FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

SEMESTER 2 (SESSION 2021/22)

BITI 3133

NEURAL NETWORK

GROUP PROJECT

LONG SHORT TERM MEMORY RECURRENT NEURAL NETWORK FOR WEATHER FORECASTING

LECTURER: Prof. Ts. Dr. Burhanuddin bin Mohd Aboobaider

MEMBERS: S1G1

1. **MUHAMMAD HAZIQ ISKANDAR BIN HANIZAL (B032010443)**
2. **NURSYAZA NISA BINTI ARFARIZAL (B032010244)**
3. **NURUL DALILA BINTI RAZALI (B032010467)**

ABSTRACT

In this paper, a two-layer recurrent network in exploring Long Short Term Memory (LSTM) is proposed to solve the large scale acoustic modelling in speech recognition. Long Short-Term Memory (LSTM) is a recurrent neural network (RNN) architecture that was created to more precisely simulate temporal sequences and their long-range relationships than ordinary RNNs. When evaluating moderately-sized models trained on a single machine, we recently shown that LSTM RNNs are more successful than DNNs and traditional RNNs for acoustic modelling. We provide the first distributed training of LSTM RNNs on a large cluster of computers utilising asynchronous stochastic gradient descent optimization. We show that a two-layer deep LSTM RNN with a linear recurrent projection layer on each LSTM layer can outperform state-of-the-art voice recognition.This design utilises model parameters more effectively than the others, converges rapidly, and beats a deep feed forward neural network with an order of magnitude more parameters. Long Short-Term Memory, LSTM, recurrent neural network, RNN, voice recognition, acoustic modelling are some of the terms used in this paper. In this paper, LSTM (RNN) will be shown in rain forecasting for temporal prediction of rainfall.

INTRODUCTION TO ARTIFICIAL NEURAL NETWORK (ANN) WITH R

Artificial Neural Networks (ANN) are algorithms that are based on brain function and are used to predict and forecast complex patterns. The Artificial Neural Network (ANN) is a deep learning approach derived from the concept of Biological Neural Networks in the human brain. An attempt to imitate the workings of the human brain led to the invention of ANN. Although they are not identical, the workings of ANN are very similar to those of biological neural networks. Only numeric and organised data are accepted by the ANN algorithm.

R is an open-source programming language that is widely used as a statistical software and data analysis. R generally comes with a Command-line interface, it is used as a leading tool for machine learning, statistics and data analysis. R is not only a statistical package but also allows us to integrate with other languages. Thus, you can easily interact with many data sources and statistical packages.

ANN FUNCTIONS AND R PROGRAMMING WITH EXAMPLES

1. ANN Activation Function

Artificial Neural Network (ANNs) are an important component that revolutionize the world by implementing the functionality of human barin in solving real life problems. ANNs uses activation functions to perform complex computations in hidden layers and transfer the result to output layer in order to enable output layer to deliver the output that we need to solve the problems. The purpose of activation functions are to introduce non-linearity in the neural network model.

Activation functions will convert the linear input signals of a node into non-linear output signals to facilitate learning of high order polynomials for deep network. Activation function are differentiable which helps them to function during backpropagation of neural network. If activation functions are not applied, the output signal of the ANN model will be a linear function, which is a polynomial of degree one. Eventhough it is easy to solve linear equations, they have limited complexity quotient and thus, is less powerful to learn complex functional mappings from data. To conclude, a ANNs model wil only be a linear regression model with limited abilities without activation functions.

2. Types of Activation Function with R Programming Example

Sigmoid Function

The sigmoid function is a non-linear activation function used primarily in feedforward neural networks. It is a differentiable real function, defined for real input values containing positive derivatives everywhere with a specific degree of smoothness. They appear in output layer of deep learning models and used for predicting probability-based outputs. Sigmoid functions is represented as:

$$f(x) = \frac{1}{(1+exp^{-x})} \quad - (1.4)$$

```
f <- function(x){1 / (1 + exp(-x))}
plot_activation_function(f, 'Logistic', c(-4,4))
```

Figure 1: R code for sigmoid function

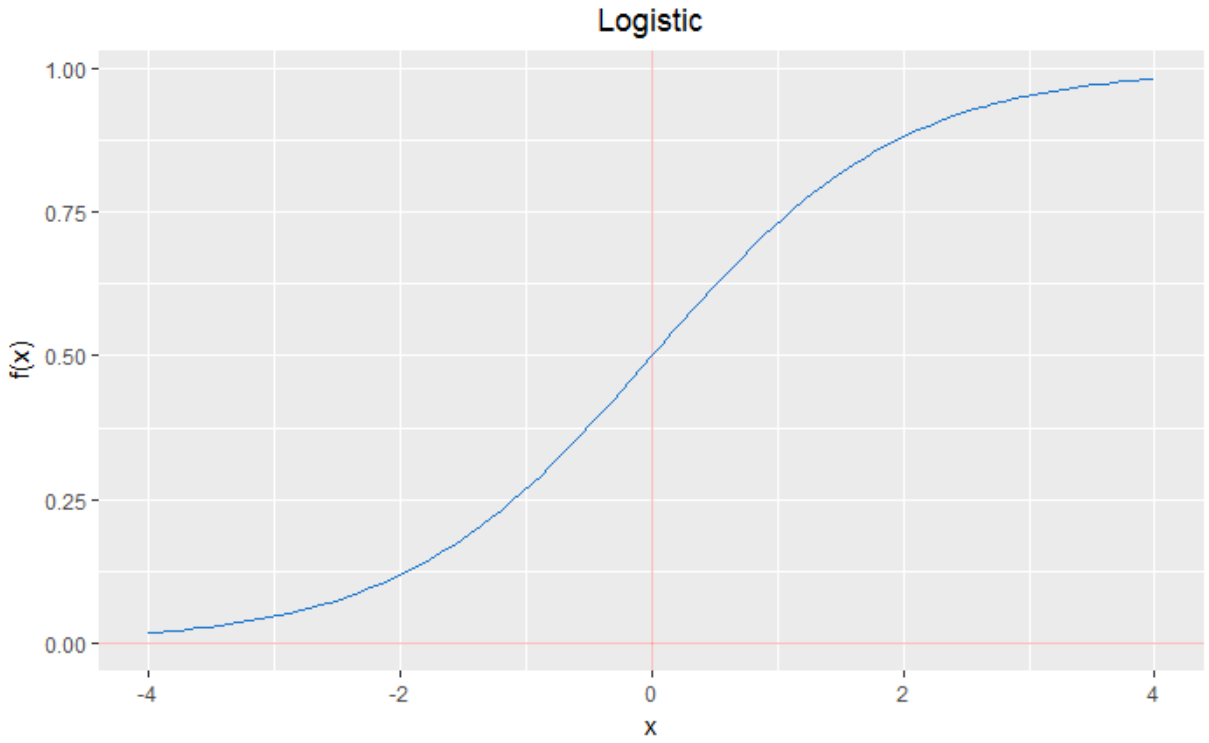


Figure 2: Graph of sigmoid activation function

Shown in figure 1 is the implementation of sigmoid activation function in RStudio. Figure 2 shows the graph plot of sigmoid activation function.

Hyperbolic Tangent Function (Tanh)

Tanh function is more extensively used than sigmoid function since it delivers better training performance for multilayer neural networks. The advantage of using Tanh functions is the production of zero-centered output, which supports backpropagation process. However, same like sigmoid function, Tanh function cannot solve vanishing gradient problem.In addition to that, Tanh function can only attain a gradient of 1 when the inpot value is 0. Due to that, some dead neurons can be produced during computation process. Tanh function can be represented as:

$$f(x) = \frac{exp\ (x_i)}{\Sigma j\ exp(x_j)} \text{ - (1.12)}$$

```
tanh_func <- function(x){tanh(x)}
plot_activation_function(tanh_func, 'TanH', c(-4,4))
```

Figure 3: R code for Tanh function

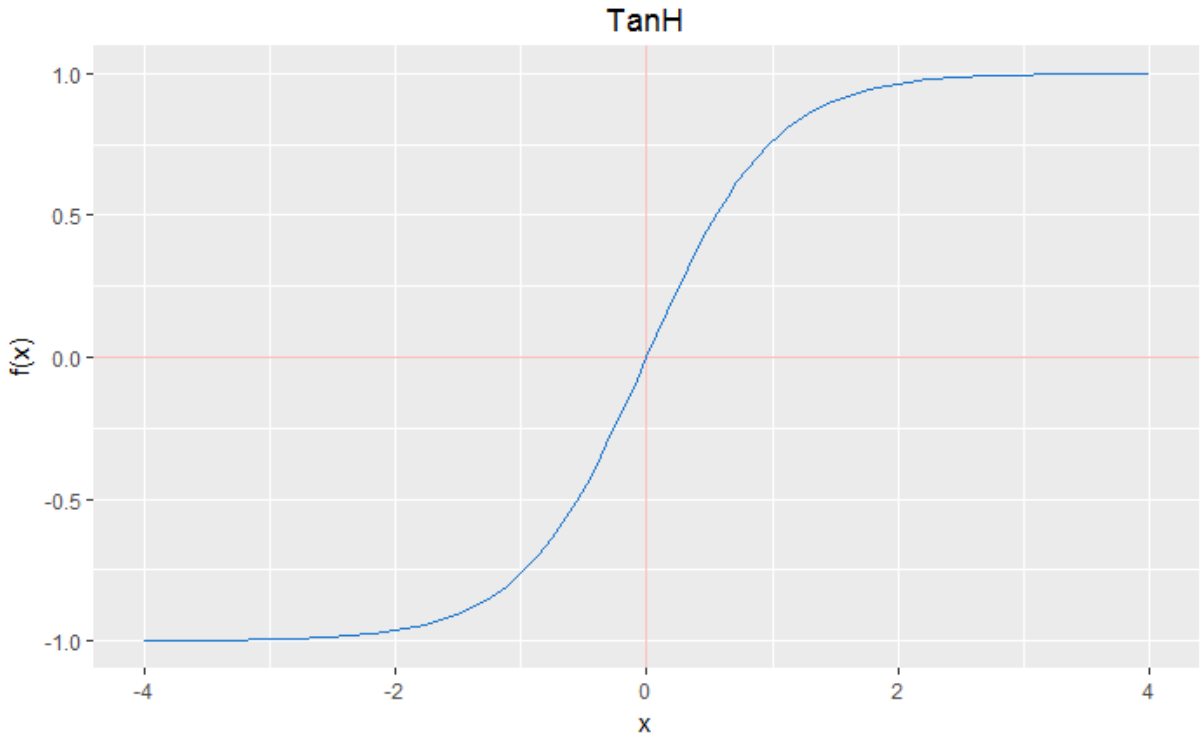


Figure 4:Graph of Tanh function

Shown in figure 3 is the implementation of Tanh activation function in RStudio. Figure 4 shows the graph plot of Tanh activation function.

Softsign Function

Softsign is an activation function used in neural network computing, primarily in regression computation problems. However, it is also used nowadays in DL based text-to-speech applications. The main difference between softsign function and Tanh function is that softsign function converges in a polynomial form whereas Tanh function converges exponentially. Softsign function is a quadratic polynomial, represented by:

$$f(x) = \frac{x}{|x| + 1} \quad - \quad (1.13)$$

```
soft_sign_func <- function(x){ x / (1 + abs(x))  }
plot_activation_function(soft_sign_func, 'Softsign', c(-4,4))
```

Figure 5 : R code for softsign function

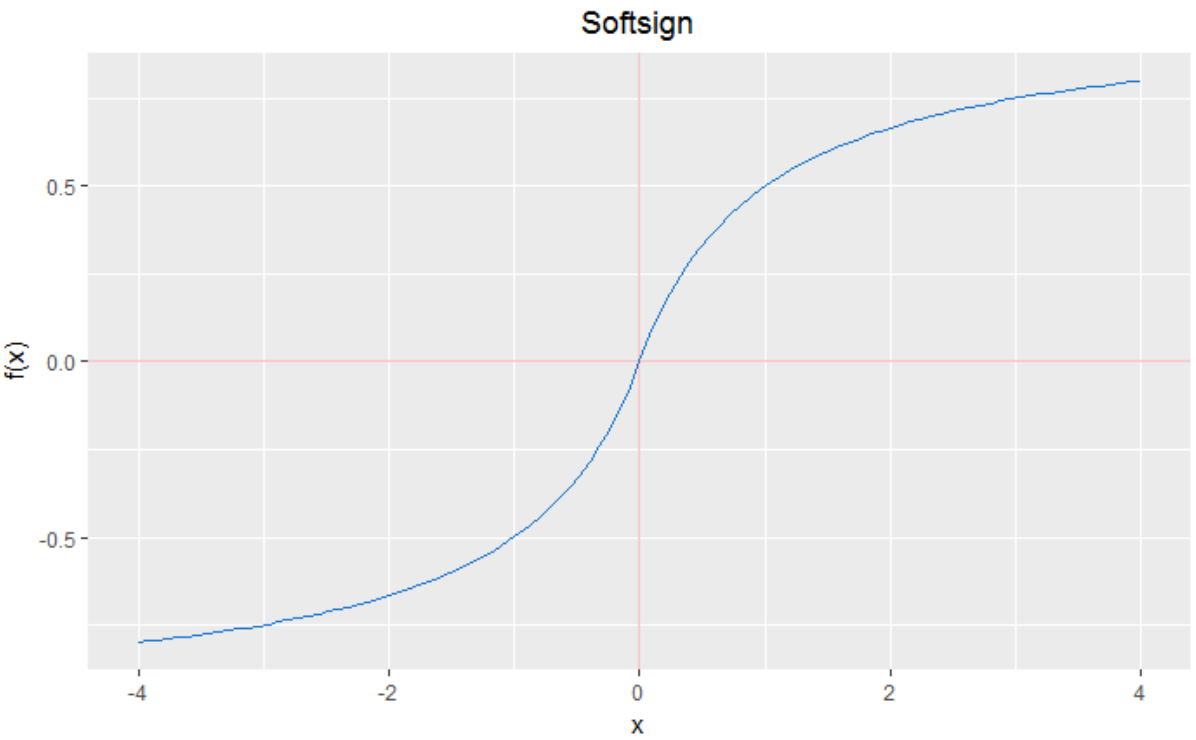


Figure 6: Graph of softsign function

Shown in figure 3 is the implementation of Tanh activation function in RStudio. Figure 4 shows the graph plot of Tanh activation function.

DETAIL DESCRIPTION ABOUT DATA, FLOW CHART, LEARNING PROCESS ETC

Description Data

The Weather dataset is obtained from the github website https://github.com/Eligijus112/Vilnius-weather-LSTM/blob/main/data/weather.csv where its context from https://openweathermap.org/api. The weather dataset comprises 294228 observations of 25 columns. The data was gathered every hour from 1990.01.01 to 2020.11.30 near the TV tower of Vilnius. Vilnius is not a large metropolis, and the TV tower is located within the city, so the temperature near the tower should be very similar to the temperature throughout the city. Below are the details of the variables:

- dt
- dt_iso
- timezone
- city_name
- lat
- lon
- temp
- feels_like
- temp_min
- temp_max
- pressure
- sea_level
- grnd_level
- humidity
- wind_speed
- wind_deg
- rain_1h
- rain_3h
- snow_1h
- snow_3h

- clouds_all
- weather_id
- weather_main
- weather_description
- weather_icon

Flow Chart

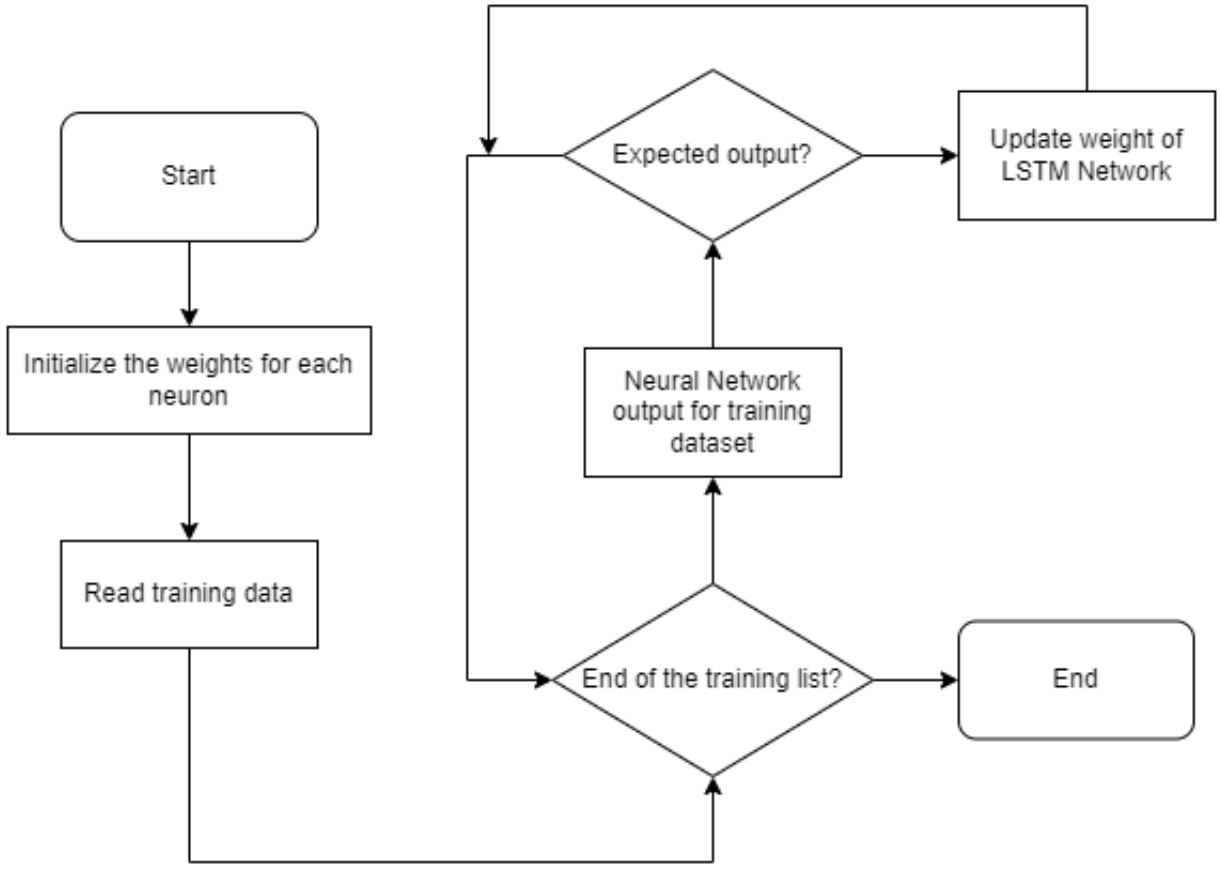


Figure 7: Flow Chart Long Short Term Memory (LSTM)

Figure 1 shows the flow chart for long short term memory for recurrent neural network. Firstly, start the program by loading the weather dataset and initializing the weights for each neuron in the network. Next, read the training data and wait for the training dataset to produce Neural Network output. If the output produced is the expected output, the program will update the weight of the LSTM. After updating the weight, it will go back to the training data to produce the next LSTM Neural Network output. This cycle continues until the data in the training list ends.

Learning Process of Neural Network

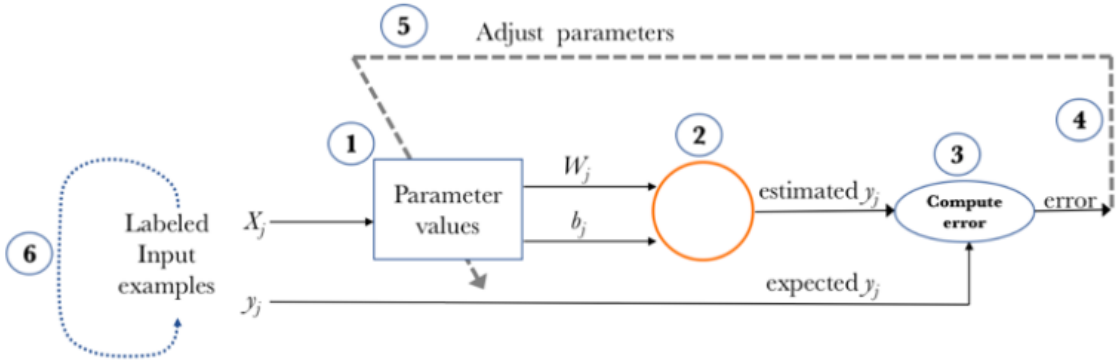


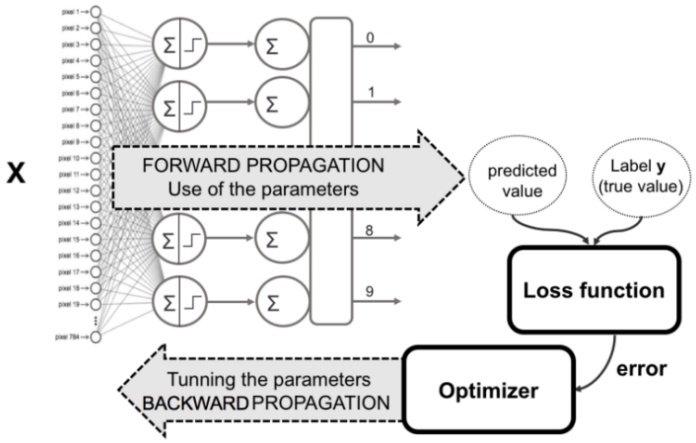
Figure 8: Learning Process of Neural Network

By learning the values of our parameters (weights wij and bj biases), it can be seen that the learning process in a neural network is an iterative process of "going and return" by the layers of neurons. The "going" represents information propagation forward, and the "return" represents information propagation backward.

The first phase of forward propagation occurs when the network is exposed to training data, which crosses the entire neural network to calculate predictions (labels). That is, routing the input data through the network in such a way that all neurons apply their transformation to the information received from the neurons of the previous layer before sending it to the neurons of the next layer. When the data has passed through all of the layers and all of the neurons have completed their calculations, the final layer will be reached with a label prediction result for those input examples.

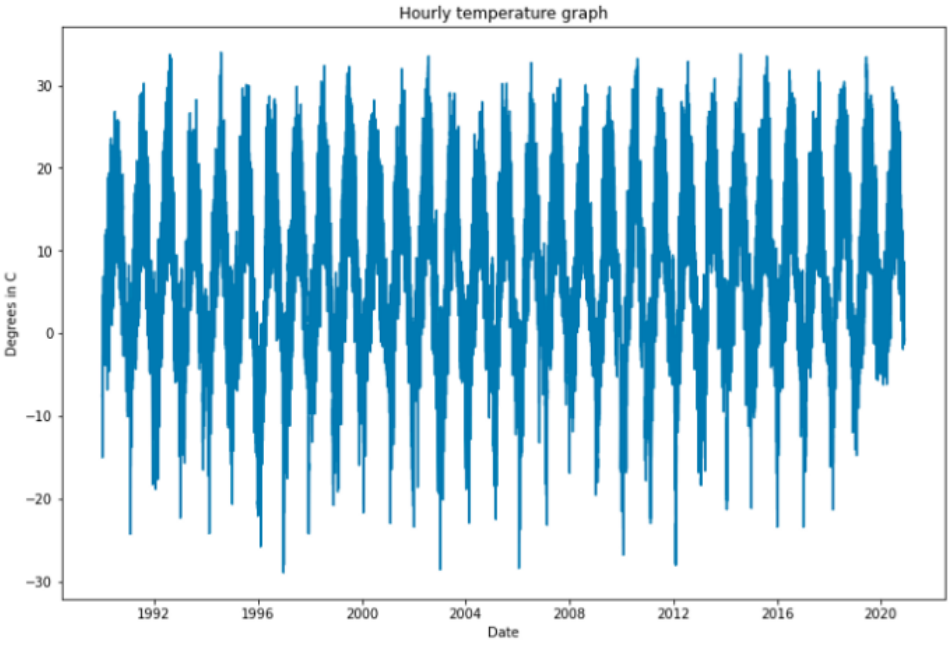
Following that, we'll use a loss function to estimate the loss (or error) and compare and measure how good or bad our prediction result was in comparison to the correct result (remember that we are in a supervised learning environment and we have the label that tells us the expected value). In an ideal world, our cost would be zero, with no difference between estimated and expected value. As a result, as the model is trained, the weights of the neurons' interconnections will be gradually adjusted until good predictions are obtained.

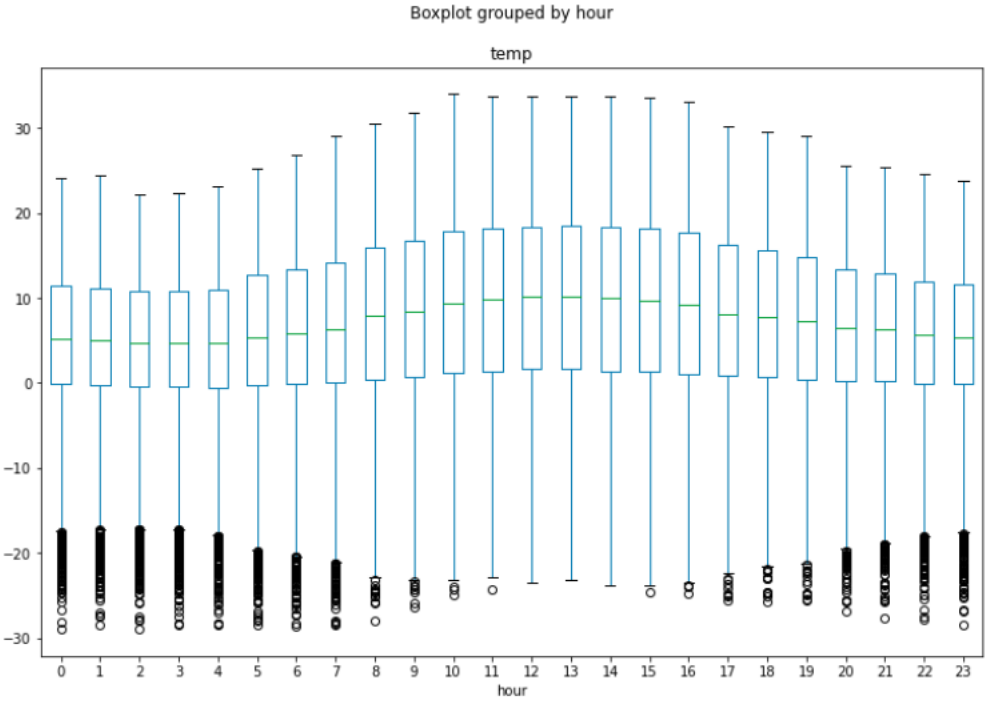
This information is propagated backwards once the loss has been calculated. As a result, it's called backpropagation. That loss information spreads from the output layer to all the neurons in the hidden layer that contribute directly to the output. However, based on the relative contribution of each neuron to the original output, the neurons of the hidden layer only receive a fraction of the total signal of the loss. This process is repeated layer by layer until all neurons in the network have received a loss signal describing their relative contribution to the overall loss.

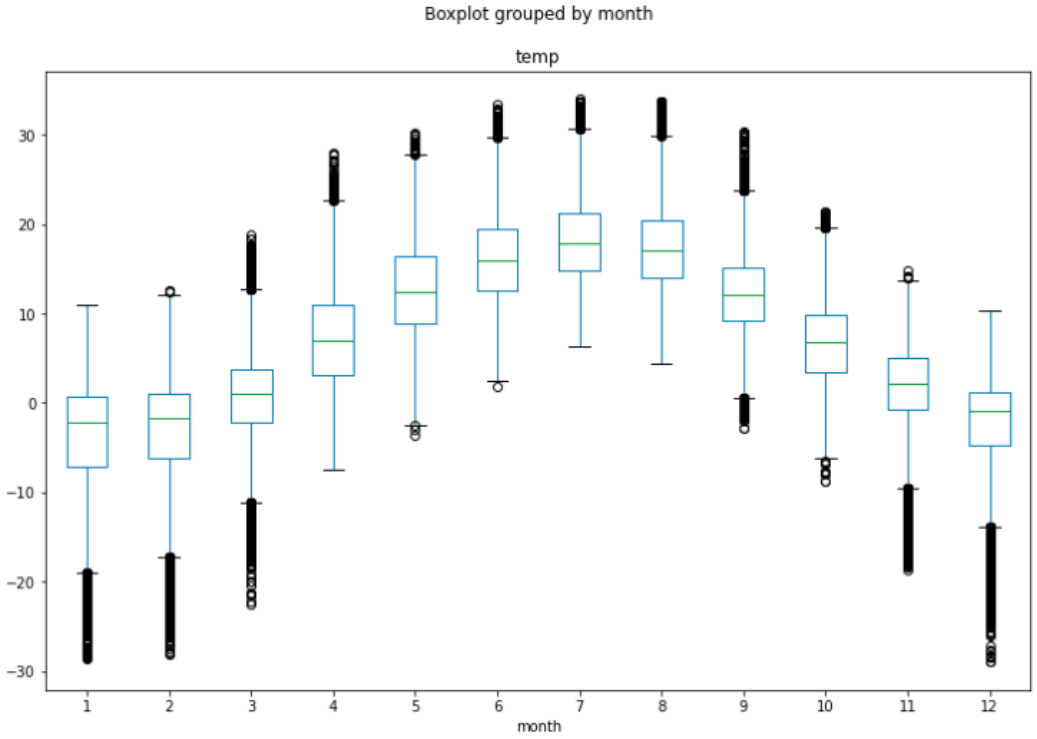


We can now adjust the weights of connections between neurons now that we have spread this information back. What we're doing is getting the loss as close to zero as possible the next time we use the network to predict something. We will use a technique known as gradient descent for this. This technique changes the weights in small increments by calculating the derivative (or gradient) of the loss function, which allows us to see which direction "to descend" towards the global minimum; in general, this is done in batches of data in successive iterations (epochs) of all the dataset that we pass to the network in each iteration.

DATA ANALYSIS WITH ANN MODELS AND THEIR SOLUTIONS






Boxplot grouped by month
temp
30
20
10
0
−10
−20
−30
1
2
3
4
5
6
7
8
9
10
11
12
month

EXPLANATION ON FINDINGS, FIGURES AND RESULTS

After splitting dataset into training data and testing data, we create the ANN object using model class. By using hyperparameters defined, we are going to forecast 1 hour ahead by using data from 48 hours back. Our model will have 10 number of neurons, 512 batch size, 0.001 learning rate and will run through 20 iterations.

After training the model using training data, we get training loss and validation loss as output. After that, we plot the graph of training loss against validation loss as shown in figure 10.

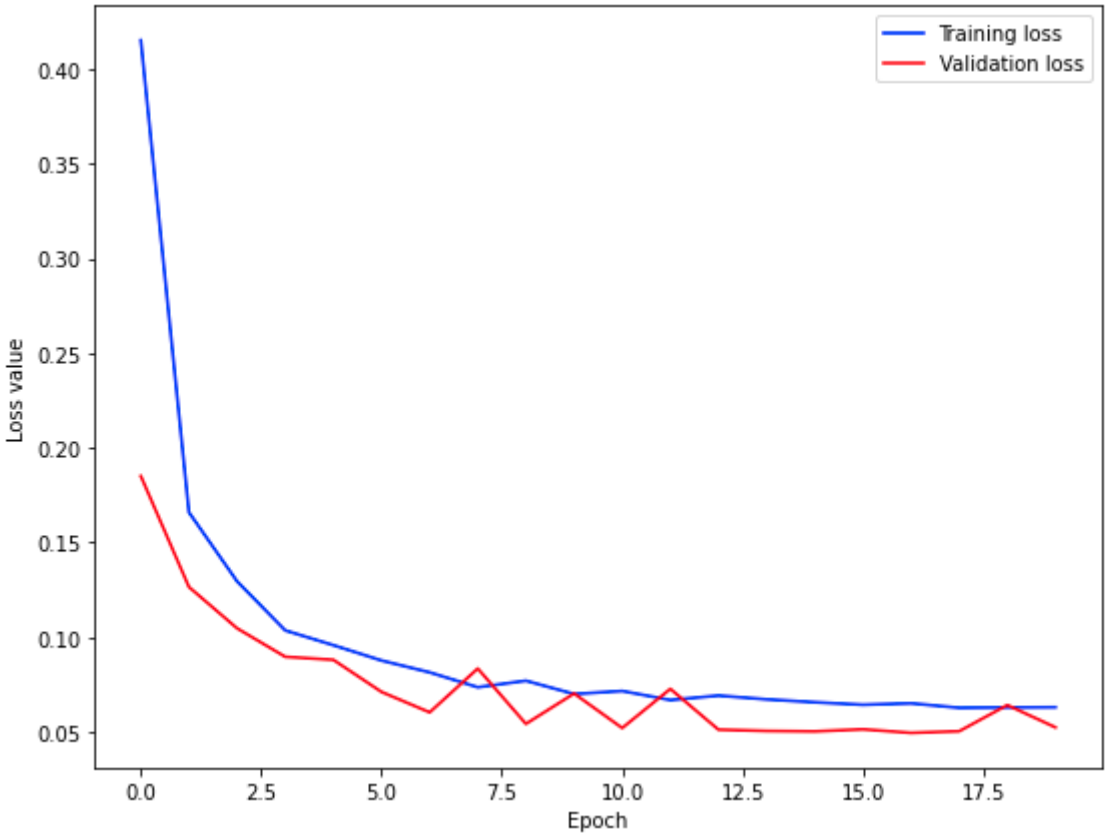


Figure 10: Graph of training loss against validation loss

With the trained model, we can forecast using testing data and compare the values with the original ones. We again plot the graph of training loss against validation loss as shown 11.

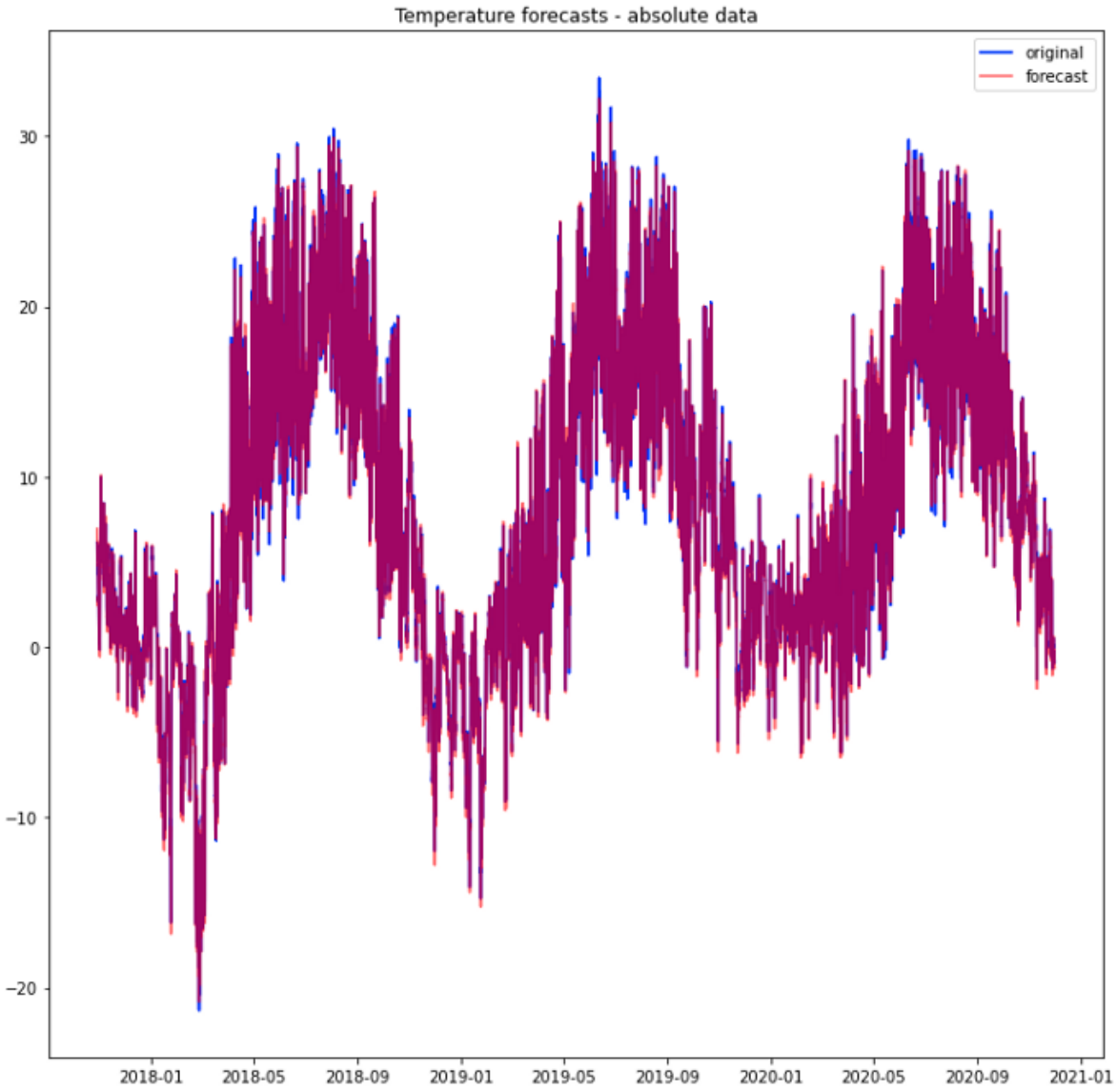


Figure 11: Graph of training loss against validation loss

The lines between training loss and validation loss are very clos to each other, they basically overlap. After that, we plot the distribution of absolute errors using graphviz and found that the value of absolute error is 0.34 degrees celsius and the average is 0.48 degrees celsius. Figure 12 shows the plot of error distribution.

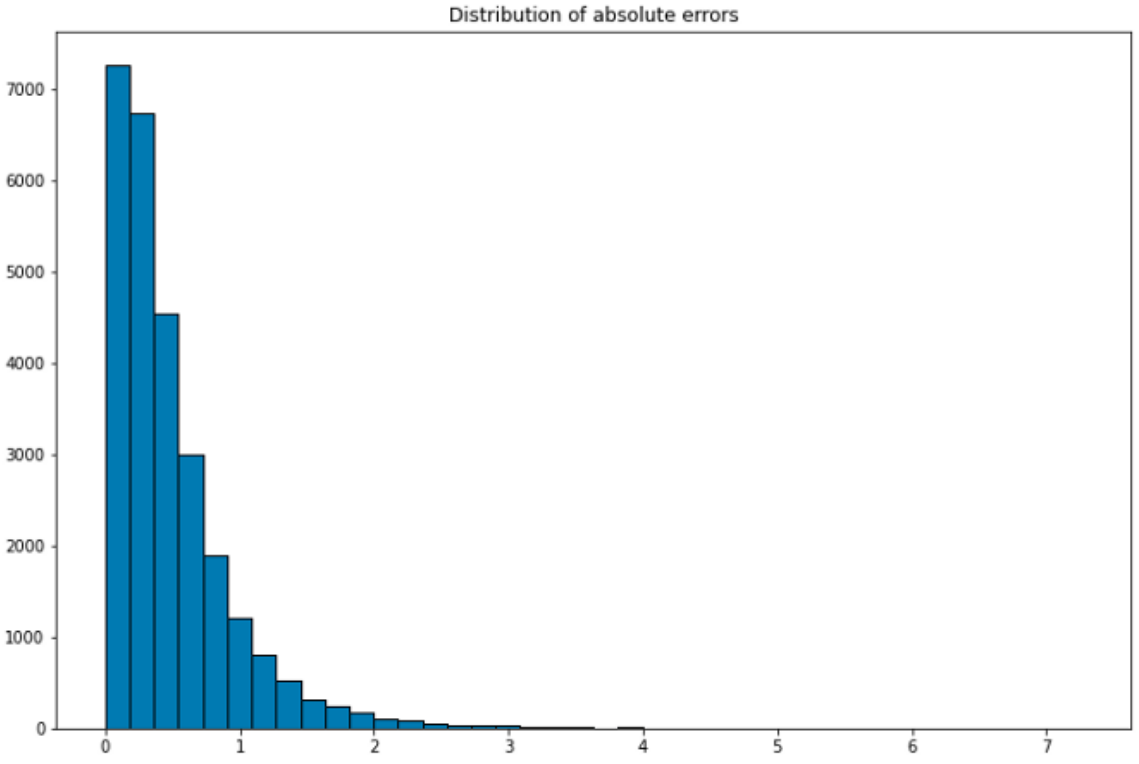


Figure 12: The graph of distribution of absolute error

In order to forecast 24 hours ahead, we just need to change the hyperparameters. After manipulating the parameters, we will forecast 24 hours ahead, using values from 168 hours from before. Next, we repeat the process of training and testing using changed hyperparameters. Figure 13 shows the losses in training for the updated parameters.

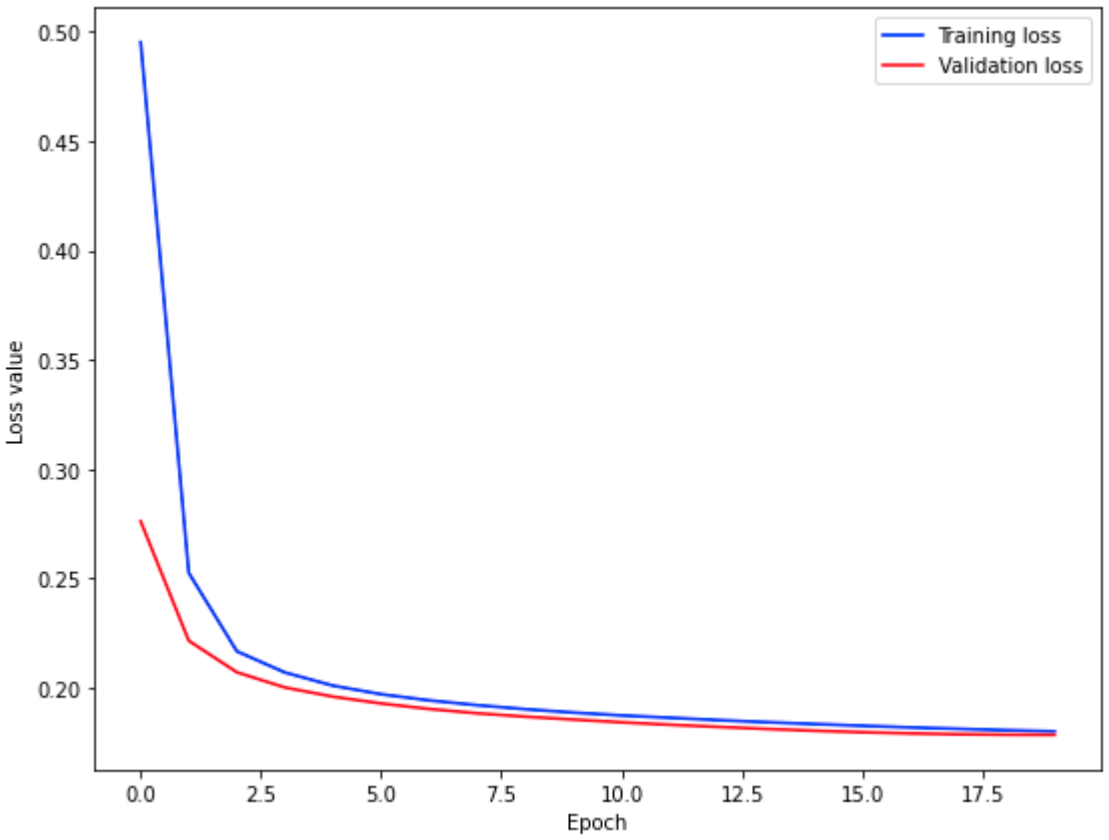


Figure 13: Graph of losses for the updated parameter

After using testing data on the updated trained model, we can inspect chunks within the 24 hours forecasting. Shown in figure 14 is the graph of multiple random chunks of 24 hours.

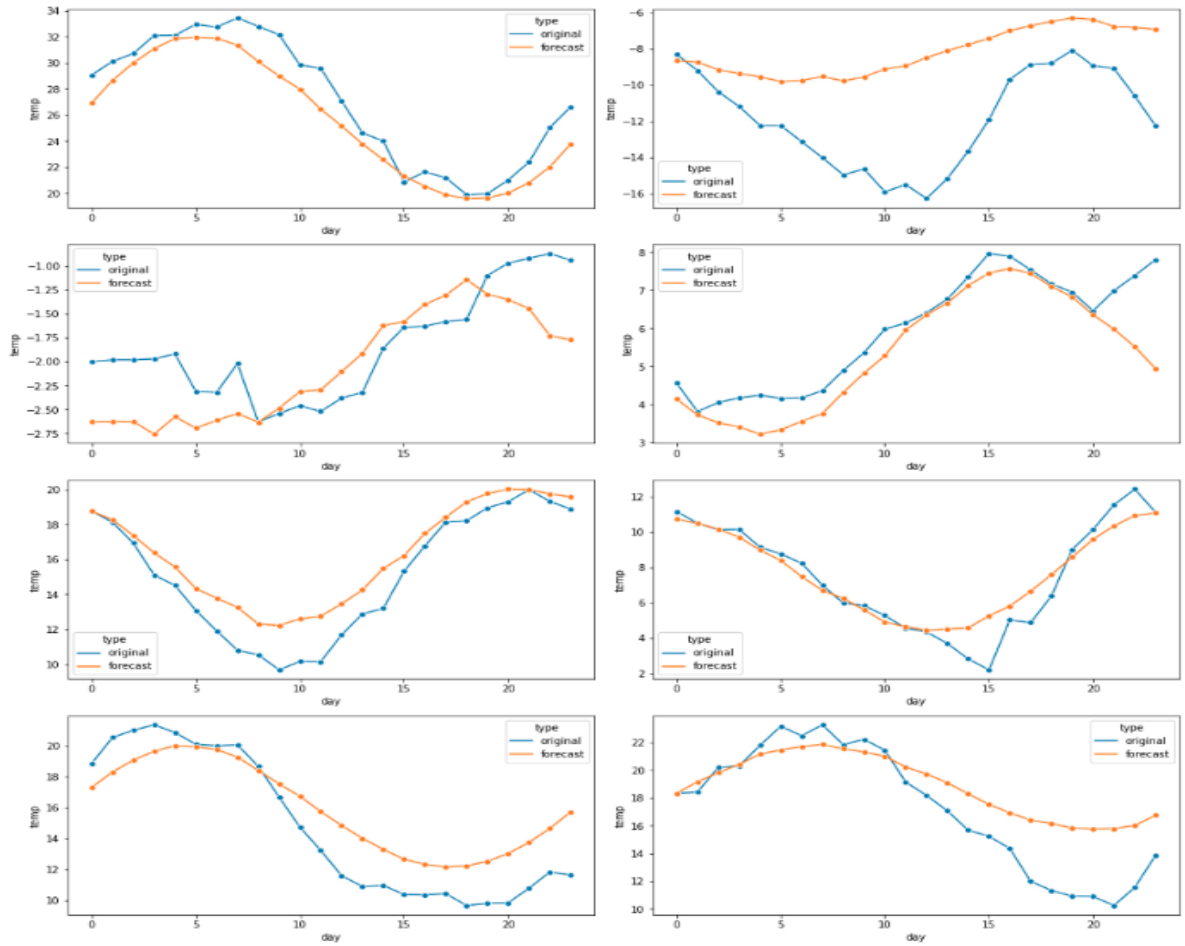


Figure 14: Graph of some random chunks of 24 hours chunk

CONCLUSION

In conclusion, when working with modelling and forecasting of the time series data we learn how to read, clean and augment the input data. We also learn how to select the hyperparameters for the lag and n steps ahead. Next, we also know how to select the hyperparameters for the deep learning model. Then, initiating the NNMultistepModel() class. Lastly, we also got to learn how to fit the model and forecast n_steps ahead.

REFERENCES

1. https://towardsdatascience.com/single-and-multi-step-temperature-time-series-forecasting-for-vilnius-using-lstm-deep-learning-b9719a0009de
2. https://www.upgrad.com/blog/types-of-activation-function-in-neural-networks/#:~:text=ANNs%20use%20activation%20functions%20(AFs,properties%20in%20the%20neural%20network.
3. https://towardsdatascience.com/learning-process-of-a-deep-neural-network-5a9768d7a651#:~:text=called%20TensorFlow%20Playground.-,Learning%20process%20of%20a%20neural%20network,multiplied%20by%20the%20input%20value.

APPENDIX