

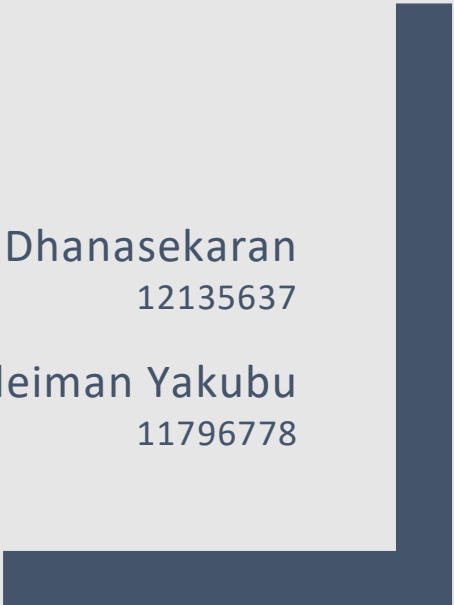


7088CEM COURSEWORK

# WEATHER PREDICTION FOR AUSTRALIA USING RNNS

Roshan Dhanasekaran  
12135637

Suleiman Yakubu  
11796778



# WEATHER PREDICTION FOR AUSTRALIA USING RNNs

*Dhanasekaran, Roshan*

*12135637*

*Faculty of Engineering, Environment, and  
Computing*

*Msc Data Science and Computational*

*Intelligence, Coventry University*

[\*dhanasekar@uni.coventry.ac.uk\*](mailto:dhanasekar@uni.coventry.ac.uk)

*Yakubu, Suleiman*

*11796778*

*Faculty of Engineering, Environment, and  
Computing*

*Msc Data Science and Computational*

*Intelligence, Coventry University*

[\*Yakubus6@coventry.ac.uk\*](mailto:Yakubus6@coventry.ac.uk)

## I. ABSTRACT

***Abstract---*** This paper attempts to predict weather information using the Australian Weather Dataset and Recurrent Neural Networks (RNN). In the paper, we perform preliminary data analysis (PDA), and data preparation techniques were deployed on the data to be used with an RNN. After the PDA, we use vanilla RNNs, Long Short-Term Memory (LSTM) RNNs, and Gated Recurrent Networks (GRU) to predict the median temperature and compare the results. The models are evaluated with the training dataset and then estimated with the testing dataset to get the final outputs. The advantages and limitations of each of the Neural Networks are analysed and discussed as well. The results of the analysis are discussed with suitable visualization to make them easier to communicate. This project was implemented using the Python programming language along with TensorFlow, Keras, NumPy, Pandas, and Sklearn. Plotly was used for the visualization of the outputs. In general, it was found that RNNs are good for predicting sequential data but LSTMs and GRUs performed better when forecasting farther in the future.

***Keywords---*** *Recurrent Neural Network, long short-term memory, preliminary data analysis, Data visualization, Gated Recurrent Network, TensorFlow, Keras, NumPy, Pandas, Sklearn, Plotly*

**NOTE: Links to the code and dataset can be found in the reference section.**

## I. INTRODUCTION

Weather and climate have always been major factors in man's existence. They determined where he could go, what crops he could plant and what animals and plants he could find around him. At first, man was at the mercy of the elements but as he developed and learned, he learned how to predict certain phenomena and correspondingly, how to prepare for them. This contributed immensely to his development.

Weather forecasting has improved to the point where we can tell what the weather is going to be days in advance, but what could we achieve if we could predict the weather and climate months or years in advance. It means pointless loss of lives, properties, resources, and productivity could be avoided. This is what is addressed in the report.

## II. PROBLEM STATEMENT

As a country, Australia is dry, hot, and prone to weather and climate extremes. These conditions can pose threats to lives, infrastructure,

and resources. The weather conditions further have a significant impact on the ecology and can affect biodiversity and the quality of water and air (Bureau of Meteorology, Commonwealth of Australia & CSIRO, 2014). This directly raises a need to be prepared for these extremes as the costs tend to be high. This preparation involves pre-knowledge about extreme weather days, weeks, months or even years in advance if possible. This is where this project comes in; this project proposes the use of Deep Neural Networks, specifically Recurrent Neural Networks (RNN) to forecast weather information and encourage planning for weather extremes, thus mitigating the adverse effects of those extremes.

### III. RELATED WORKS

(Schroeter, 2016) utilised a model in which he nowcasted rainfall for Australia using an Artificial Neural Network (ANN). Nowcasting weather entails predicting it on a short scale. His model was meant to address the shortcomings of the previous radar-based model which failed due to sparse radar coverage, especially in the regional areas of Australia.

The ANN's performance was compared against Hydro-Estimator, a proprietary model used by the US National Oceanographic and Atmospheric Administration (NOAA). This model uses infrared satellite data to estimate the location and rate of rainfall, especially in large, populated areas. The ANN used by the author was a simple feed-forward Neural Network with 10 features in the input layer, 30 neurons in the hidden layer, and

a single output neuron which produces a regressed value that shows how much precipitation is anticipated. The network used standard backpropagation via Stochastic Gradient Descent (SGD) for training.

Even though the network showed promising results at first, the performance was comparable to the Hydro-Estimator model and as such the author concluded that the network could have performed better with better parameter tuning, a more balanced dataset, and some feature engineering such as Principal Component Analysis (PCA).

While the entire the project was an attempt to forecast weather to benefit from the foreknowledge, the scope of the forecast was severely limited and as such would not have been of much use in the use case of this project.

(Polishchuk et al., Sep 22, 2021) used machine learning to predict rainfall one day in advance. The model the authors built uses more classical machine learning algorithms as the authors treated the prediction as a binary classification problem (the two classes are; it will rain tomorrow, and it will not rain tomorrow). The authors used a Random Forest classifier that was trained on meteorological data of each day for a period of about 10 years (between 2007 to 2017).

The authors further stated clearly that their model, which used statistical data to predict rainfall, should be used along with models that

predicted rainfall using meteorological data. As such, their model was created to be part of a system that helped predict weather in order to mitigate the effects of its extremes such as forest fires.

The main drawback of their model is that it predicts only one day in advance. This means that there is barely enough time to put sufficient measures in place to handle the effects of extreme weather. As such, their model on its own cannot form the basis of a plan.

(Raval et al., 2021) compiled a collection of Machine Learning algorithms, including a neural network, and compared their performances. They also treated the forecasting problem as a binary classification. This means that they forecasted rain only a day into the future.

In their comparison of algorithms, they prioritised recall and F1 score, and the neural network had the best performance. The neural network used had 7 dense layers, 3 batch normalization layers and 3 dropout layers with 60% dropout. They concluded that the neural net was best for forecasting rainfall.

The authors failed to point out any directions that future works could take. Also, as their model predicted rain only a day into the future, it has limited use in planning for natural disasters without modification.

## IV. DATASET DESCRIPTION

The dataset used in this paper was taken from Kaggle called *Rain in Australia*. This dataset is linked with the Australia Bureau of meteorology

and consists of 10 years of daily weather and climate change observation from several locations across Australia. The default dataset contains 23 columns. We further add one more column; Median Temperature (*MedTemp*) to the dataset. The final shape of the dataset is 32229 rows and 24 columns. Since we are working with real-life datasets imperfections are expected and this was addressed in the data preparation steps.

The major columns we will be using with this dataset are the date, time, and *MedTemp* columns. The description of the dataset is shown in Table 1 below:

Variable name	Variable Format	Description
DATE	Timestamp	Timestamp for daily data
LOCATION	Geolocation	States in Australia
MIN TEMP	Numerical	The maximum temperature reached in a day
MAX TEMP	Numerical	The minimum temperature reached in a day
MEDTEMP	Numerical	Average temperature

Table 1: The dataset

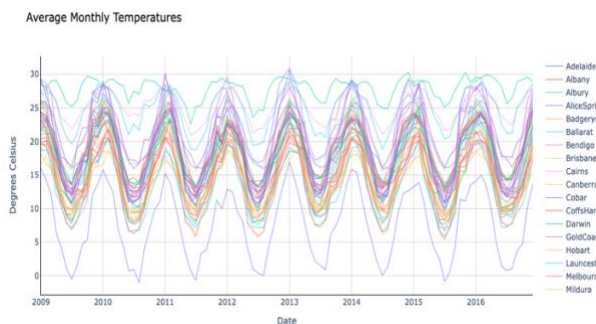
The rest of the columns are not utilized for the processing therefore we do not add them to our table. Check appendix - B

## V. PRELIMINARY DATA ANALYSIS

This section covers the analysis of the data we intend to plot and understand the dataset. This section contains a time series plot, and correlation heatmap, finds the distribution of the data, and checks for outliers in the data by plotting box plots. Check appendix - C

### 1. Time Series Plot

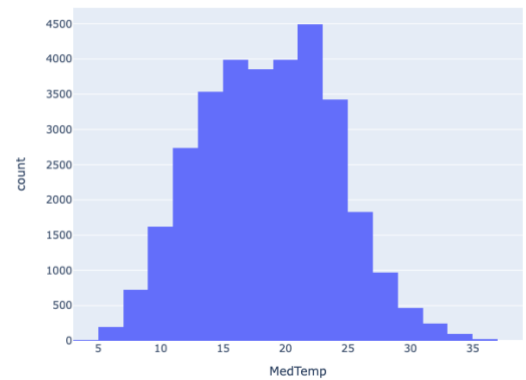
A time series plot is a graphical representation to understand the Average monthly temperature change from the year 2006 to 2017. refer image 1.0



*Figure: 1 Time series data for Average monthly temperature*

### 2. Histogram

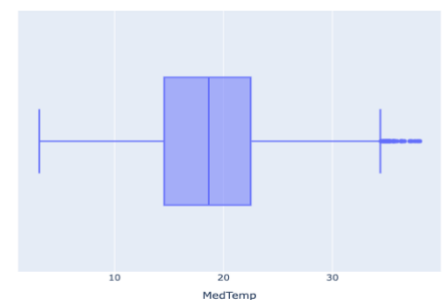
We intend to find the distribution by plotting a histogram. This will give us a better understanding of the dataset and if the data has a gaussian distribution.



*Figure: 2 Distribution plot of MedTemp*

### 3. Box Plot

The box plots are a good indicator of outliers in the dataset. This will give us a summary of the data. Refer figure 3



*Figure: 3 Box plot for MedTemp*

### 4. Scatter Plots

We intend to plot scatter plots against MedTemp and MinTemp and vice versa this helps us to understand the correlation of the data. refer to figures 4 and 5 .

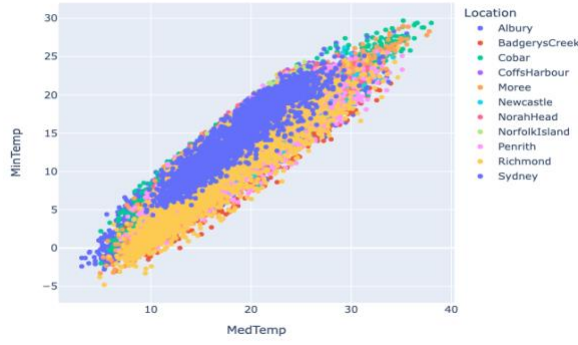


Figure 4 Scatter plot for MinTemp vs MedTemp

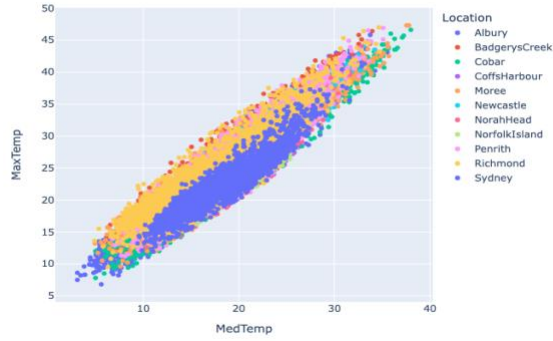


Figure 5 Scatter plot for MaxTemp and MedTemp

## 5. Correlation Heatmap

A correlation. A heatmap is a graphical representation of a correlation matrix that can represent a correlation between different variables. It can be used to understand the linear relationship between several variables in a dataset. We can use this technique to drop out highly correlated variables in the dataset refer to figure 6

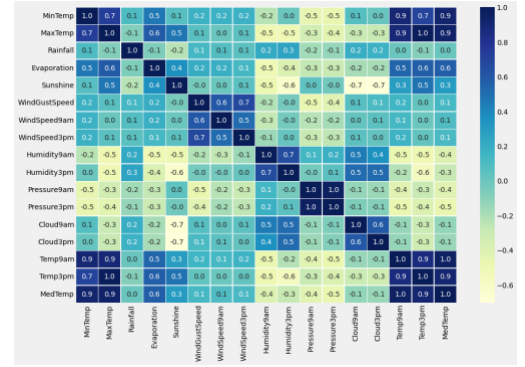


Figure: 6 Correlation heat map

## VI. Data Preparation

### 1. Removing Null Values and Duplicates Values

This is one of the important steps in preparing the data. We intend to check and verify every step of the code and eliminate any null or duplicate values which might be generated. By doing this before starting any learning it is wise to check and remove anomalies in the data. This will drastically help the model to learn better and will not stumble into an error. The Pandas library was used to tackle this problem.

### 2. Converting Timestamps

When doing RNN we are working with sequential data it is particularly important to transform the Dates to machine-readable format even though this entirely depends on the dataset. Pandas can be used to transform dates into a machine-understandable format.

Time Stamps	Time Stamps
20081201	2008-12
20091302	2009-13
20170620	2017-06

*Table:2 Converting time stamps*

### 3. Min-Max Scaling

Another crucial step before training the model is scaling the data into a certain range. that is exactly a min-max scalar. It can convert any value and put it in a range of 0 to 1 or -1 to 1 depending on the user input. With this dataset, we scale the data from -1 to 1. This will in turn help the model to understand and learn better. Min Min-scalar can be imported from the Sklearn library.

### 4. Reshaping

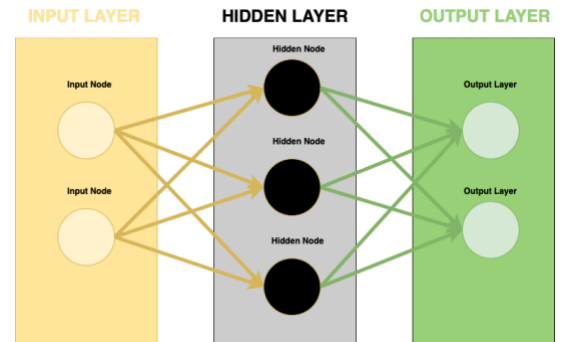
Like CNN and a few other networks RNN only takes 3D data therefore reshaping needs to be done before fitting to avoid errors. The input is reshaped by the start date followed by the total number of values and timestep and finally with 1. NumPy library can help us to shape the array.

## VII. Background: RNN, LSTM AND GRU

### 1. Simple recurrent neural network

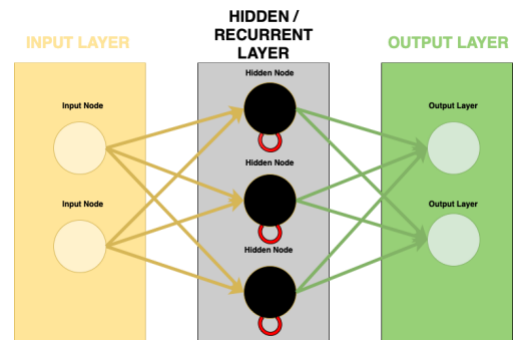
Modelling and forecasting sequential data can be particularly challenging and time-consuming with regular machine learning and Artificial Neural Network (ANN) models. Fortunately, a particular type of ANN can solve the problem of forecasting time series data with precision. The structure of the RNN is remarkably

like the structure of a typical feed-forward neural network but with a slight difference.



*Figure 7 simple FNN architecture*

The above figure depicts a simple feed-forward neural network. Note that a typical FNN can have any number of input nodes, output nodes and hidden nodes. The above architecture is in a (2-3-2) structure. The RNN has a feedback loop in the hidden nodes, enabling the learnt information to be passed back into the node several times. These hidden units are commonly called recurrent units which give Recurrent Neural Networks (RNN) their name. The architecture is given in figure 9 below.



*Figure 8 simple RNN architecture*

The information is processed in the recurrent unit for a predefined number of timesteps, A timestep can define the number of times the input



should go into the hidden layer, and the time stamp is adjusted based on the dataset and forecasting period to better results. In basic terms, if we have one timestep it is more or less of a regular hidden layer. If we have 9 timesteps the input will be processed 9 times and passed through the activation function.

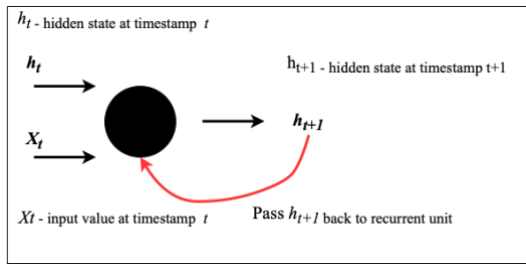


Figure 9 inside a recurrent node

All the input stamps at the initial are  $h_0$  and initialized with 0. The next hidden layer output is  $h_{t+1}$  which is again passed back into the recurrent unit this action carries on until the defined timestep is reached.

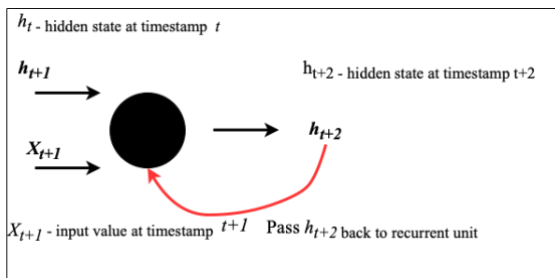


Figure 10 inside the recurrent unit

The previously processed input is passed back into the recurrent layer with the current input timestamp which is depicted in figure 11.

## 2. Long Short-Term Memory (LSTM)

One major drawback of regular RNNs is their inability to remember information from

earlier time steps in long sequences. For example, they forget the earlier words of a sentence when they are used for word prediction. This is due to the vanishing gradient problem. What this means is that as Back Propagation tries to update the weights for learning, a smaller gradient will contribute less and less to learning. This makes them unsuitable for forecasting far into the future.

LSTMs were created as solutions to this problem. They solve this problem using gates and an additional cell state. An LSTM has three gates namely; The Forget Gate, the Input Gate, and the Output Gate. These gates make use of a sigmoid activation function to selectively allow (filter) information through as shown in the figure below:

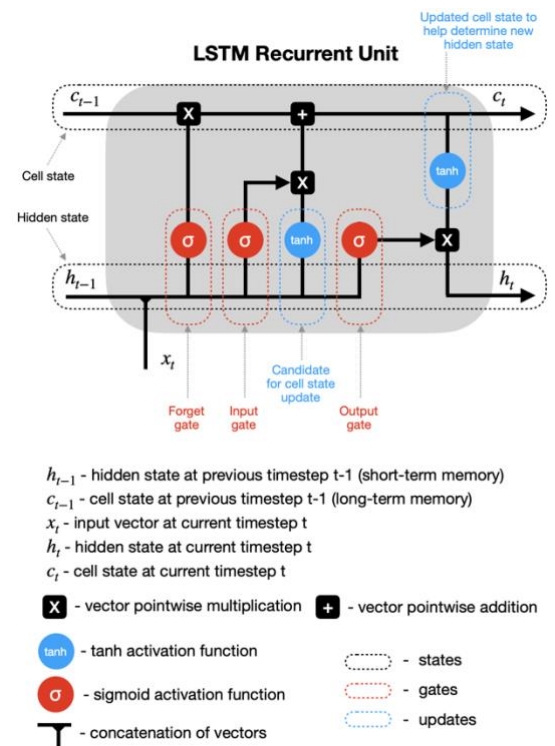


Figure: 11 LSTM Architecture

In LSTMs, the output is dependent on the previous hidden state ( $h_{t-1}$ ), the current input, and the previous cell state ( $c_{t-1}$ ). The network then uses



the gates mentioned above to determine how much information from each of the inputs is passed on to the next stage. It does this in the following steps:

- Forget Gate decides which information from the previous cell state is important and passes it on to the next cell state. Here, the previous hidden state and the current input are combined and passed through the gate which has a sigmoid activation function. The activation determines how much of the combination is kept. It could be any value between 0 and 1. This value is then multiplied by the previous cell state ( $c_{t-1}$ ) to determine how much of the previous cell state will be passed on to the current cell state.

- In this step, the combination of the input and the previous hidden state is passed through a tanh activation function to create a “new memory” vector. The input gate evaluates the hidden state and current input by passing them through a sigmoid activation function. The result of the tanh and the input gate sigmoid activation function is then pointwise multiplied. The input gate determines how much the previous hidden state and the current input influence the new cell state

- The output gate determines what the new hidden state will be. The new hidden state is a combination of information from the current cell state, the previous hidden state, and the current input. A combination of the previous hidden state

and the current input is passed through a sigmoid activation function and then pointwise multiplied with the newly updated cell state passed through a tanh activation function. The sigmoid activation function will, as is customary, filter the input and previous hidden state and determine how much of their combination will influence the new hidden state.

**NB:** In an LSTM network, the weights are shared across the network and then updated using Back Propagation Through Time (BPTT).

### 3. Gated Recurrent Units (GRU)

Like the LSTM method discussed previously, a GRU is an alternate approach to tackle the problem of vanishing or exploding gradients in a simple RNN. Like LSTMs, GRU also uses gates which can regulate the flow of information, these gates can learn which data in a sequence is important to keep or discard. By doing that it will use relevant information to perform predictions. Most state-of-the-art results achieved by Recurrent Neural Networks were done by LSTM or GRU ((Dey & Salem, Aug 2017). GRUs have many use cases like speech recognition and speech syntheses and text generation to name a few.

The GRU has two gates a reset gate( $r_t$ ) and an update gate( $z_t$ ). The update gate acts like the input and the update gate of an LSTM, it decides which information to discard, and which information should be kept. The reset gate decides how much past information to forget. Since there

are lesser gates or tensors compared with LSTM therefore, they are a little faster in training than LSTM. It is advisable to use both LSTM and GRU and determine which suits their use case. The architecture is shown in figure 12 below ("GRU Recurrent Neural Networks—A Smart Way to Predict Sequences in Python", 2022).

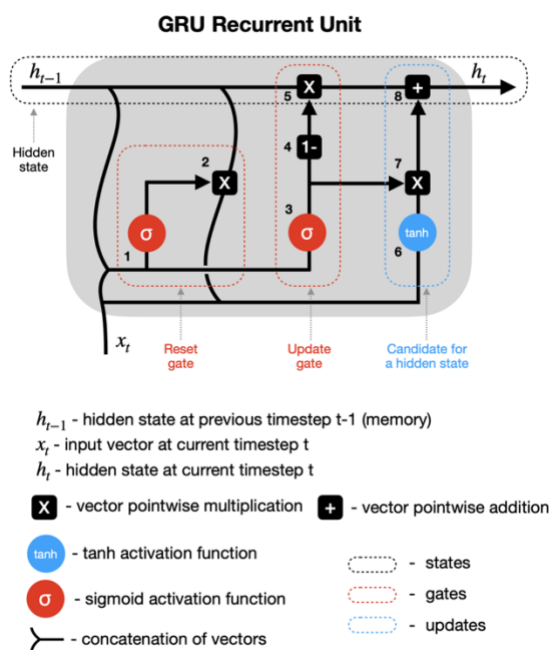


Figure 12: GRU Architecture

The general formula for the two reset gates ( $r_t$ ) and update gate ( $z_t$ )

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$

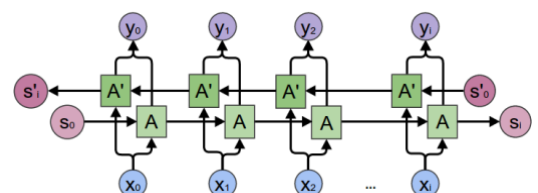
**Reset Gate** -- Previous hidden state ( $h_{t-1}$ ) and current input ( $x_t$ ) are combined (multiplied by weights and biases) and transmitted via a reset gate. Step one determines which values should be discarded (0), remembered (1), or just partially maintained because the sigmoid function has a range between 0 and 1. Using the outputs from step

one as a multiplier, step two resets the previous hidden state.

**Update Gate** -- Although steps three and one may appear to be similar, it's worth remembering that the weights and biases used to scale these vectors were different, resulting in a unique sigmoid output. So, in step four, after applying a sigmoid function to a combined vector, we subtract it from a vector that holds only 1s and multiply it by the prior concealed state (step five). That is a component of updating the concealed state with updated data.

**Hidden state** -- The outputs are coupled with current inputs ( $x_t$ ), multiplied by their respective weights, added biases, and then sent via a tanh activation function after step two resets a prior hidden state (step six). The new hidden state  $h_t$  is created by multiplying the hidden state candidate by the output of an update gate (step 7) and combining it with the previously modified hidden state  $h_{t-1}$ .

**NB:** For both LSTMs and GRUs, a bi-directional model was used. In essence, a bi-directional model allows each data point to be influenced by the data points before it and the ones after it. The diagram below shows a high-level overview of a bi-directional model.



*Figure 13: Bi-directional LSTM (Source: (Olah, 2015))*

#### 4. Model Evaluation Metrics

Training Data:

- 1) Loss
- 2) Mean Squared Error
- 3) Root Mean Square Error
- 4) Mean Absolute Error
- 5) Validation Loss
- 6) Validation Mean Squared Error
- 7) Validation Mean Absolute Error

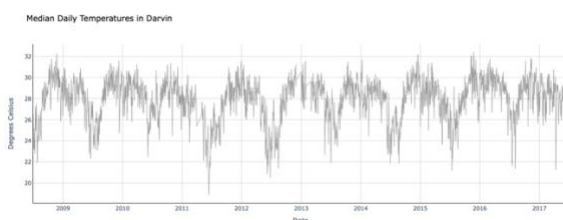
Testing Data:

- 1) Loss
- 2) Mean Squared Error
- 3) Mean Absolute Error

### VIII. Model Evaluation

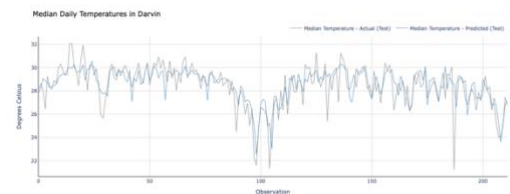
#### 1. Recurrent Neural network (RNN)

We intend to run our vanilla RNN model with the training data and after the model is prepared, we can use to model to predict the test data and check the results with supporting plots. Note we use one location from the dataset and predict the MedTemp for location. Below figure 13 shows the MedTemp for the location Darwin from the year 2018 to 2017. Refer appendix - D and E



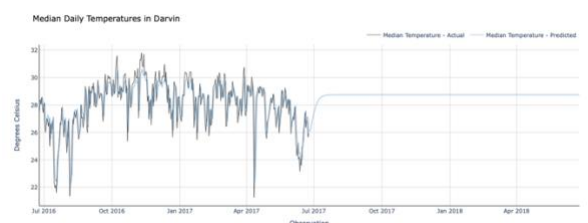
*Figure: 13 MedTemp in Darwin from the year 2006 to 2017*

The model predictions are used to then plot a figure on the test dataset therefore we can check how good our model predictions are checked in figure 14.



*Figure: 14 MedTemp prediction*

As the results show the prediction is quite good when done on test data but the main objective of our RNN model is to forecast the MedTemp far away in the future. We intend to add 365 days into the dataset and try to forecast the MedTemp in Darwin. The results are shown below in figure 15. Refer appendix - G



*Figure: 15 MedTemp Post 2017 predictions*

The figure quite clearly shows the limitations of the simple RNN network. It lacks the memory to predict far in the future and has many drawbacks; the vanishing gradient is one and the complex and slow training procedure is another. Its counterpart addresses these issues.

Recurrent Neural Network		
Algorithm parameters		
Trial	Train	Test
Total params	7	7
Epoch	40	40
Optimizer	Adam	Adam
Batch size	1	1
Results		
	Train	Test
No of samples	1665	465
Loss	0.0057	0.0057
Mean Squared Error	0.0054	0.0073
Mean Absolute Error	0.057	0.059
Root Mean Squared Error	0.0741	0.0871

Table 3 RNN result summary

In conclusion, although the RNN is good at predicting it forgets earlier datapoints when presented with a long sequence and takes a significantly long training time. Note that we have an encoder and a decoder layer in our model with no call-backs or dropout layers. The results were, however, close enough to the actual data

## 2. Long Short-Term Memory (LSTM)

We train our model with two types of LSTM. We then investigate to see which of the models gives us the best results and is computationally viable. Refer appendix – I

### 1. Bidirectional LSTM (BDLSTM)

Running 1000 epochs can be very time-consuming and not desktop-friendly especially when a simple LSTM model has been used. Therefore, the inspiration for BDLSTMs came from bidirectional RNN, which employs two distinct hidden layers to analyse sequence input in

both forward and backward directions. The two hidden layers are linked to the same output layer by BDLSTMs. In several areas, including phoneme classification and voice recognition, it has been demonstrated that bidirectional networks perform significantly better than unidirectional ones (Cui et al., 2020). The results of the BDLSTM are shown in the figure. Refer appendix- I

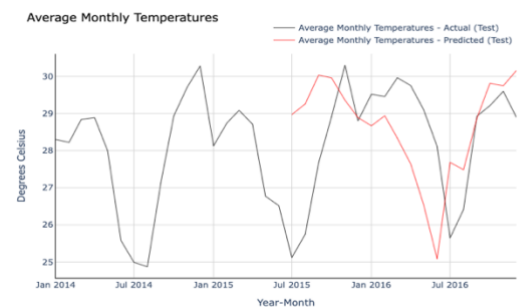


Figure 16 BDLSTM outputs for predicting MedTemp in Darwin

Bidirectional LSTM			
Algorithm parameters			
Trail	Train	Validation	Test
Total params	33,601	33,601	33,601
Epoch	1000	1000	1000
Optimizer	Adam	Adam	Adam
Batch size	1	1	1
Time Step	18	18	18
Results			
No of samples	1665		465
Loss	0.0003	0.363	0.4114
Mean Squared Error	0.00037	0.3632	0.4114
Mean Absolute Error	0.0143	0.4560	0.5230
Root Mean Squared Error	0.0193	0.6027	0.6414
Run Time	7.34 minutes		

*Table 4 Bidirectional LSTM result summary*

## 2. CuDNN Long Short-Term Memory (CuDNN LSTM)

The motivation behind using the CuDNN LSTM model is simply the lower computation time which is a vital factor while running a huge dataset. The CuDNN LSTMs take advantage of Nvidia GPUs (Cuda) to make running the epochs smoother and faster. The GPUs are the major contribution in doing so google Collab provides state-of-the-art GPU support for machine learning and deep learning. The strong parallel processing provided by the GPU can significantly reduce the runtime of our model. Below the figure are the outputs of the model. Refer appendix – J and K

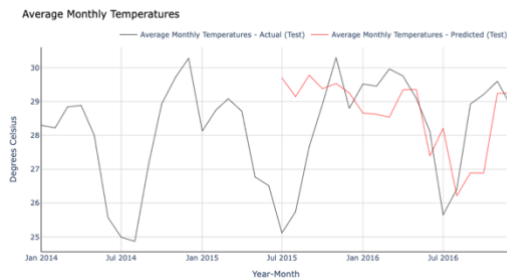


Figure 17: CuDNN LSTM outputs for predicting MedTemp in Darwin

CUDNN LSTM			
Algorithm parameters			
Trail	Train	Validation	Test
Total params	199,297	199,297	199,297
Epoch	1000	1000	1000
Optimizer	Adam	Adam	Adam
Batch size	1	1	1
Time Step	18	18	18
Results			
No of samples	1665	-	463
Loss	0.0119	0.4081	0.4793

<b>Mean Squared Error</b>	0.0119	0.4081	0.4793
<b>Mean Absolute Error</b>	0.0332	0.5026	0.5355
<b>Root Mean Squared Error</b>	0.8371	0.9057	0.9277
<b>Run Time</b>	1.3 minutes		

*Table: 5 CuDNN LSTM result summary*

The outputs clearly show that LSTM gates can help us store memory and use them to predict far away in the future it was also found that sometimes bidirectional LSTM can have better accuracy than the regular LSTM model but in the cost of computational power on the other hand CuDNN was able to take advantage of the hardware and produce outputs with good accuracy with less time taken this will be helpful on the type of dataset on which is deployed.

## 3. Gated Recurrent Unit (GRU)

Since the LSTM and GRU can hold memory with their gates which raises the question of how much memory it can hold and for how long? With the GRU model we intend to find it by sampling four separate locations and trying to predict. In theory, it can save an infinite amount of information but in our case, it was more than capable of handling and predicting for 4 separate locations with good accuracy as shown in figure 18 and table 6. Refer appendix- K and M

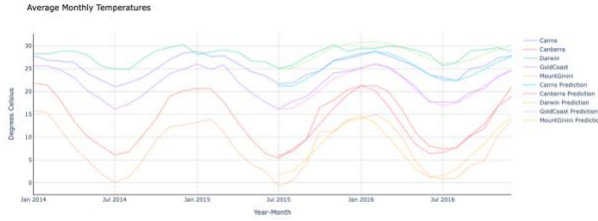


Figure 18 GRU Outputs

GRU			
Algorithm parameters			
Trail	Train	Validation	Test
Total params	25,601	25,601	25,601
Epoch	50	50	1000
Optimizer	Adam	Adam	Adam
Batch size	1	1	1
Time Step	18	18	18
Results			
No of samples	1665	465	
Loss	0.0036	0.0032	0.0082
Mean Squared Error	0.0036	0.0032	0.0082
Mean Absolute Error	0.046	0.0436	0.0701
Root Mean Squared Error	0.060	0.056	0.0904
Run Time	12.34 minutes		

Table 6 GRU Result Summary

It is evident that our model can produce satisfactory results when the data is well prepared although it takes time and computational power to produce the outputs.

## IX. Discussion

For this report, we investigated predicting some weather information (specifically daily median temperature) using Artificial Neural Networks. We tried to compare the predictive capabilities of three (3) different classes of ANNs, namely, vanilla RNN, LSTMs, and GRUs.

As expected, the vanilla RNN performed well for short-term forecasting as evidenced by the values obtained by the model evaluation metrics. We achieved remarkably low errors (mean squared error, mean absolute error, and root means squared error) for both the train and test portions of the evaluation. This indicates that the model didn't overfit and memorize our data. However, our model forecasted well just one day into the future; any attempts to predict anything farther than a day and the predictions were remarkably poor as shown in the plots. This is mainly because normal RNNs have very short memory (due to the vanishing/exploding gradient problem) and have difficulty "remembering" long sequences.

To mitigate this issue, we introduced the Long Short-Term Memory models (Bi-Directional LSTMs and Cu-DNN LSTMs). These models did not have the same disadvantage as the normal RNNs due to their use of gates to decide what portion of the data to "remember." The Bi-Directional LSTMs provide satisfactory results because each point in the recurrent unit is influenced by two "directions" (i.e., Each point in the sequence is influenced by both the points in its past and the points in its future). This helps them



make much better predictions than regular LSTMs. The Bi-Directional LSTM is particularly suited for forecasting. In the case of our dataset, it meant that the model learned how each datapoint in our dataset was affected by/affected the others. This is what led to better predictions.

The Cu-DNN LSTM was employed to show what could be possible with faster computations. It showed training times multiples faster than the Bi-Directional LSTMs with comparable results. They are a viable option to use when one wants to crunch through large datasets and has access to GPUs.

The use of the LSTMs and the GRU was meant to show the memory limitations of the regular RNN. This was really demonstrated in the GRUs. We used the GRUs to predict the median temperatures of four distinct locations over a period of 18 months and still got satisfactory results. This clearly demonstrates the longer memory capabilities of the GRU.

## X. Conclusion

In Conclusion, we have used RNN to predict some weather information in Australia. Our first RNN model could forecast the median temperature accurately although due to lack of memory, it was not able to predict far in the future. This problem was addressed by its counterparts LSTM and GRU. It was also found that using bidirectional LSTM and GRU gave us a good prediction, but the training time was remarkably high. When deployed with various datasets like the stock market, and text generation it was found that

GRU was faster than LSTM (training time), in terms of small datasets GRU outperformed LSTM in many instances. In terms of computational power, GRU was inferior to LSTM and the accuracy was also a little higher than in LSTM. Therefore it boils down to the user preference and the dataset (Yang et al., Jun 2020). All models can produce remarkable results when it comes to time-series data prediction.

Further work can be done by combining and predicting weather information such as humidity, wind speed and direction, light intensity, and atmospheric pressure. This kind of work can be important towards predicting natural disasters, renewable power generation, and the impact of global warming and pollution on the environment.

GRU Recurrent Neural Networks—A Smart Way to Predict Sequences in Python.

## XI. References

Bureau of Meteorology, Commonwealth of Australia, & CSIRO. (2014). *Science support for planning and safety in australian communities weather and climate forecasting for australia*. Australian Bureau of Meteorology.

Cui, Z., Ke, R., Pu, Z., & Wang, Y. (2020). Stacked bidirectional and unidirectional LSTM recurrent neural network for forecasting network-wide traffic states with missing values. *Transportation Research Part C: Emerging Technologies*, 118, 102674. <https://doi.org/10.1016/j.trc.2020.102674>



Dey, R., & Salem, F. M. (Aug 2017). Gate-variants of gated recurrent unit (GRU) neural networks. Paper presented at the 1597-1600. <https://doi.org/10.1109/MWSCAS.2017.8053243>

Polishchuk, B., Berko, A., Chyrun, L., Bublyk, M., & Schuchmann, V. (2021). The rain prediction in australia based big data analysis and machine learning technology. Paper presented at the , 1 97-100. <https://doi.org/10.1109/CSIT52700.2021.9648691>

Quinn, B., & Abdelfattah, E. Machine learning meteorologist can predict rain. Paper presented at the 57. <https://doi.org/10.1109/UEMCON47517.2019.8992997>

Raval, M., Sivashanmugam, P., Pham, V., Gohel, H., Kaushik, A., & Wan, Y. (2021). Automated predictive analytics tool for rainfall forecasting. *Scientific Reports*, 11(1), 17704. <https://doi.org/10.1038/s41598-021-95735-8>

Schroeter, B. J. E. (2016). Artificial neural networks in precipitation nowcasting: An australian case study. *Artificial neural network modelling* (pp. 325-339). Springer International Publishing. [https://doi.org/10.1007/978-3-319-28495-8\\_14](https://doi.org/10.1007/978-3-319-28495-8_14)

Yang, S., Yu, X., & Zhou, Y. (Jun 2020). (Jun 2020). LSTM and GRU neural network performance comparison study: Taking yelp review dataset as an example. Paper presented at the 98-101. <https://doi.org/10.1109/IWECAI50956.2020.00027>

Medium. (2022). Retrieved 10 July 2022, from <https://towardsdatascience.com/gru-recurrent-neural-networks-a-smart-way-to-predict-sequences-in-python-80864e4fe9f6>.

Dataset:

<https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package>

RNN\_Code:

<https://colab.research.google.com/drive/1jvY9cDw33-vI2Nmz1A5TzWUBjX-3kWNj?usp=sharing>

LSTM\_Code:

<https://colab.research.google.com/drive/1StuG4ewQcSn4h1NBdWQZLf1WQJFGMCFD?usp=sharing>

GRU\_Code:

<https://colab.research.google.com/drive/1ZX2GrXhG7EiiYztyR1eQJqCY9nvDZXB-?usp=sharing>

## XII. Appendices

### Appendix A– Importing libraries

```
from tensorflow import keras
print('Tensorflow/Keras: %s' % keras.__version__)
from keras.models import Sequential
from keras import Input
from keras.layers import Dense, SimpleRNN, LSTM, Dropout, GRU, Bidirectional, RepeatVector,
TimeDistributed, CuDNNLSTM
from tensorflow.keras.optimizers import Adam, SGD, RMSprop

import pandas as pd
print('pandas: %s' % pd.__version__)
import numpy as np
print('numpy: %s' % np.__version__)
import math

import sklearn
print('sklearn: %s' % sklearn.__version__)
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from keras.metrics import MeanAbsoluteError

import plotly
import plotly.express as px
import plotly.graph_objects as go
print('plotly: %s' % plotly.__version__)

import matplotlib.pyplot as plt

import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional

import math
from sklearn.metrics import mean_squared_error
```

## Appendix B– Importing Dataset

```
pd.options.display.max_columns=50
```

```
df = pd.read_csv("/content/weatherAUS.csv")
```

```
df=df[pd.isnull(df['MinTemp'])==False]
```

```
df=df[pd.isnull(df['MaxTemp'])==False]
```

```
df['MedTemp']=df[['MinTemp', 'MaxTemp']].median(axis=1)
```

```
df.head()
```

## Appendix C– Data Analysis

```
# Distribution plot
```

```
import plotly.express as px
```

```
df2 = df["MedTemp"]
```

```
fig = px.histogram(df2, x="MedTemp", nbins = 20)
```

```
fig.show()
```

```
#Box plot
```

```
import plotly.express as px
```

```
df2 = df["MedTemp"]
```

```
fig = px.box(df2, x="MedTemp")
```

```
fig.show()
```

```
#Scatter plot
```

```
import plotly.express as px
```

```
df2 = df
```

```
fig = px.scatter(df2, x="MedTemp", y = "MinTemp", color = "Location")
```

```
fig.show()
```

```

import plotly.express as px

df2 = df

fig = px.scatter(df2, x="MedTemp", y = "MaxTemp", color = "Location")

fig.show()


# corr heatmap

fig_dims=(15,10)

fig, ax = plt.subplots(figsize=fig_dims)

sns.heatmap(df.corr(),ax=ax, annot = True, linewidth=0.4,fmt = ".1f", cmap="YlGnBu")

plt.show()

```

## Appendix D– Data Preparation

```

# plotting data

dfCan=df[df['Location']=='Darwin'].copy()


fig = go.Figure()

fig.add_trace(go.Scatter(x=dfCan['Date'],
                        y=dfCan['MedTemp'],
                        mode='lines',
                        name='Median Temperature',
                        opacity=0.8,
                        line=dict(color='gray', width=1)
                        ))

fig.update_layout(dict(plot_bgcolor = 'white'))

fig.update_xaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey',

```

```
    zeroline=True, zerolinewidth=1, zerolinecolor='lightgrey',  
    showline=True, linewidth=1, linecolor='black',  
    title='Date'  
)
```

```
fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey',  
    zeroline=True, zerolinewidth=1, zerolinecolor='lightgrey',  
    showline=True, linewidth=1, linecolor='black',  
    title='Degrees Celsius'  
)
```

```
fig.update_layout(title=dict(text="Median Daily Temperatures in Darwin",  
    font=dict(color='black')))  
fig.show()
```

*# data prepration*

```
def prep_data(datain, time_step):
```

```
    y_indices = np.arange(start=time_step, stop=len(datain), step=time_step)
```

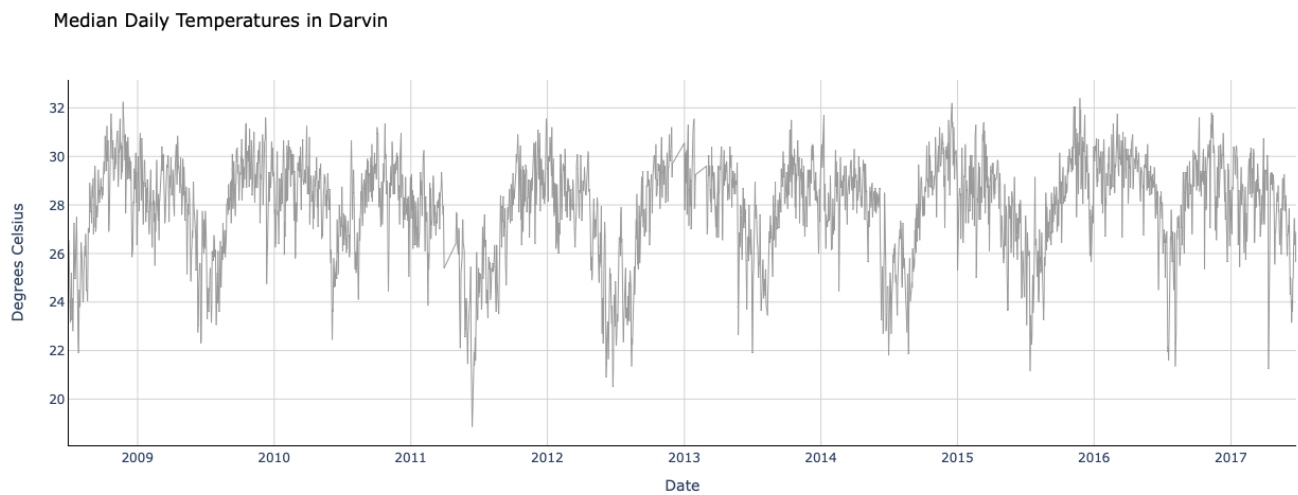
```
    y_tmp = datain[y_indices]
```

```
    rows_X = len(y_tmp)
```

```
    X_tmp = datain[range(time_step*rows_X)]
```

```
    X_tmp = np.reshape(X_tmp, (rows_X, time_step, 1))
```

```
    return X_tmp, y_tmp
```



Dravin Time series plot

## Appendix E– RNN Model

```
X=dfCan[['MedTemp']]
```

```
scaler = MinMaxScaler()
```

```
X_scaled=scaler.fit_transform(X)
```

```
train_data, test_data = train_test_split(X_scaled, test_size=0.2, shuffle=False)
```

```
time_step = 3
```

```
X_train, y_train = prep_data(train_data, time_step)
```

```
X_test, y_test = prep_data(test_data, time_step)
```

```
model = Sequential(name="First-RNN-Model")
```

```
model.add(Input(shape=(time_step,1), name='Input-Layer'))
```

```
model.add(SimpleRNN(units=1, activation='tanh', name='Hidden-Recurrent-Layer'))
```

```
model.add(Dense(units=1, activation='tanh', name='Hidden-Layer'))
```

```
model.add(Dense(units=1, activation='linear', name='Output-Layer'))
```

```
model.compile(optimizer='adam',
```

```
loss='mean_squared_error',  
metrics=['MeanSquaredError', 'MeanAbsoluteError']  
)
```

```
model.fit(X_train,  
          y_train,  
          batch_size=1,  
          epochs=40,  
          callbacks=None,  
          validation_split=0.0,  
          validation_data=(X_test, y_test),  
          shuffle=True,  
          )
```

```
pred_train = model.predict(X_train)
```

```
pred_test = model.predict(X_test)
```

```
print("")
```

```
print('----- Model Summary -----')
```

```
model.summary()
```

```
print("")
```

```
print('----- Weights and Biases -----')
```

```
print("Note, the last parameter in each layer is bias while the rest are weights")
```

```
print("")
```

```
for layer in model.layers:
```

```
    print(layer.name)
```

```
    for item in layer.get_weights():
```

```
        print(" ", item)
```



```

print("")

print('----- Evaluation on Training Data -----')

print("MSE: ", mean_squared_error(y_train, pred_train))

print("")

```

```

print('----- Evaluation on Test Data -----')

print("MSE: ", mean_squared_error(y_test, pred_test))

print("")

```

## Appendix F– RNN Model Outputs

```

fig = go.Figure()

fig.add_trace(go.Scatter(x=np.array(range(0,len(y_test))),

                        y=scaler.inverse_transform(y_test).flatten(),

                        mode='lines',

                        name='Median Temperature - Actual (Test)',

                        opacity=1,

                        line=dict(color='gray', width=1)

                        ))

fig.add_trace(go.Scatter(x=np.array(range(0,len(pred_test))),

                        y=scaler.inverse_transform(pred_test).flatten(),

                        mode='lines',

                        name='Median Temperature - Predicted (Test)',

                        opacity=1,

                        line=dict(color='steelblue', width=1)

                        ))

fig.update_layout(dict(plot_bgcolor = 'white'))

```

```

fig.update_xaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey',

```

```

zeroline=True, zerolinewidth=1, zerolinecolor='lightgrey',

showline=True, linewidth=1, linecolor='black',

title='Observation'

)

fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey',

zeroline=True, zerolinewidth=1, zerolinecolor='lightgrey',

showline=True, linewidth=1, linecolor='black',

title='Degrees Celsius'

)

fig.update_layout(title=dict(text="Median Daily Temperatures in Darwin",

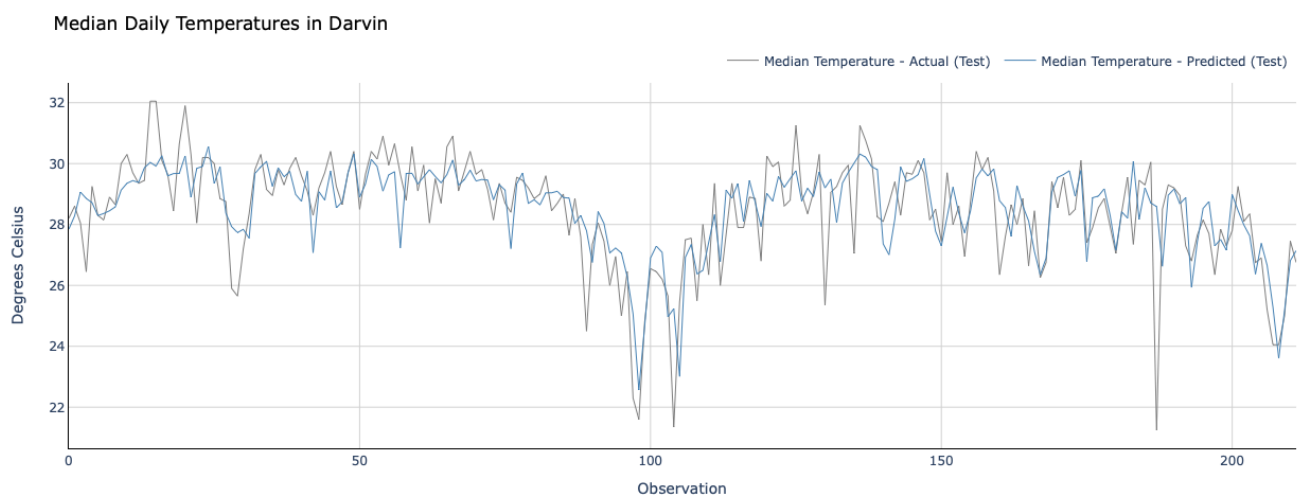
font=dict(color='black')),

legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="right", x=1)

)

fig.show()

```



## RNN Model Output

### Appendix G– RNN Limitation plots

```

X_every=dfCan[['MedTemp']]

X_every=scaler.transform(X_every)

```

```
for i in range(0, len(X_every)-time_step):
```

```
    if i==0:
```

```
        X_comb=X_every[i:i+time_step]
```

```
    else:
```

```
        X_comb=np.append(X_comb, X_every[i:i+time_step])
```

```
X_comb=np.reshape(X_comb, (math.floor(len(X_comb)/time_step), time_step, 1))
```

```
print(X_comb.shape)
```

```
dfCan['MedTemp_prediction'] = np.append(np.zeros(time_step),  
scaler.inverse_transform(model.predict(X_comb)))
```

```
fig = go.Figure()
```

```
fig.add_trace(go.Scatter(x=dfCan['Date'],
```

```
                        y=dfCan['MedTemp'],
```

```
                        mode='lines',
```

```
                        name='Median Temperature - Actual',
```

```
                        opacity=0.8,
```

```
                        line=dict(color='black', width=1)
```

```
                        ))
```

```
fig.add_trace(go.Scatter(x=dfCan['Date'],
```

```
                        y=dfCan['MedTemp_prediction'],
```

```
                        mode='lines',
```

```
                        name='Median Temperature - Predicted',
```

```
                        opacity=0.8,
```

```
                        line=dict(color='red', width=1)
```

```
                        ))
```

```
fig.update_layout(dict(plot_bgcolor = 'white'))
```

```
fig.update_xaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey',
                 zeroline=True, zerolinewidth=1, zerolinecolor='lightgrey',
                 showline=True, linewidth=1, linecolor='black',
                 title='Observation'
                )
```

```
fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey',
                 zeroline=True, zerolinewidth=1, zerolinecolor='lightgrey',
                 showline=True, linewidth=1, linecolor='black',
                 title='Degrees Celsius'
                )
```

```
fig.update_layout(title=dict(text="Median Daily Temperatures in Darwin",
                             font=dict(color='black')),
                  legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="right", x=1)
                 )
```

```
fig.show()
```

```
inputs=X_comb[-1:]
```

```
pred_list = []
```

```
for i in range(365):
```

```
    pred_list.append(list(model.predict(inputs)[0]))
```

```
    inputs = np.append(inputs[:,1:,:], [[pred_list[i]]], axis=1)
```

```
newdf=pd.DataFrame(pd.date_range(start='2017-06-26', periods=365, freq='D'), columns=['Date'])
```

```
newdf['MedTemp_prediction']=scaler.inverse_transform(pred_list)
```

```
dfCan2=pd.concat([dfCan, newdf], ignore_index=False, axis=0, sort=False)
```

```
fig = go.Figure()
```

```
fig.add_trace(go.Scatter(x=dfCan2['Date'][-730:],  
                        y=dfCan2['MedTemp'][-730:],  
                        mode='lines',  
                        name='Median Temperature - Actual',  
                        opacity=0.8,  
                        line=dict(color='black', width=1)  
                        ))
```

```
fig.add_trace(go.Scatter(x=dfCan2['Date'][-730:],  
                        y=dfCan2['MedTemp_prediction'][-730:],  
                        mode='lines',  
                        name='Median Temperature - Predicted',  
                        opacity=0.8,  
                        line=dict(color='steelblue', width=1)  
                        ))
```

```
fig.update_layout(dict(plot_bgcolor = 'white'))
```

```
fig.update_xaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey',  
                zeroline=True, zerolinewidth=1, zerolinecolor='lightgrey',  
                showline=True, linewidth=1, linecolor='black',  
                title='Observation'  
                )
```

```
fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey',
```

```

zeroline=True, zerolinewidth=1, zerolinecolor='lightgrey',

showline=True, linewidth=1, linecolor='black',

title='Degrees Celsius'

)

```

```

fig.update_layout(title=dict(text="Median Daily Temperatures in Darwin",

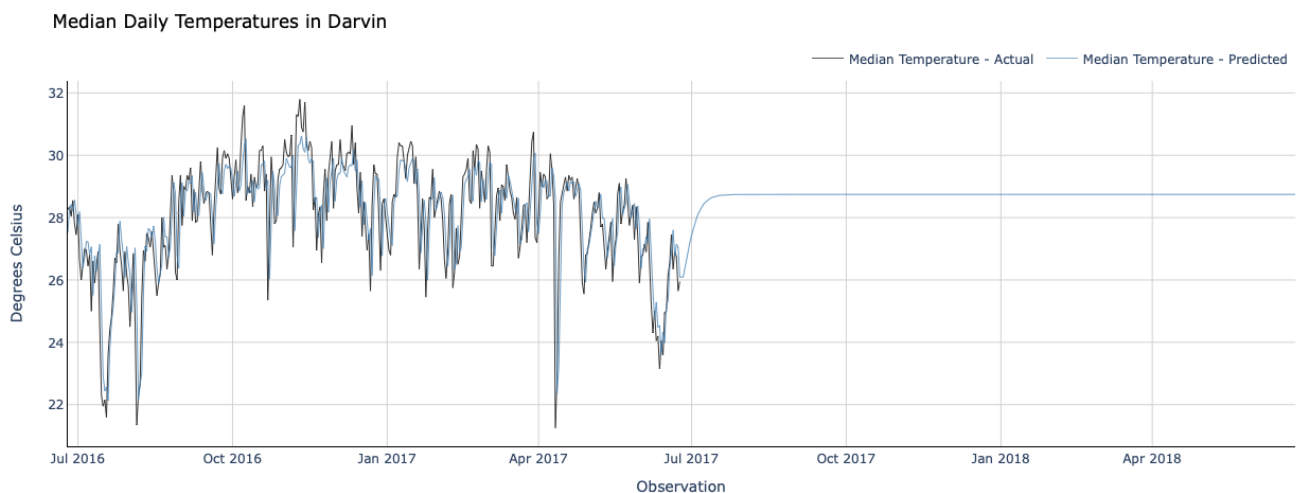
                             font=dict(color='black')),

                  legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="right", x=1)

)

fig.show()

```



RNN limitation plot

## Appendix H– LSTM Data preparation

```
pd.options.display.max_columns=150
```

```
df=pd.read_csv('/content/weatherAUS.csv', encoding='utf-8', usecols=['Date', 'Location', 'MinTemp', 'MaxTemp'])
```

```
df=df[pd.isnull(df['MinTemp'])==False]
```

```
df=df[pd.isnull(df['MaxTemp'])==False]
```

```
df['Year-Month'] = (pd.to_datetime(df['Date'], yearfirst=True)).dt.strftime('%Y-%m')
```

```
df['MedTemp'] = df[['MinTemp', 'MaxTemp']].median(axis=1)
```

```
df
```

```
df2 = df[['Location', 'Year-Month', 'MedTemp']].copy()
```

```
df2 = df2.groupby(['Location', 'Year-Month'], as_index=False).mean()
```

```
df2_pivot = df2.pivot(index='Location', columns='Year-Month')['MedTemp']
```

```
df2_pivot = df2_pivot.drop(['Dartmoor', 'Katherine', 'Melbourne', 'Nhil', 'Uluru'], axis=0)
```

```
df2_pivot = df2_pivot.drop(['2007-11', '2007-12', '2008-01', '2008-02', '2008-03', '2008-04', '2008-05', '2008-06', '2008-07', '2008-08', '2008-09', '2008-10', '2008-11', '2008-12', '2017-01', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06'], axis=1)
```

```
df2_pivot
```

```
df2_pivot['2011-04'] = (df2_pivot['2011-03'] + df2_pivot['2011-05'])/2
```

```
df2_pivot['2012-12'] = (df2_pivot['2012-11'] + df2_pivot['2013-01'])/2
```

```
df2_pivot['2013-02'] = (df2_pivot['2013-01'] + df2_pivot['2013-03'])/2
```

```
df2_pivot = df2_pivot.reindex(sorted(df2_pivot.columns), axis=1)
```

```
# plot
```

```
fig = go.Figure()
```

```
for location in df2_pivot.index:
```

```
    fig.add_trace(go.Scatter(x=df2_pivot.loc[location, :].index,
```

```
                            y=df2_pivot.loc[location, :].values,
```

```
                            mode='lines',
```

```
                            name=location,
```



```

        opacity=0.8,
        line=dict(width=1)
    ))

```

```

fig.update_layout(dict(plot_bgcolor = 'white'), showlegend=True)

```

```

fig.update_xaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey',
                 zeroline=True, zerolinewidth=1, zerolinecolor='lightgrey',
                 showline=True, linewidth=1, linecolor='black',
                 title='Date'
                )

```

```

fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey',
                 zeroline=True, zerolinewidth=1, zerolinecolor='lightgrey',
                 showline=True, linewidth=1, linecolor='black',
                 title='Degrees Celsius'
                )

```

```

fig.update_layout(title=dict(text="Average Monthly Temperatures", font=dict(color='black')))

```

```

fig.show()

```

## Appendix I– LSTM Bidirectional model

```

def shaping(datain, timestep):

```

```

    arr=datain.to_numpy().flatten()

```

```

    cnt=0

```

```

    for mth in range(0, len(datain.columns)-(2*timestep)+1):

```

```

        cnt=cnt+1

```

```

        X_start=mth

```

```
X_end=mth+timestep
```

```
Y_start=mth+timestep
```

```
Y_end=mth+2*timestep
```

```
if mth==0:
```

```
    X_comb=arr[X_start:X_end]
```

```
    Y_comb=arr[Y_start:Y_end]
```

```
else:
```

```
    X_comb=np.append(X_comb, arr[X_start:X_end])
```

```
    Y_comb=np.append(Y_comb, arr[Y_start:Y_end])
```

```
X_out=np.reshape(X_comb, (cnt, timestep, 1))
```

```
Y_out=np.reshape(Y_comb, (cnt, timestep, 1))
```

```
return X_out, Y_out
```

```
timestep=18
```

```
location='Darwin'
```

```
df_train=df2_pivot.iloc[:, 0:-2*timestep].copy()
```

```
df_test=df2_pivot.iloc[:, -2*timestep:].copy()
```

```
dfloc_train = df_train[df_train.index==location].copy()
```

```
dfloc_test = df_test[df_test.index==location].copy()
```

```
X_train, Y_train = shaping(datain=dfloc_train, timestep=timestep)
```

```
X_test, Y_test = shaping(datain=dfloc_test, timestep=timestep)
```

```
model = Sequential(name="LSTM-Model")
```

```

model.add(Input(shape=(X_train.shape[1],X_train.shape[2]), name='Input-Layer'))

model.add(Bidirectional(LSTM(units=32, activation='tanh', recurrent_activation='sigmoid', stateful=False),
name='Hidden-LSTM-Encoder-Layer'))

model.add(RepeatVector(Y_train.shape[1], name='Repeat-Vector-Layer'))

model.add(Bidirectional(LSTM(units=32, activation='tanh', recurrent_activation='sigmoid', stateful=False,
return_sequences=True), name='Hidden-LSTM-Decoder-Layer'))

model.add(TimeDistributed(Dense(units=1, activation='linear'), name='Output-Layer'))

```

```

model.compile(optimizer='adam',

              loss='mean_squared_error',

              metrics=['MeanSquaredError', 'MeanAbsoluteError'],

              loss_weights=None,

              weighted_metrics=None,

              run_eagerly=None,

              steps_per_execution=None

              )

```

```

history = model.fit(X_train,

                    Y_train,

                    batch_size=1,

                    epochs=1000,

                    validation_split=0.2,

                    validation_data=(X_test, y_test),

                    shuffle=True,

                    validation_freq=100,

                    use_multiprocessing=True,

                    )

```

```

pred_train = model.predict(X_train)

```

```
pred_test = model.predict(X_test)
```

```
print("")
```

```
print('----- Model Summary -----')
```

```
model.summary()
```

```
print("")
```

```
print('----- Weights and Biases -----')
```

```
print("Too many parameters to print but you can use the code provided if needed")
```

```
print("")
```

```
#for layer in model.layers:
```

```
# print(layer.name)
```

```
# for item in layer.get_weights():
```

```
# print(" ", item)
```

```
#print("")
```

```
print('----- Evaluation on Training Data -----')
```

```
for item in history.history:
```

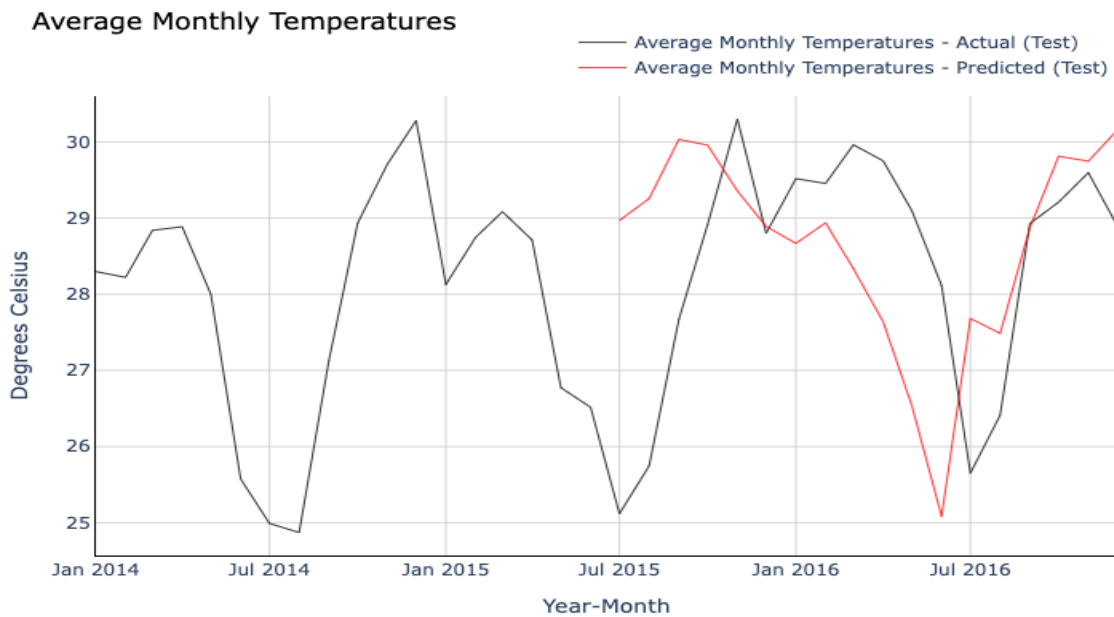
```
    print("Final", item, ":", history.history[item][-1])
```

```
print("")
```

```
print('----- Evaluation on Test Data -----')
```

```
results = model.evaluate(X_test, Y_test)
```

```
print("")
```



## Bidirectional LSTM Model Outputs

### Appendix J– LSTM CuDnn LSTM model

timestep=18

location='Darwin'

```
df_train=df2_pivot.iloc[:, 0:-2*timestep].copy()
```

```
df_test=df2_pivot.iloc[:, -2*timestep:].copy()
```

```
dfloc_train = df_train[df_train.index==location].copy()
```

```
dfloc_test = df_test[df_test.index==location].copy()
```

```
X_train, Y_train = shaping(datain=dfloc_train, timestep=timestep)
```

```
X_test, Y_test = shaping(datain=dfloc_test, timestep=timestep)
```

```
model = Sequential(name="LSTM-Model") # Model
```

```
model.add(Input(shape=(X_train.shape[1],X_train.shape[2]), name='Input-Layer'))
```

```
model.add(CuDNNLSTM(units=128, stateful=False, return_sequences=True, name='Hidden-LSTM-Encoder-Layer'))
```

```
#model.add(RepeatVector(Y_train.shape[1], name='Repeat-Vector-Layer'))
```

```
model.add(CuDNNLSTM(units=128, stateful=False, return_sequences=True, name='Hidden-LSTM-Decoder-Layer'))
```

```
model.add(TimeDistributed(Dense(units=1, activation='linear'), name='Output-Layer'))
```

```
model.compile(optimizer='adam',  
              loss='mean_squared_error',  
              metrics=['MeanSquaredError', 'MeanAbsoluteError'],  
              loss_weights=None,  
              weighted_metrics=None,  
              run_eagerly=None,  
              steps_per_execution=None  
            )
```

```
history = model.fit(X_train,  
                    Y_train,  
                    batch_size=1,  
                    epochs=1000,  
                    validation_split=0.2,  
                    validation_data=(X_test, y_test),  
                    shuffle=True,  
                    validation_freq=100,  
                    use_multiprocessing=True,  
                    )
```

```
pred_train = model.predict(X_train)
```

```
pred_test = model.predict(X_test)
```

```
print("")
```

```
print('----- Model Summary -----')
```





```
))
```

```
fig.add_trace(go.Scatter(x=np.array(dfloc_test.columns[-timestep:]),  
    y=pred_test.flatten(),  
    mode='lines',  
    name='Average Monthly Temperatures - Predicted (Test)',  
    opacity=0.8,  
    line=dict(color='red', width=1)  
))
```

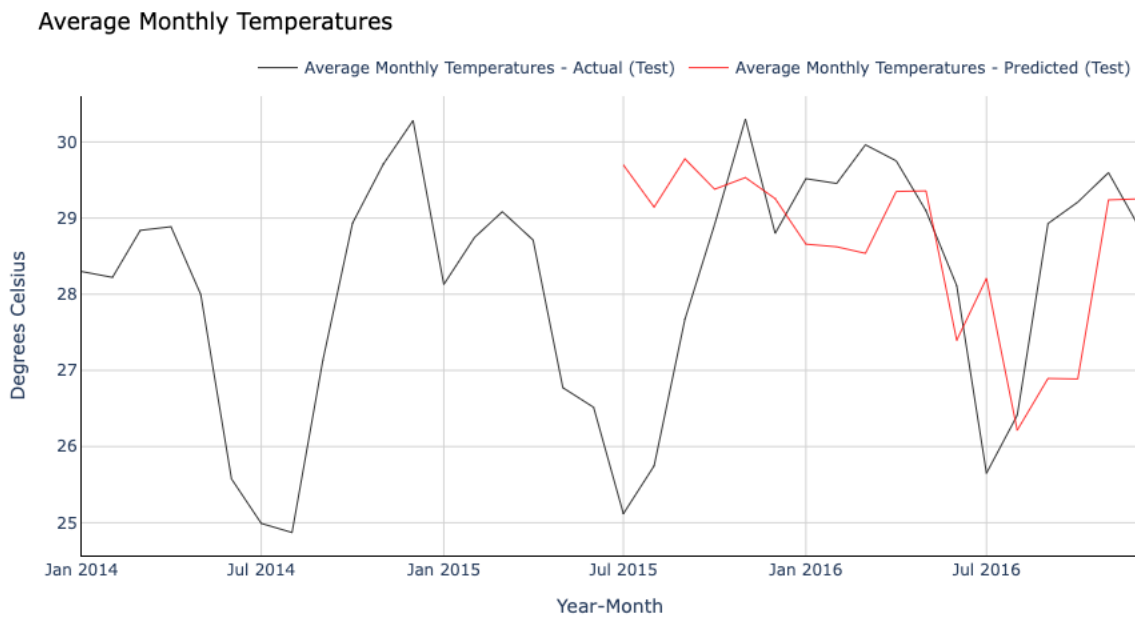
```
fig.update_layout(dict(plot_bgcolor = 'white'))
```

```
fig.update_xaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey',  
    zeroline=True, zerolinewidth=1, zerolinecolor='lightgrey',  
    showline=True, linewidth=1, linecolor='black',  
    title='Year-Month'  
)
```

```
fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey',  
    zeroline=True, zerolinewidth=1, zerolinecolor='lightgrey',  
    showline=True, linewidth=1, linecolor='black',  
    title='Degrees Celsius'  
)
```

```
fig.update_layout(title=dict(text="Average Monthly Temperatures", font=dict(color='black')),  
    legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="right", x=1)  
)
```

```
fig.show()
```



*CuDNN LSTM Model Output*

## Appendix L– GRU Model

```
def shaping(datain, timestep, scaler):
```

```
    for location in datain.index:
```

```
        datatmp = datain[datain.index==location].copy()
```

```
        arr=datatmp.to_numpy().flatten()
```

```
        arr_scaled=scaler.transform(arr.reshape(-1, 1)).flatten()
```

```
        cnt=0
```

```
        for mth in range(0, len(datatmp.columns)-(2*timestep)+1):
```

```
            cnt=cnt+1
```

```
            X_start=mth
```

```
            X_end=mth+timestep
```

```
            Y_start=mth+timestep
```

```
Y_end=mth+2*timestep
```

```
if mth==0:
```

```
    X_comb=arr_scaled[X_start:X_end]
```

```
    Y_comb=arr_scaled[Y_start:Y_end]
```

```
else:
```

```
    X_comb=np.append(X_comb, arr_scaled[X_start:X_end])
```

```
    Y_comb=np.append(Y_comb, arr_scaled[Y_start:Y_end])
```

```
X_loc=np.reshape(X_comb, (cnt, timestep, 1))
```

```
Y_loc=np.reshape(Y_comb, (cnt, timestep, 1))
```

```
if location==datain.index[0]:
```

```
    X_out=X_loc
```

```
    Y_out=Y_loc
```

```
else:
```

```
    X_out=np.concatenate((X_out, X_loc), axis=0)
```

```
    Y_out=np.concatenate((Y_out, Y_loc), axis=0)
```

```
return X_out, Y_out
```

```
timestep=30
```

```
scaler = MinMaxScaler(feature_range=(-1, 1))
```

```
df_train=df2_pivot.iloc[:, 0:-2*timestep].copy()
```

```
df_test=df2_pivot.iloc[:, -2*timestep:].copy()
```

```
scaler.fit(df_train.to_numpy().reshape(-1, 1))
```

```
X_train, Y_train = shaping(datain=df_train, timestep=timestep, scaler=scaler)
```

```
X_test, Y_test = shaping(datain=df_test, timestep=timestep, scaler=scaler)
```

```
model = Sequential(name="GRU-Model")
```

```
model.add(Input(shape=(X_train.shape[1],X_train.shape[2]), name='Input-Layer'))
```

```
model.add(Bidirectional(GRU(units=32, activation='tanh', recurrent_activation='sigmoid', stateful=False),  
name='Hidden-GRU-Encoder-Layer'))
```

```
model.add(RepeatVector(X_train.shape[1], name='Repeat-Vector-Layer'))
```

```
model.add(Bidirectional(GRU(units=32, activation='tanh', recurrent_activation='sigmoid', stateful=False,  
return_sequences=True), name='Hidden-GRU-Decoder-Layer'))
```

```
model.add(TimeDistributed(Dense(units=1, activation='linear'), name='Output-Layer'))
```

```
model.compile(optimizer='adam',
```

```
    loss='mean_squared_error',
```

```
    metrics=['MeanSquaredError', 'MeanAbsoluteError', 'RootMeanSquaredError']
```

```
)
```

```
history = model.fit(X_train,
```

```
    Y_train,
```

```
    batch_size=1,
```

```
    epochs=50,
```

```
    verbose=1,
```

```
    validation_split=0.2,
```

```
    validation_data=(X_test, y_test),
```

```
    validation_freq=10,
```

```
    use_multiprocessing=True
```

```
)
```

```
#pred_train = model.predict(X_train
```

```
pred_test = model.predict(X_test)
```

```
##### Step 7 - Print Performance Summary
```

```
print("")
```

```
print('----- Model Summary -----')
```

```
model.summary() # print model summary
```

```
print("")
```

```
print('----- Weights and Biases -----')
```

```
print("Too many parameters to print but you can use the code provided if needed")
```

```
print("")
```

```
#for layer in model.layers:
```

```
#    print(layer.name)
```

```
#    for item in layer.get_weights():
```

```
#        print(" ", item)
```

```
#print("")
```

```
print('----- Evaluation on Training Data -----')
```

```
for item in history.history:
```

```
    print("Final", item, ":", history.history[item][-1])
```

```
print("")
```

```
print('----- Evaluation on Test Data -----')
```

```
results = model.evaluate(X_test, Y_test)
```

```
print("")
```

## Appendix M– GRU Model Outputs

```
location=['Cairns', 'Canberra', 'Darwin', 'GoldCoast', 'MountGinini']
```

```
dfloc_test = df_test[df_test.index.isin(location)].copy()
```

```
X_test, Y_test = shaping(datain=dfloc_test, timestep=timestep, scaler=scaler)
```

```
pred_test = model.predict(X_test)
```

```
fig = go.Figure()
```

```
for location in dfloc_test.index:
```

```
    fig.add_trace(go.Scatter(x=dfloc_test.loc[location, :].index,  
                             y=dfloc_test.loc[location, :].values,  
                             mode='lines',  
                             name=location,  
                             opacity=0.8,  
                             line=dict(width=1)  
                             ))
```

```
for i in range(0, pred_test.shape[0]):
```

```
    fig.add_trace(go.Scatter(x=np.array(dfloc_test.columns[-timestep:]),  
                             y=scaler.inverse_transform(pred_test[i].reshape(-1,1)).flatten(),  
                             mode='lines',  
                             name=dfloc_test.index[i]+' Prediction',  
                             opacity=1,  
                             line=dict(width=2, dash='dot')  
                             ))
```

```
fig.update_layout(dict(plot_bgcolor = 'white'))
```

```
    zeroline=True, zerolinewidth=1, zerolinecolor='lightgrey',
```

```
    showline=True, linewidth=1, linecolor='black',
```

```

        title='Year-Month'
    )

fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='lightgrey',

                zeroline=True, zerolinewidth=1, zerolinecolor='lightgrey',

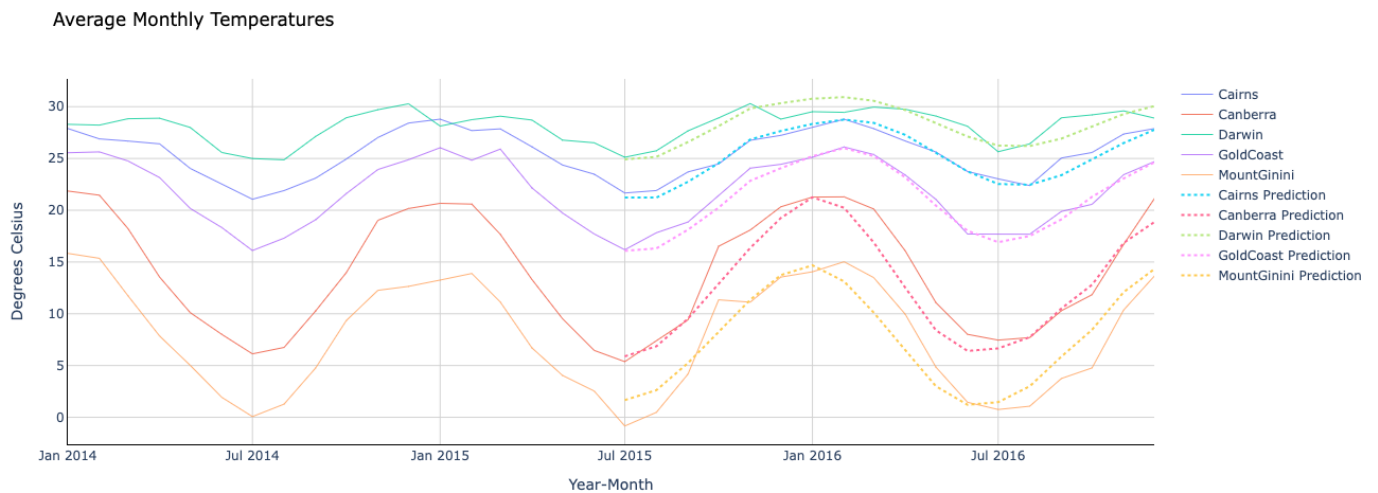
                showline=True, linewidth=1, linecolor='black',

                title='Degrees Celsius'
    )

fig.update_layout(title=dict(text="Average Monthly Temperatures", font=dict(color='black')))

fig.show()

```



*GRU Model Output*