Dallas Temperature Prediction using RNN and LSTM

Ishwari Joshi
Computer Science
UT Dallas
Richardson, US
IGJ180000@utdallas.edu

***Abstract*—Both RNNs and LSTMs can be used for time series forecasting. This project compares the results of the implementation of a simple, single-layer RNN from scratch with a single LSTM layer network built using the Keras library. The dataset used consists of hourly weather data in Dallas, Texas from 2012 to 2017. The results show that the LSTM follows the trends in temperature very closely and is a better predictor of the dips and peaks in temperature while the RNN is inferior in predicting the temperature fluctuations. LSTM is better suited to this application due to its selective use of long-term and short-term dependencies as well as the complex patterns that temperature displays depending on past temperature, pressure, humidity, and wind speed measurements over time. This project can be extended to implement more layers in the RNN and even the LSTM from scratch. There are open research issues about the optimal choice of models and predictors, appropriate sequence lengths, number and types of layers, generalizability of models (will this model apply to other cities?), and explainability (what is causing temperature predictions to be correct or incorrect?).**

Keywords—RNN, LSTM, temperature prediction

I. Introduction and Background Work

Time series forecasting is a technique applied to predict future values and trends over time based on past historical data. It assumes that future values follow a similar trend to past values. Time series forecasting is typically used for tasks such as weather, stock market price, and product sales prediction.

The dataset used in this project is the Historical Hourly Weather Data 2012-2017 from Kaggle [1]. It contains hourly weather data from 30 US and Canadian cities plus 6 Israeli cities. with 45253 total data observations. For this implementation, I focus on the data for the city of Dallas. The dataset consists of 7 csv files: city_attributes, humidity, pressure, temperature, weather_description, wind_direction, and wind_speed. Of these, I focus on temperature, pressure, humidity, and wind_speed by using them as predictors for the temperature the next day (24 hours later). The dataset is publicly hosted on GitHub [2].

A recurrent neural network (RNN) and long short-term memory (LSTM) network are implemented to perform time series forecasting on Dallas temperature. The coding language used is PySpark on Databricks. PySpark is the Python API that is used for Spark. Databricks is a tool built on top of Spark that allows users to develop and run Spark-based applications. As an open-source cluster computing framework, it processes data in a fast, efficient, and easy-to-use way.

II. Theoretical and Conceptual Study

A. Recurrent Neural Network (RNN)

RNNs are a class of neural networks that are useful for processing sequential or time series data. A RNN works by taking in an input from a sequence, such as a single temperature value, and passing it through the input layer, hidden layer, and output layer. The hidden layer remembers information about previous inputs and is where the "recurrent" part of a recurrent neural network occurs. This means that the hidden units are connected to each other across timesteps. Each hidden step has information about all previous hidden steps, enabling its ability to feed each sequence element both to the output and forward to the next sequence element as shown in Fig. 1. In this way, a RNN can construct a memory of the sequence and new features made up of past sequence elements [3].

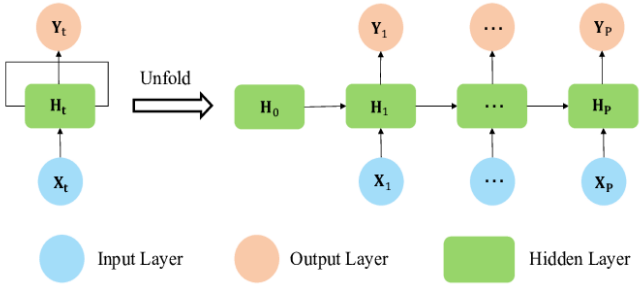

Fig. 1. RNN diagram [4]

An important property of RNNs is parameter (weight and bias) sharing, where the RNN shares the same weights for each input, hidden, and output step across the entire sequence. In other words, there are three weight matrices: one for the input step, one for the hidden step, and one for the output step. For example, 4 input units in the input layer, 5 hidden units in the hidden layer, and 1 output unit in the output layer will result in the following weight matrix shapes: (4, 5) for the input weights, (5, 5) for the hidden weights, and (5, 1) for the output weights. In addition, the hidden bias shape will be (1, 5) and the output bias shape will be (1, 1) as shown in Fig. 2. These depend only on the layer design and have no relation to sequence length. A RNN can learn relationships between sequence elements without having separate parameters for each sequence element.

```
[[ 0.04366013  0.19247122  0.09191436  0.04014474 -0.06828522]
 [ 0.13049166 -0.0558237   0.35041242  0.41471258 -0.10425307]
 [ 0.26092681  0.0258444   0.06086091  0.38066521 -0.38367701]
 [-0.36928278 -0.42912971  0.29750423  0.24879096  0.33094893]]
(4, 5)
[[ 0.42808926  0.26757555 -0.03445391  0.25091292 -0.34142573]
 [ 0.12514917 -0.31899452  0.39772397  0.01954173 -0.07632868]
 [-0.21058786  0.24528207 -0.03922034  0.06120918 -0.43040749]
 [ 0.10521639  0.10026146  0.10458895  0.39690035  0.16262502]
 [-0.12565995 -0.05632033  0.17676672 -0.3933463   0.14916068]]
(5, 5)
[[ 0.15262315 -0.25904171 -0.33189841 -0.1650859  -0.12190079]]
(1, 5)
[[ 0.0627859 ]
 [-0.05491648]
 [ 0.43681484]
 [-0.35594194]
 [-0.26038855]]
(5, 1)
[[-0.30293398]]
(1, 1)
```

Fig. 2. Parameter initialization - 4 input units, 5 hidden units, and 1 output unit

Training a RNN consists of a forward pass and a backward pass. The forward pass predicts an output for each input sequence element. It has three steps: (1) take each sequence element and multiply it by the input weight, (2) multiply the previous hidden state with the hidden weight, add in the input, and apply a nonlinear activation function, and (3) multiply the hidden state by the output weight. Bias terms are added in the hidden and output steps. For capturing a non-linear prediction model, the tanh activation function is a good choice because its derivative has a steep slope, making it ideal for gradient descent.

Backpropagation through time sends the gradient with respect to each hidden step backwards to the previous sequence position, updating the parameters of the RNN to reduce training error. The error is backpropagated from the last to the first timestep, where the error is the difference between the RNN's prediction and the actual value. The gradient is the partial derivative of error with respect to the parameters. In other words, the gradient measures the change in weights and biases based on the change in error [5].

The problem with RNNs is that they tend to forget past inputs due to their short-term memory, making it difficult for them to predict long-term dependencies. The vanishing gradient problem is where the gradients used in the weight update computations start to approach zero, which prevents the network from learning new weights.

B. Long Short-Term Memory Network

LSTM is a type of RNN that solves the vanishing gradient problem. LSTM captures long-term dependencies better than RNNs. LSTM are used as the building blocks for the RNN layers. A LSTM layer is an RNN layer that learns long-term dependencies between timesteps in time series data. The layer state consists of the hidden state, also known as the output state, and the cell state. The hidden state contains the output of the LSTM layer at each timestep. The cell state contains the information that has been learned from previous timesteps [6]. LSTM has memory that can be viewed as a gated call, with "gated" meaning that the cell decides whether to store or delete information based on the importance it assigns to the information. The layer adds or removes information from the cell state at each timestep. By doing so, the LSTM learns what information is important and what is not over time.

In a LSTM cell, there are three gates: an input gate, an output gate, and a forget gate as shown in Fig. 3. These gates are sigmoid activation functions which open or close based on state values. This determines which information to retain and what to forget. The input gate determines whether to let new input in. The forget gate deletes the information that is not important. The output gate allows the information to impact the output at the current timestep. In this way, LSTM solves the vanishing gradient problem by keeping the gradients steep and not vanishing [5].

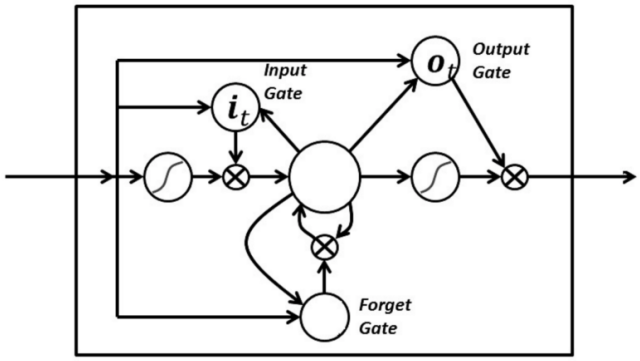

Fig. 3. LSTM diagram [5]

III. Preparing the Data

Since the data is in separate csv files, we begin by creating a single DataFrame that holds the necessary predictors: temperature, pressure, humidity, and wind speed. Each of the csv files is read into a separate DataFrame and then joined into one DataFrame. The null values in each column are replaced with the respective previous row value. For example, if the pressure value in row 5 is null, we propagate the pressure value from row 4 there as an attempt to maintain the data's trend over time. A new column is inserted to contain the temperature of the next day (24 hours later); let this be known as temperature tomorrow. By shifting the temperature column up 24 rows, we can fill in the "temperature tomorrow" column. Then, the last 24 rows of data are dropped (after shifting, those rows will be unknown and thus cannot be predicted). We now know the actual values that the network should predict. Fig. 4 shows the first few rows of the final DataFrame and Fig. 5 shows the weather data visualized over time.

	datetime	temp	pressure	humidity	windspeed	temp_tomorrow
1	2012-10-01T13:00:00.000+0000	61.86200000000006	1011	87	3	64.7099730662
2	2012-10-01T14:00:00.000+0000	61.90335357260007	1011	86	3	64.83200000000002
3	2012-10-01T15:00:00.000+0000	62.025380506400076	1011	86	3	69.47600000000008
4	2012-10-01T16:00:00.000+0000	62.147407442000066	1011	86	3	72.266
5	2012-10-01T17:00:00.000+0000	62.269434375800074	1011	86	3	75.72200000000004
6	2012-10-01T18:00:00.000+0000	62.39146130960008	1011	86	3	79.68200000000002

Fig. 4. DataFrame contents

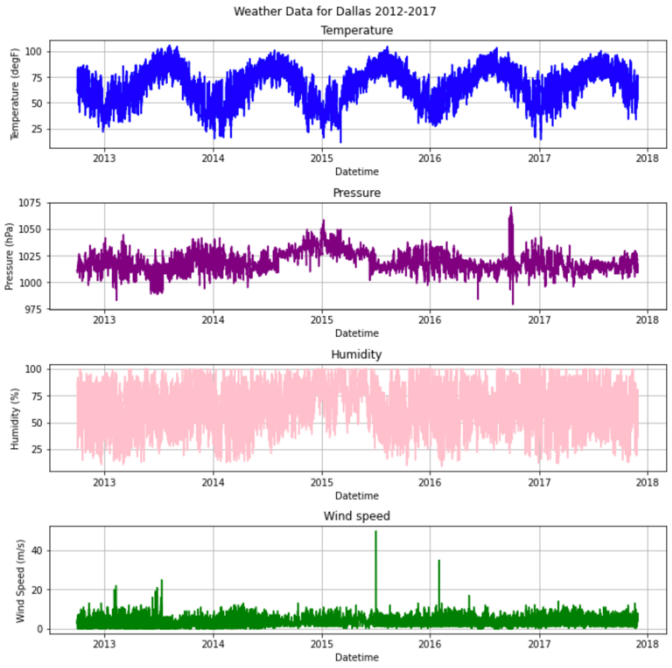

Fig. 5. Weather data visualization

IV. RNN Implementation

Because weather data tends to show correlation (for example, the higher the humidity, the higher likelihood for rainfall), current hourly measurements of temperature, pressure, humidity, and wind speed are all used as predictors for temperature tomorrow (24 hours later). Current temperature alone may not be as good of an indicator for temperature tomorrow. The more important and relevant predictors, the better. The data is standardized using StandardScaler() and then split into training, validation, and test sets. The first 70% of data is used for training, the next 15% of data is used for validation, and the final 15% of data is used for testing. Each set is split into x and y data with x containing the predictors data (4 columns) and y containing the predictions data (1 column). Functions to initialize the parameters, run a forward pass, and run a backward pass are written.

The RNN is defined to have 4 input units (because there are 4 features), 5 hidden units, and 1 output unit. The parameters of the RNN are initialized randomly before the training loop. The RNN is trained over 50 epochs with a learning rate of 1e-5. For each epoch, each sequence length of 20 in the training set is run through a forward and backward pass and the training loss is evaluated. The loss used is mean squared error (MSE). A sequence length of 20 means that 20 hours of weather is used to get the predictions of temperature 24 hours later from the current time. Every 10 epochs, each sequence length of 20 in the validation set is run through a forward pass and the validation loss is displayed. Once training is complete, each observation in the test set is run through a forward pass, the last entry in the outputs is saved as the prediction of temperature tomorrow. The test loss (MSE) is evaluated for all the predictions made on the test data.

V. LSTM Implementation

As with the RNN, the data is standardized using StandardScaler() and then split into training and test sets. Here, the first 85% of data is used for training and the final 15% of data is used for testing (since there is no need for a validation set). The data containing the predictors is reshaped to (number of observations, number of features, 1) to feed into the LSTM. The LSTM model is built using the Keras library with a Sequential() layer, a LSTM layer with 5 units, input shape of (4, 1), and tanh activation, and a Dense (fully connected) layer with 1 unit. The model uses the default learning rate (1e-3) of the Adam optimizer with MSE loss. Fig. 6 is a summary of the model.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 5)                 140

 dense (Dense)               (None, 1)                 6

=================================================================
Total params: 146 (584.00 Byte)
Trainable params: 146 (584.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Fig. 6. LSTM model summary

The model is then trained, and the MSE is plotted over epochs in Fig. 7. The model is evaluated on the test set to get the test loss. Finally, the model is used to make predictions on the test data.

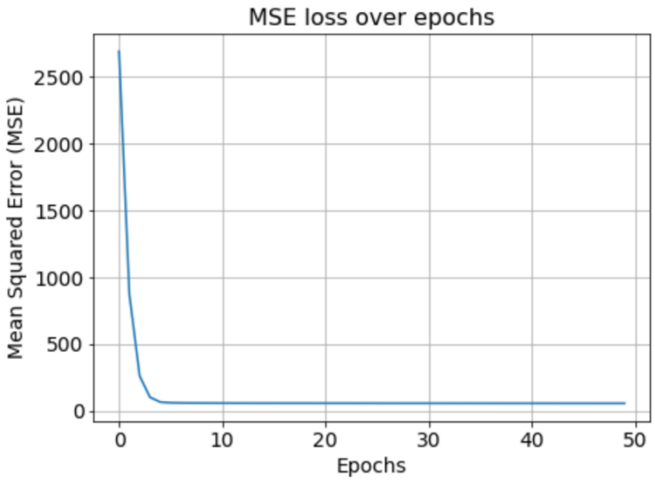

Fig. 7. LSTM MSE loss over epochs

VI. RNN Results and Analysis

For a sequence length of 20, the RNN's training loss decreases over epochs while the validation loss increases in Fig. 8. This is a sign of overfitting, meaning that the network is being too specific to the training set and not generalizing well to other new data not used in training. However, the test loss (67.97) is comparable to the training loss. The test predictions are plotted on top of the actual temperature tomorrow values in Fig. 9. While the predictions follow the general pattern of the

temperature, they do not drop below 55 degrees Fahrenheit even when the actual values drop below 40. In the summer months from June to August 2017, the RNN predicts values lower than the actual.

```
Epoch: 10 train loss 55.42309837624749 valid loss 88.53519780553101
Epoch: 20 train loss 50.67947278980579 valid loss 95.99164922255915
Epoch: 30 train loss 47.71584131375005 valid loss 100.89719062052887
Epoch: 40 train loss 46.273653102458596 valid loss 106.0640037999241
Epoch: 50 train loss 44.83803785133089 valid loss 111.41279013625108
```

Fig. 8. RNN training and validation loss over epochs (sequence length 20)

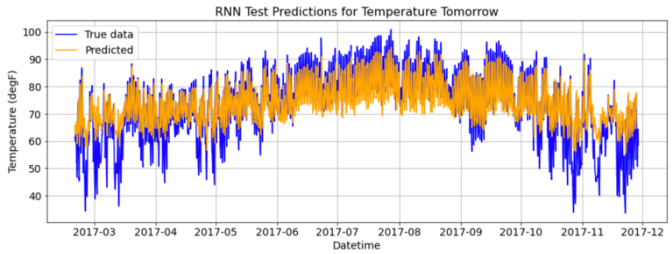

Fig. 9. RNN predictions vs. actual (sequence length 20)

Another trial with a sequence length of 10 is run to see if there is any improvement in the predictions. The training loss decreases similarly to sequence length 20, but the validation loss increases and then decreases from epoch 40 to 50 as shown in Fig. 10. The decrease in validation loss may indicate that this RNN has less overfit than the previous RNN. The test loss (73.89) is higher than before, but looking at the plot (Fig. 11), there one key difference: the summer months are much more accurately predicted.

```
Epoch: 10 train loss 57.057044324458595 valid loss 92.72122408452842
Epoch: 20 train loss 52.743903367374585 valid loss 102.64247516055697
Epoch: 30 train loss 49.20832407184691 valid loss 105.38864646629011
Epoch: 40 train loss 49.8192275933687 valid loss 118.62728114007719
Epoch: 50 train loss 48.60508477831948 valid loss 110.28154088482036
```

Fig. 10. RNN training and validation loss over epochs (sequence length 10)

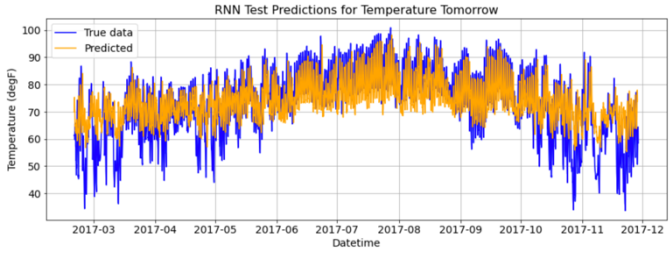

Fig. 11. RNN predictions vs. actual (sequence length 10)

More parameters i.e., learning rate, number of epochs, and sequence length, can be changed to see what most affects the RNN performance and if this simple, single-layer RNN can predict the lower temperatures. In Fig. 12, at the start of the test data, the RNN overpredicts the actual value by roughly 10 degrees Fahrenheit, but the tenth prediction is spot on. Fig. 13 shows temperatures from July where the RNN underpredicts the actual value by roughly over 10 degrees Fahrenheit. Although the RNN does not follow the complex changes in temperature over time, it does make some accurate predictions and should not be entirely discounted for its performance.

	Datetime	True Data	Predicted
1	2017-02-19T08:00:00.000+0000	61.3796	69.88942226163356
2	2017-02-19T09:00:00.000+0000	61.3796	70.54076121689148
3	2017-02-19T10:00:00.000+0000	61.3796	70.4883516233472
4	2017-02-19T11:00:00.000+0000	61.19960000000007	75.14248958597457
5	2017-02-19T12:00:00.000+0000	61.19960000000007	75.30125203711461
6	2017-02-19T13:00:00.000+0000	61.19960000000007	75.23846605267536
7	2017-02-19T14:00:00.000+0000	61.48760000000001	71.0085320601171
8	2017-02-19T15:00:00.000+0000	61.48760000000001	70.5918952878425
9	2017-02-19T16:00:00.000+0000	59.7182	70.6137014817164
10	2017-02-19T17:00:00.000+0000	65.58260000000007	65.56653641927639

Fig. 12. RNN predictions vs. actual (sequence length 10) at start of test set

	Datetime	True Data	Predicted
1	2017-07-14T18:00:00.000+0000	83.192	82.09547731222774
2	2017-07-14T19:00:00.000+0000	87.8	80.39987953981623
3	2017-07-14T20:00:00.000+0000	90.71600000000001	79.23910439249339
4	2017-07-14T21:00:00.000+0000	90.71600000000001	77.94681250631086
5	2017-07-14T22:00:00.000+0000	91.02200000000003	76.74103788490461
6	2017-07-14T23:00:00.000+0000	91.27400000000002	77.06514457227095
7	2017-07-15T00:00:00.000+0000	91.14800000000002	76.01429270824816
8	2017-07-15T01:00:00.000+0000	89.6	74.87752949220146
9	2017-07-15T02:00:00.000+0000	85.38800000000005	74.10034308948826
10	2017-07-15T03:00:00.000+0000	82.77800000000006	74.23284820881645

Fig. 13. RNN predictions vs. actual (sequence length 10) in middle of July

VII. LSTM Results and Analysis

On the other hand, the LSTM very closely follows the temperature trend. It has a test loss of 51.80, which is lower than both the RNNs above, and can predict the lower temperatures very accurately unlike the RNN. We can see in Fig. 14 that the LSTM outperforms the RNN and captures the complex patterns that the RNN cannot.

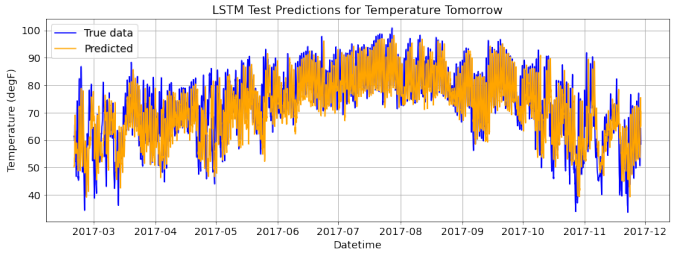

Fig. 14. LSTM predictions vs. actual

The RNN and LSTM are trained over the same number of epochs but use different learning rates. The RNN's learning rate is 1e-5 while the LSTM's is 1e-3. Higher learning rates were tried for the RNN, but it did not make any meaningful predictions due to underfitting. With a higher learning rate, training goes through the epochs faster but generates oscillations in validation loss which may result in underfit. With a lower learning rate, smaller changes are made to the weights on each update, but more epochs are required to complete the training and the training time is longer. In Fig. 15, at the start of the test data, the LSTM underpredicts within 7 degrees Fahrenheit. These predictions are within a closer margin than the RNN (Fig. 12). Fig. 16 shows temperatures from July where the LSTM predicts the temperatures very accurately with a margin of only a few degrees Fahrenheit.

	Datetime	True Data	Predicted
1	2017-02-19T08:00:00.000+0000	61.3796	54.060776
2	2017-02-19T09:00:00.000+0000	61.3796	54.060776
3	2017-02-19T10:00:00.000+0000	61.3796	54.060776
4	2017-02-19T11:00:00.000+0000	61.19960000000007	54.48448
5	2017-02-19T12:00:00.000+0000	61.19960000000007	54.48448
6	2017-02-19T13:00:00.000+0000	61.19960000000007	54.48448
7	2017-02-19T14:00:00.000+0000	61.48760000000001	57.077805
8	2017-02-19T15:00:00.000+0000	61.48760000000001	57.077805
9	2017-02-19T16:00:00.000+0000	59.7182	57.077805
10	2017-02-19T17:00:00.000+0000	65.58260000000007	64.52386

Fig. 15. LSTM predictions vs. actual at start of test set

	Datetime	True Data	Predicted
1	2017-07-14T18:00:00.000+0000	83.192	91.80663
2	2017-07-14T19:00:00.000+0000	87.8	92.876274
3	2017-07-14T20:00:00.000+0000	90.71600000000001	93.143616
4	2017-07-14T21:00:00.000+0000	90.71600000000001	93.58768
5	2017-07-14T22:00:00.000+0000	91.02200000000003	94.012085
6	2017-07-14T23:00:00.000+0000	91.27400000000002	93.644
7	2017-07-15T00:00:00.000+0000	91.14800000000002	92.90764
8	2017-07-15T01:00:00.000+0000	89.6	90.55528
9	2017-07-15T02:00:00.000+0000	85.38800000000005	88.47398
10	2017-07-15T03:00:00.000+0000	82.77800000000006	86.06637

Fig. 16. LSTM predictions vs. actual in middle of July

VIII. Conclusion and Future Work

Variations of learning rate, sequence length, and number of hidden units were tried for the RNN, but better results were not achieved. Hyperparameter optimization is difficult in part due to the RNN training taking about 15-20 minutes to run. The number of epochs could be increased to see the effect on RNN performance. With a low learning rate, more epochs are needed to train the network and perhaps, the training here was cut short. Another limitation of the RNN includes using uniform random initialization of parameters, which is not a suitable choice for nonlinear optimization. A good starting guess is crucial to good RNN performance. In addition, a RNN is bound to miss long-term trends because of vanishing gradients.

For these reasons, a LSTM should be used. The Keras library LSTM model chooses optimal starting weights that are not randomly chosen and results in a good and faster solution. LSTM training takes about half of the time compared to RNN. In conclusion, LSTM is better suited for time series prediction.

Future work could include adding more layers to the RNN and implementing the LSTM from scratch to get a deeper understanding of its architecture. There are open research issues about the optimal choice of models and predictors, appropriate sequence lengths, number and types of layers, generalizability of models (will this model apply to other cities?), and explainability (what is causing temperature predictions to be correct or incorrect?).

References

[1] https://www.kaggle.com/datasets/selfishgene/historical-hourly-weather-data

[2] https://github.com/ishwari21/bigdataproject

[3] https://www.youtube.com/watch?app=desktop&v=4wuIOcD1LLI

[4] J. Ye, J. Zhao, K. Ye, and C. Xu, "How to Build a Graph-Based Deep Learning Architecture in Traffic Domain: A Survey," IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 5, May 2022.

[5] https://builtin.com/data-science/recurrent-neural-networks-and-lstm#

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural computation, vol. 9, no. 8, pp.1735–1780, 1997.