temperature.

C. *Hyperparameters*

The following hyperparameters that we included are different activation functions, loss functions, learning rates, number of epochs, and size of time sequences. In addition, we explored the dimensions of our RNN and considered the size of the input and hidden. All of these hyperparameters can be adjusted within the driver function.

D. *R-squared*

R^2 is an evaluation metric that quantifies how well the independent variable explains the variation of the dependent variable. It ranges from 0 to 1, where 1 indicates a perfect fit, and 0 represents that none of the variation of the dependent variables is determined by the independent variables [9].

$$R^2 = 1 - \frac{RSS}{TSS} \tag{9}$$

$$\text{RSS} = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{10}$$

$$\text{TSS} = \sum_{i=1}^{N} (y_i - \bar{y}_i)^2 \tag{11}$$

V. Results and Analysis

A. *Selection of Temperature as Sole Input*

After manually assessing the impact of temperature, humidity, and pressure on the performance of the model, we found that temperature was the most influential. Therefore, to keep the model simplified while preserving performance, we decided to use temperature data as the only input.

B. *Activation Function Performance*

To optimize the performance of our RNN, we tested multiple activation functions: sigmoid, tanh, and ReLu, and measured the resulting MSE from each across multiple learning rates (0.015, 0.001, 0.01) and hidden layer sizes (5, 20, 50).

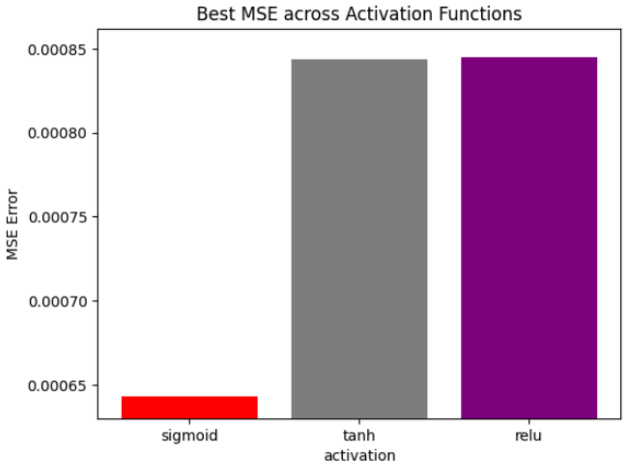


Fig. 7. Best MSE for Activation Functions

In Fig. 7., the best MSE across experiments is displayed. The sigmoid activation function performed best overall, with the minimum MSE being 0.00064. Tanh and ReLu are formidable activation functions, but might not give results due to their susceptibility to the vanishing gradient problem and dying ReLu problem, respectively. The main reason for the sigmoid’s superior performance relative to other activations since its range is (0,1), which is also the range of our normalized data. Since sigmoid matches exactly with the range of our data, it gives better results than other activation functions by a significant margin.

C. *Learning Rate Performance*

We also experimented with different learning rates and looked at the minimum MSE that resulted from each.

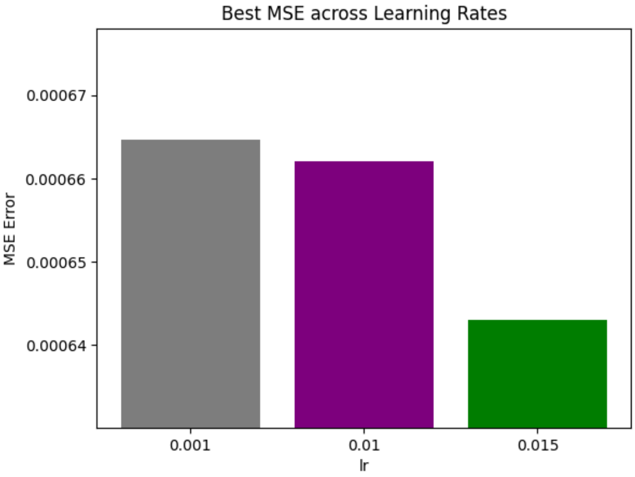


Fig. 8. Best MSE for Learning Rates

Fig. 8. shows that the highest learning rate, 0.015 performed the best. This is due most likely to the longer times required for smaller learning rates to converge and the number of epochs used during training (10 epochs).

We chose to not go any lower than 0.001 as too low of a learning rate can slow the training down excessively and can cause the model to get stuck at a local minimum, greatly affecting model performance.

We did not go any higher than 0.015 as the higher the learning rate, the greater the chance of missing the minima and causing an exploding gradient. Overall, the errors are very similar, but this hyper-parameter tuning helped discover important shifts in performance nevertheless.

D. *Hidden Layer Size Performance*

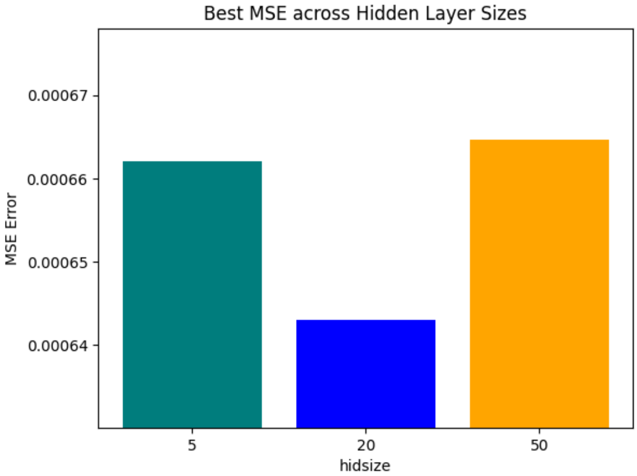


Fig. 9. Best MSE for Hidden Layer Size

In Fig. 9., the best MSE across experiments for different hidden layer sizes is shown. A hidden layer size of 20 is the best-performing metric, with the minimum MSE being around 0.00064. A hidden layer size of 5 likely led to a very simple model which caused underfitting. This would lead to a worse test performance since the model would not be able to understand the underlying patterns of the unseen test data. On the other hand, a hidden layer size of 50 likely led to an overly complex model, which caused overfitting. This would also lead to a worse test performance since it would lead to incorrect conclusions on train data and thus perform incorrectly on test data, which may or may not represent the distribution of data observed in the test data. Overall, the errors are very similar, but this hyper-parameter tuning helped discover important shifts in performance nevertheless.

E. *Model's Accuracy Performance*

In Fig. 10., we can better observe that our RNN's normalized predictions followed very closely the actual normalized temperatures throughout most of the sequence, with a testing MSE loss of 0.00121. We also observed an MAE error of 0.0205. In addition to these metrics, we also calculated an R^2 score of 0.889. All of these metrics are good indicators of performance and show that the model can capture the patterns in the data. The R^2 score demonstrates that the RNN can model a large proportion of the dataset's variance. However, in certain places, for instance, between the 900 and 1000 time sequences, there's a slight divergence where the actual values showed a slight drop.

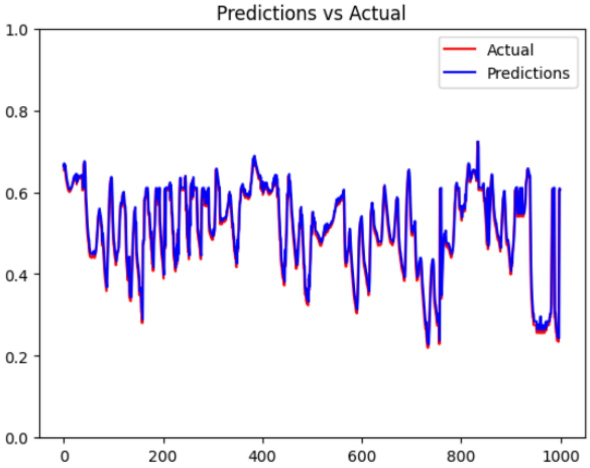


Fig. 10. Predicted vs. Actual Temperatures Comparison

Based on our results, we determined that our RNN with a learning rate of 0.015, sigmoid activation function, and hidden layer size of 20 served as the best model for temperature prediction.

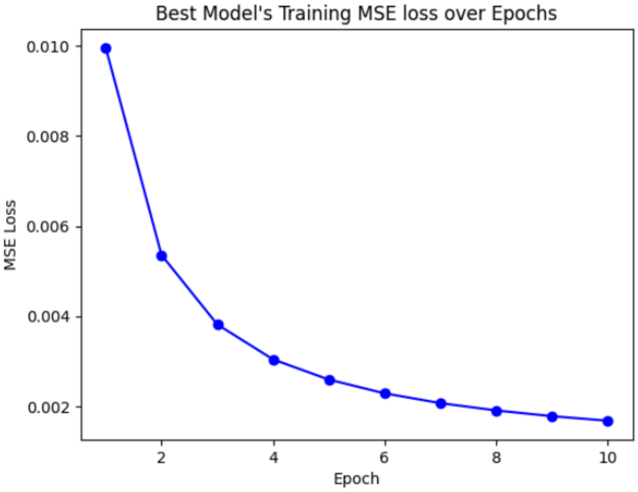


Fig. 11. MSE Loss vs Epochs for Best Model

VI. Conclusion and Future work

In conclusion, we found that using an RNN model for data prediction was highly accurate and effective. With a time sequence of the temperature of the last N hours, the RNN is effective in predicting the temperature of the next hour.

In the future, we may also consider converting the simple RNN into an LSTM. Simple RNNs have an issue known as the vanishing gradient. The vanishing gradient problem results in slow convergence or stagnation in the learning of the model. LSTMs deal with this issue using a more complex gating mechanism. This allows the model to generate the final results quicker, as it reduces the convergence rate. The LSTM model can also learn long-term dependencies more effectively due to its ability to mitigate the vanishing gradient problem.

Alternatively, integrating attention mechanisms into the RNN would allow the model to focus on more relevant parts of the input. Our dataset consisted of three parameters. However, they were not equally important. By using attention mechanisms, the model can prioritize important parameters, like temperature, over the other parameters. This would allow the model to improve its predictive capabilities and give more accurate results.

References

[1] A. Plasencia and J. Hawila, “Why is Texas weather so crazy?,” wfaa.com, Feb. 21, 2022. Available: https://www.wfaa.com/article/weather/why-is-texas-weather-so-crazy/287-9ec52d7c-8c77-4a64-baaf-e9ae1b7e0d90. [Accessed: May 06, 2024]

[2] K. Yao, B. Peng, Y. Zhang, D. Yu, G. Zweig and Y. Shi, "Spoken language understanding using long short-term memory neural networks," 2014 IEEE Spoken Language Technology Workshop (SLT), South Lake Tahoe, NV, USA, 2014, pp. 189-194, doi: 10.1109/SLT.2014.7078572. keywords: {Logic gates; Recurrent neural networks; Training; Vectors; Semantics; Speech; Recurrent neural networks; long short-term memory; language understanding},

[3] D. Beniaguev, “Historical hourly weather data 2012-2017,” Kaggle, https://www.kaggle.com/datasets/selfishgene/historical-hourly-weather-data [Accessed: May 4, 2024].

[4] A. Richmond, “Loss functions in machine learning explained.” *DataCamp*, www.datacamp.com/tutorial/loss-function-in-machine-learning [Accessed: May 4, 2024].

[5] A. Nagar, “Neural Networks” (class lecture, Machine Learning, University of Texas at Dallas, Richardson, March 6, 2024.

[6] Goodfellow, I., Bengio, Y., & Courville, A. (2016). “Deep learning” MIT Press, http://www.deeplearningbook.org [Accessed: May 4, 2024]

[7] Nabi, J. (2021, December 10). Recurrent Neural Networks (RNNs) - towards data science. *Medium*. https://towardsdatascience.com/recurrent-neural-networks-rnns-3f06d7653a85

[8] UCAR, “The Highs and Lows of Air Pressure | UCAR Center for Science Education,” *scied.ucar.edu*. https://scied.ucar.edu/learning-zone/how-weather-works/highs-and-lows-air-pressure#:~:text=A%20low%20pressure%20system%20has

[9] Numeracy, Maths, and Statistics - Academic Skills kit. https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/regression-and-correlation/coefficient-of-determination-r-squared.html