Applying Recurrent Neural Networks for Weather feature predictions

Praveen Aravendan
Department of Computer Science and Engineering
The University of Texas at Dallas
Richardson, USA

Obuli Vignesh Rangasamy
Department of Computer Science and Engineering
The University of Texas at Dallas
Richardson, USA

Varsha Samararatne
Department of Computer Science and Engineering
The University of Texas at Dallas
Richardson, USA

Gagandeep Singh Jossan
Department of Computer Science and Engineering
The University of Texas at Dallas
Richardson, USA

***Abstract*—Neural networks have had remarkable success in solving a wide variety of problems. From face recognition to self-driving cars, they are the essence of supervised and unsupervised deep learning. There are many cases where data naturally forms sequences, and in these instances content and order are important. When learning from sequential data, memory becomes essential for processing a sequence of related data in an ordered context. This is where Recurrent Neural Networks (RNN) take center stage. RNNs have 'memory'. They remember the past, and the decisions they take are influenced by what has been learned in the past. However, the challenge is that short-term memory is fundamentally limited because of the vanishing-gradient problem which makes the memory of vanilla RNNs very short indeed. The solution is a very special kind of RNN, Long Short-Term Memory (LSTM). An LSTM is capable of learning from relatively long-term dependencies and is ideal for dealing with time series data such as weather. Thus, in this project, we used an LSTM to predict the temperature and humidity of three major cities in Texas. The temperature model for each of the three cities achieved higher percentages of accuracy compared to the humidity models.**

Keywords—Recurrent Neural Network, LSTM, Neural Network, time series, weather, prediction

I. INTRODUCTION

Neural Networks are modelled loosely after biological neural networks that constitute animal brains. They are multi-layer networks of artificial neurons or nodes that are used to, among many other things, make predictions and classify things, for example, images of dogs and cats. The idea is to use many training examples and then develop a method which can learn from these training examples. The neural network uses these examples to automatically infer rules for recognizing patterns in the data. Convolutional Neural Networks (CNN) are a category of neural networks that have proven very effective in areas such as classification and image recognition. CNNs consist of an input layer, multiple hidden layers, and an output layer. The hidden layers consist of a series of convolutional layers which convolve with a dot product [2][3].

A Recurrent Neural Network (RNN) is a special type of neural network which makes use of sequential information. In a traditional neural network, it is assumed that the inputs and the outputs are independent of each other. However, for many tasks this is not a good assumption. For example, when it comes to predicting the next word in a sentence, it is required to have knowledge of the previous words. RNNs are recurrent in that they perform the same task for every element of a series and the output is depended on the previous computations. These networks have "memory" and they capture information about what has been calculated up to that point [4][6].

A. Long Short-Term Memory (LSTM)

All RNNs have feedback loops in the recurrent layer which allows them to maintain information in memory overtime. However, it is difficult to train standard RNNs to solve problems that involve learning long-term temporal dependencies. Consider for example a language model trying to predict the last word in the sentence "the clouds are in the *sky*" based on previous words. It seems reasonably obvious the next word is sky and we do not need any further context. In these cases where there is a narrow gap between the relevant contextual information and where it is needed, RNNs can learn to use past information. But consider a case where we need more information, for example, trying to predict the last word in the sentence "I grew up in Italy…I speak fluent *Italian*. Recent information suggests that the next word is likely to be the name of some language. But to narrow down the exact language requires the textual context of Italy from further back in the sentence. As the gap between relevant information and where it is needed grows large, RNNs are not able to learn to connect the information. This is due to the vanishing-gradient problem where the gradient of the loss function decays exponentially with time [2][4].

Fortunately, LSTMs do not have this problem and they are capable of learning long-term dependencies. The key to this ability is that an LSTM is characterized by a persistent linear cell state. The cell state is akin to a conveyor belt which runs straight down a chain with some minor linear interactions. It is relatively easy for information to flow along this belt unchanged. The ability to add or remove information to the cell state is carefully regulated by structures known as gates [4][5].

LSTM architecture is characterized by this cell state surrounded by non-linear layers which feed input and parse

output from it. The cell state works together with 4 gating layers, the forget, (2x) input and output gates. These gates are made from a sigmoid neural net layer and pointwise multiplication operation. The sigmoid layer outputs numbers between 0 and 1 which indicate how much of each component passes the gate. Following is a brief description of each of the gates [1][2].

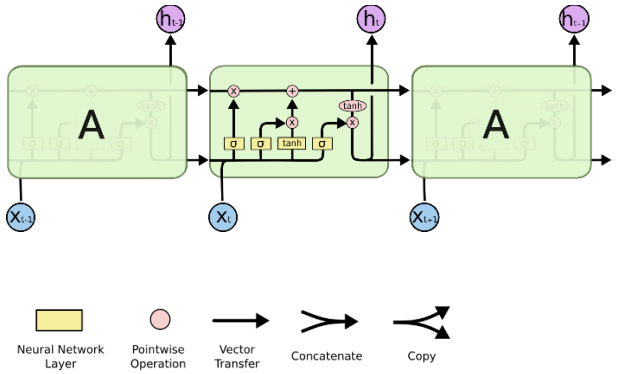

Fig. 1. Structure of the repeating module in LSTM which contains four interacting layers. [6]

i. Forget gate

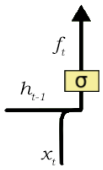

Fig. 2. Structure of forget gate within an LSTM cell. [6]

Based on the current input data, the forget gate determines which values of the old cell state to get rid of. This gate takes in two inputs: h_{t-1} and x_t. h_{t-1} is the output of the previous cell and x_t is the input at that specific time step. These inputs are multiplied by weight matrices with a bias added. A sigmoid function is then applied which outputs values ranging from 0 to 1 for each value in the cell state. If '0' is the output, the cell state forgets that specific information, and if '1' is the output, the cell state remembers it [3].

ii. Input gates

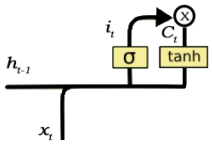

Fig. 3. Structure of input gates within an LSTM cell. [6]

The two input gates have different activation functions and they together decide which values to add to the cell state depending on the input. A sigmoid layer decides which values to update and a tanh layer creates a vector of new candidate values to be added to the state. These two are combined to make an update to the state [6].

iii. Output gate

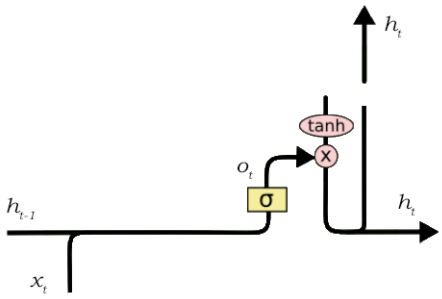

Fig. 4. Structure of output gate within an LSTM cell. [6]

The output gate decides which parts of the cell state should be passed to the output. A tanh function is applied to the cell state which creates a vector of values ranging from -1 to 1. To regulate values that need to be outputted from this vector, a filter is created using h_{t-1} and x_t. The filter is then applied to the vector and the result is sent both as output and hidden state of the next cell [2][6].

II. PROJECT STRUCTURE

In this project we implemented LSTM architecture to create models that forecast temperature and humidity for three major cities in Texas. A series of temperature and humidity data points for San Antonio, Dallas and Houston were given as inputs to the neural network.

A. Dataset

The dataset used in the project consisted of historical hourly weather data for 2012-2017 for 30 US and Canadian cities, and 6 Israeli cities. Out of these cities, we choose three cities in Texas given that we are based in Texas. The dataset includes approximately 5 years of the latest high temporal resolution data or hourly measurements of various weather attributes such as air pressure, humidity, temperature, wind direction etc. For our prediction task, we chose temperature and humidity for simplicity. Additionally, these are rather common measurements that are widely used by most people on a regular basis and thus their prediction may be the most useful.

The temperature dataset contains 45253 samples. Removing null values leave 45252 samples for San Antonio temperature, 45250 for Houston and 45249 for Dallas. The samples for each city were divided into 30000 samples for training the model and the rest were allocated for testing. To mimic sequential data, we divided every set of 34 training samples into subsets of 30 and 4 data points. The 30 either represent the past 30 hours' temperature or humidity. And the 4 represent the next 4 hours temperature or humidity. The model

was trained on the 30 hours of temperature or humidity data and attempts to predict the next 4 hours of temperature or humidity at each of the subsets.

B. Model

An LSTM Model has been implemented along with the usage of python libraries like NumPy, matplotlib, sklearn, pandas, and keras. For the temperature dataset we used two different models to compare accuracy and loss generated. The second model has more LSTM layers along with dropout. Dropout is a regularization technique which randomly drops some neurons from the network when processing a batch during training. This forces each neuron to learn on its own rather than relying on other neurons.

In the second model we also lowered the learning rate and increased the batch size and epochs to improve accuracy. Smaller learning rates need more training epochs because smaller changes are made to the weights during each update in backpropagation. It was decided mean squared error would be the loss function. The efficiency of the model can be determined by the accuracy, and the chosen loss function. For the humidity dataset we compare accuracy and loss generated by the models by executing the first model with lower epochs and the second model with higher epoch. For this type of implementation dropout can be used to fine tune the model, and a dense SoftMax layer has been used in the end to display the probability of confidence for the predicted values.

C. Results

For the temperature dataset we run two different models, the first model has less LSTM layers compared to the second one. For the Humidity dataset, the first model has lower epochs and a higher batch size, whereas the second model has higher epochs and a lower batch size. The results were visualized using the matplotlib python package.

The visualized results are shown below:

Model 1 San Antonio Temperature model:

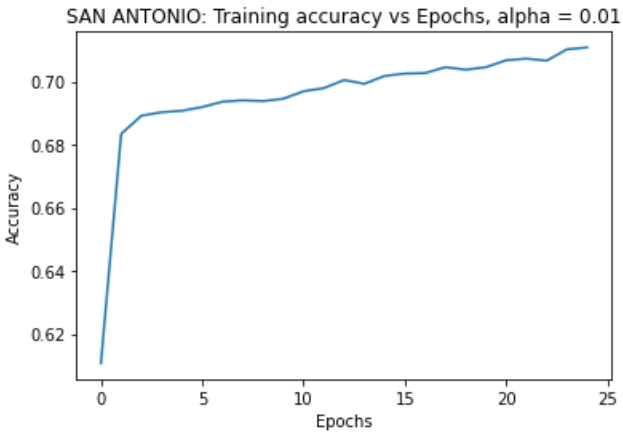

Fig. 5. Training accuracy vs Epochs for the San Antonio temperature model

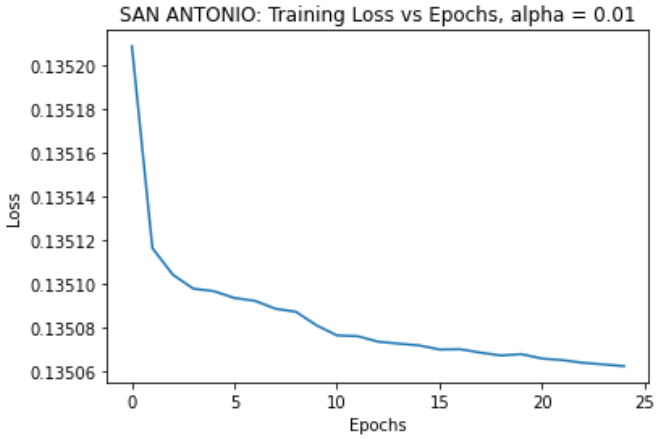

Fig. 6. Training loss vs Epochs for the San Antonio temperature model

Model 2 San Antonio Temperature model with improved accuracy:

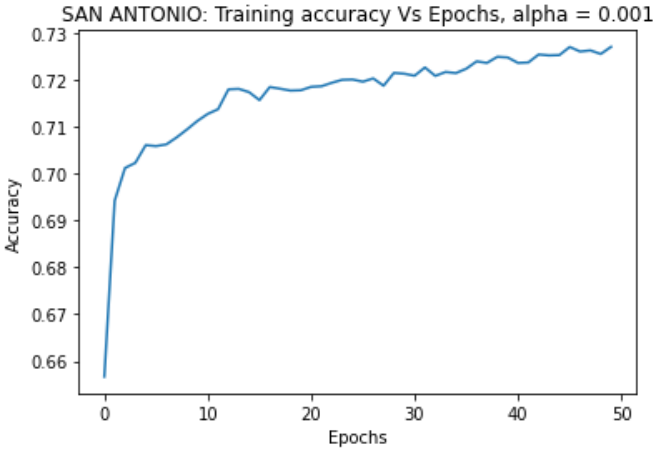

Fig. 7. Training accuracy vs Epochs for the San Antonio temperature model that resulted in improved accuracy

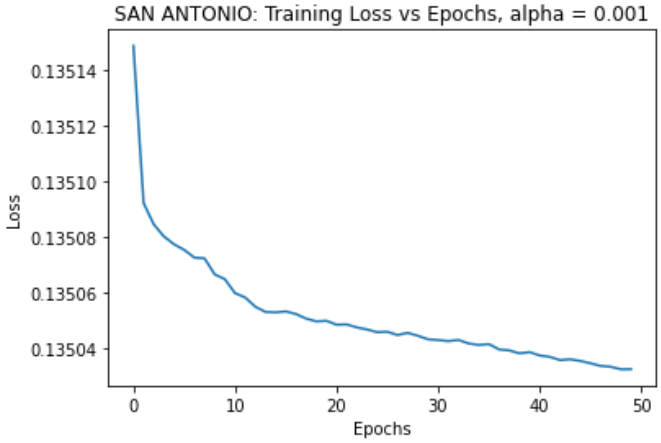

Fig. 8. Training loss vs Epochs for the San Antonio temperature model with improved accuracy

Like San Antonio, the second model with more LSTM layers resulted in higher accuracy for both Dallas and Houston. These results are shown below.

Model 2 Dallas Temperature model with highest accuracy

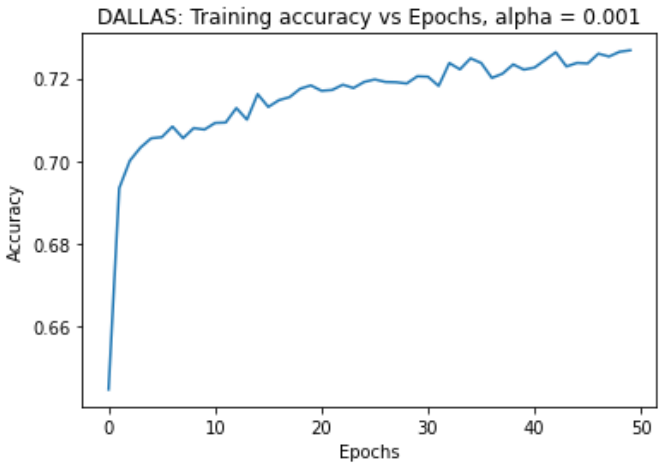

Fig. 9. Training accuracy vs Epochs for the Dallas temperature model with the highest accuracy

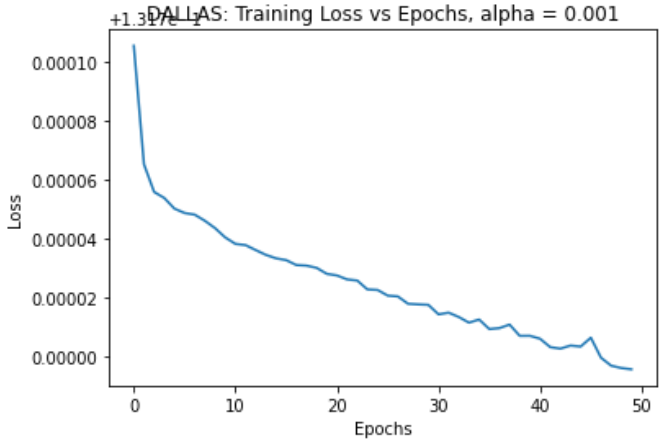

Fig. 10. Training loss vs Epochs for the Dallas temperature model with the highest accuracy

Model 2 Houston Temperature model with highest accuracy

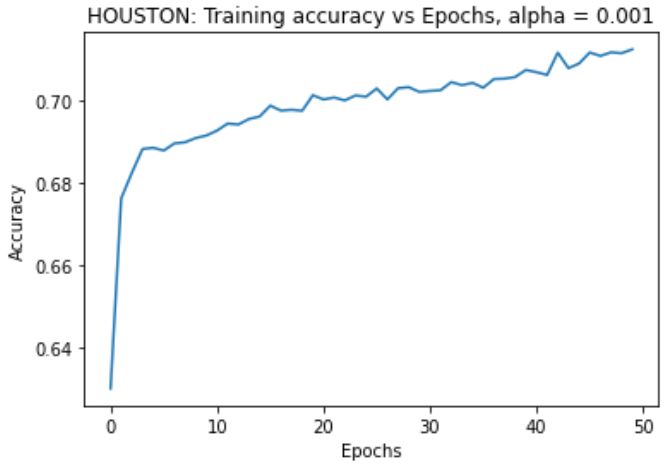

Fig. 11. Training accuracy vs Epochs for the Houston temperature model with the highest accuracy

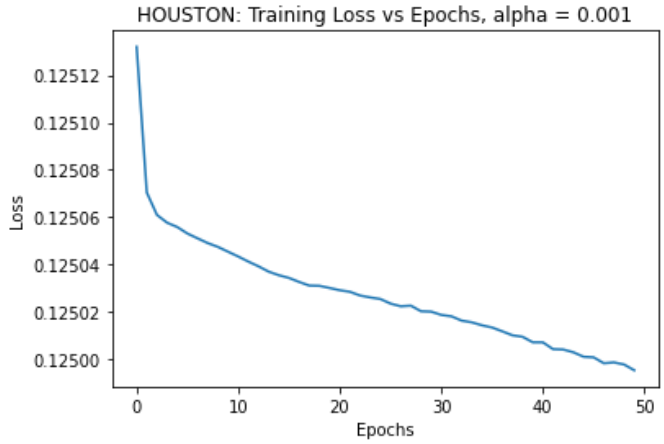

Fig. 12. Training loss vs Epochs for the Dallas temperature model with the highest accuracy

For the humidity models, the accuracy was lower for all the three cities compared to the temperature models. Two models were run for each city and the model that resulted in higher accuracy is included in the results below.

Humidity model for San Antonio with the highest accuracy

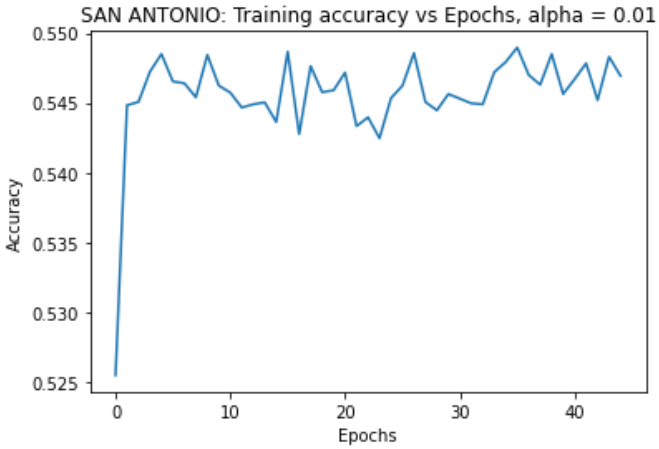

Fig. 13. Training accuracy vs Epochs for the San Antonio humidity model with the highest accuracy

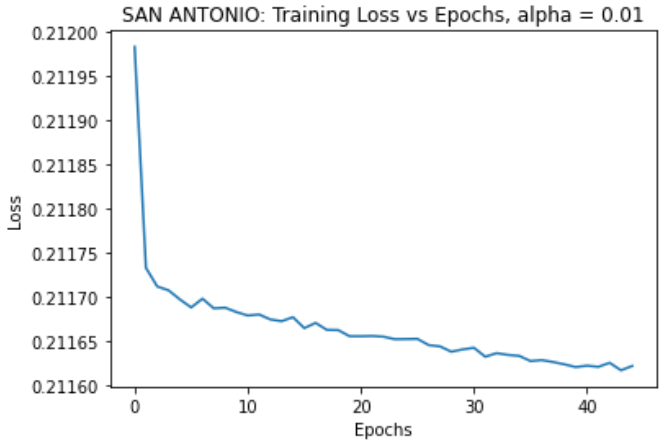

Fig. 14. Training loss vs Epochs for the San Antonio humidity model with the highest accuracy

Humidity model for Dallas with the highest accuracy

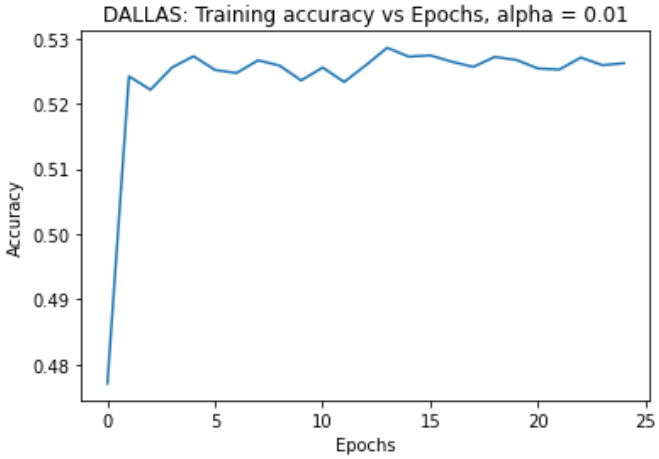

Fig. 15. Training accuracy vs Epochs for the Dallas humidity model with the highest accuracy

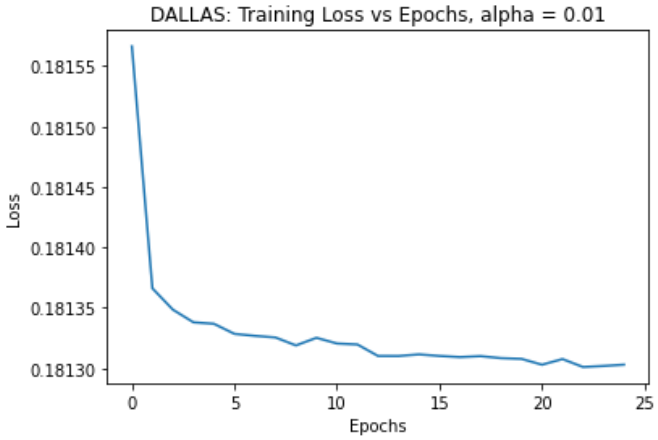

Fig. 16. Training loss vs Epochs for the Dallas humidity model with the highest accuracy

Humidity model for Houston with the highest accuracy

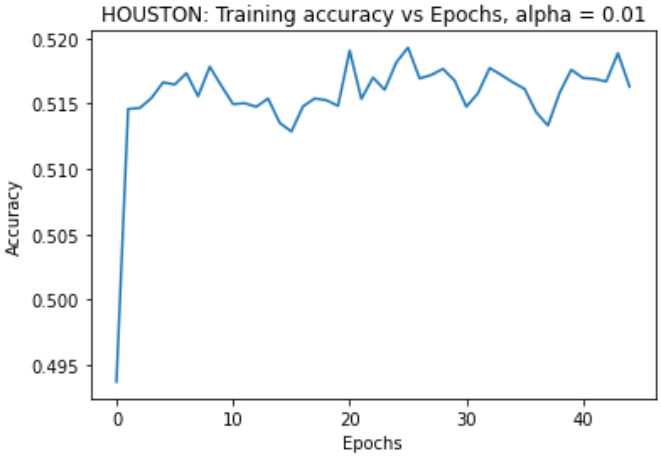

Fig. 17. Training accuracy vs Epochs for the Houston humidity model with the highest accuracy

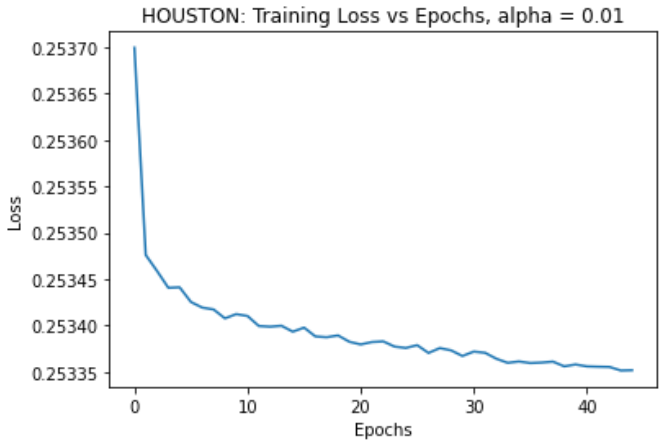

Fig. 18. Training loss vs Epochs for the Houston humidity model with the highest accuracy

D. Analysis

Two models were run for the temperature dataset for each of the three cities. The first model had only one LSTM layer whereas the second model consisted of three LSTM layers along with dropout. The additional LSTM layers contributed to improving the accuracy in the second model. For each of the three cities, the second model resulted in slightly improved accuracy. For example, accuracy was 70.67 percent in the first model for San Antonio temperature and it increased to 72.53 percent in the second model.

Additionally, changing hyperparameters like batch size, number of epochs and learning rate also contributed to the slight improvement in testing accuracy. The increase in the number of epochs with a decrease in batch size helped to reduce training loss. Lowering the training rate to 0.001 helped to increase the accuracy in the second model but we did not observe any significant changes in the training loss. We experimented with different optimizers such as Adam and SGD. SGD resulted in very low accuracies close to 20 percent. The adaptive learning rate optimization algorithm Adam resulted in the highest training accuracies for all the models.

E. Conclusion and Future Work

The goal in this project was to build models using LSTM neural networks to predict temperature and humidity for three cities in Texas. Based on the temperature or humidity of the past 30 hours, the models predict the temperature or humidity of the next 4 hours. The temperature models resulted in high accuracy above 70 percent, but the humidity models resulted in accuracy of around 55 percent. The humidity model especially can be improved in the future by hyperparameter tuning. Additionally, the models can be improved by using a bidirectional LSTM which improves performance with sequential data. They train two LSTMs on input data instead of one. The first LSTM is trained on the existing sequence as is and the second one is trained on a reversed input sequence.

REFERENCES

[1] Sherstinsky, Alex, “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network,” Physical D: Nonlinear Phenomena, vol. 404, March 2020.

[2] Lipton, C. Zachary, Berkowitz. John, Elkan. Charles A Critical Review of Recurrent Neural Networks for Sequence Learning, 2015.

[3] Yong. Yu, Xiaosheng. Si, Changhua. Hu, Jianxun. Zhang , “A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures,” Neural Computation, vol. 31, No. 7, Massachusetts Institute of Technology, 2019, pp. 1235–1270.

[4] Gers. A Felix, Schmidhuber. Jurgen, Cummins. Fred, “Learning to Forget: Continual Prediction with LSTM,” Technical Report IDSIA-01-99,1999.

[5] Greff. Klaus, Srivastava K. Rupesh, Koutnik. Jan, Steunebrink R. Bas, Schmidhuber. Jurgen, “LSTM: A Search Space Odyssey,” Transactions on Neural Networks and Learning Systems, 2016.

[6] Srivastava. Pranjal, “Essentials of Deep Learning: Introduction to Long Short Term Memory,” Analytics Vidhya, 2017.