Artifical Neural Network and Evolution Final Project Report

Comparison of Time Series Forecasting using Feed Forward Neural Network, Recurrent Neural Network and Hybrid Method

Egawati Panjei – 112866032

Spring 2013

Table of Contents

List of Tables

Comparison of Time Series Forecasting using Feed Forward Neural Network, Recurrent Neural Network and Hybrid Method

Time series forecasting have been applied in many areas like environment, industry, economy and finance. People use several methods like Linear regression (LR), exponential smoothing (ES), autoregressive integrated moving average (ARIMA) to predict linear time series. Previous studies have shown that artificial neural network can model non-linear pattern of time series data. “The neural network structures and training procedures will have a great impact on forecasting performance” (Tang and Fishwick). However in real world, time series data consist of linear and non-linear patterns. Purwanto et al suggest to combine Linear Method and Non Linear Method for more accurate time series forecasting. This project deals with examining the performance of neural network combining with linear method.

Project Objectives

The objective of this project is to compare the performance of time series forecasting using:

- Feed Forward Neural Network (FFNN) with Standard Back Propagation **[1][2]**
- Recurrent Neural Network (RNN) with Back Propagation Through Time **[1][2]**
- FFNN combined with Linear Method
- RNN combined with Linear Method

From the previous report by Purwanto et al, autoregressive integrated moving average (ARIMA) is the best linear method to forecast time series [3]. Therefore in this project I combine FFNN and RNN with ARIMA. For RNN, I choose Elman Recurrent Neural Network, which use output from hidden layer as input for the network [1][2]. Purwanto et all used Multi Layer Perceptron (MLP) Neural Network for their Non-Linear Model. However RNN has advantages of its ability to store previous information compared to MLP. Therefore the experiments conducted in this project evaluate whether the combination of RNN with ARIMA yields the best performance.

Literature Review

This part consist of the short explanation about FFNN, Simple RNN and ARIMA method

Feed Forward Neural Network (FFNN)

FFNN consists of basic units called neuron located in input, hidden and output layer. Every unit in a layer is connected to all units in their neighbor layer by parameter called weights w and bias b. Fig.1 describes the general architecture of MLP, where:

x_i = input of neuron i

y_j = output from hidden layer of neuron j

y_k = output of neuron k

w_{ji} = weight between input and hidden layer

b_j = bias between input and hidden layer

w_{kj} = weigth between hidden and output layer

b_k = bias between hidden and output layer

Input for every hidden layer unit v_j derives from

$$net_j = \sum_{i=0} w_{ji} x_i \tag{1}$$

The output of each neuron is calculated using

$$y_j = \phi_j(net_j) \tag{2}$$

$\phi_j(net_j)$ is an activation function commonly using sigmoid functions

$$\phi_j(net_j) = \frac{1}{1+\exp(-net_j)} \tag{3}$$

For every neuron in output layer,

$$net_k = \sum_{i=0} w_{kj} y_j \tag{4}$$

$$o_k = \phi_k(net_k) \tag{5}$$

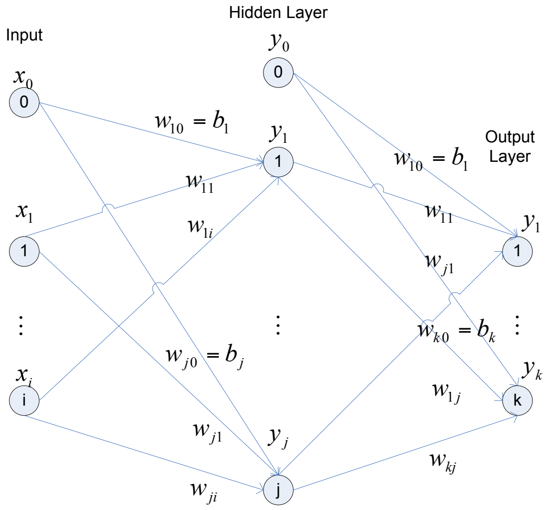


Figure 1. General Network Topology of FFNN

During the training phase the network, go through two stages. The first stage is feed-forward including equations (1) to (5). At the second stage called back-propagation, FFNN learns by adjusting its weight after calculating error between its outputs and targets using gradient descent techniques [1]. The standard back-propagation algorithm consist of:

1. Compute error information terms of output layer $\delta_k = (t_k - o_k)f'(net_k)$ (6)
2. Calculate delta weights between hidden and output layer $\Delta w_{jk} = \alpha \delta_k y_j$ (7)
3. Compute error terms of hidden layer $\delta_j = \left(\sum_k^K \delta_k w_{kj}\right) f'(net_j)$ (8)
4. Calculate delta weights between hidden and output layer $\Delta w_{ij} = \alpha \delta_j x_i$ (9)
5. Update weights $w_{new} = w_{old} + \Delta \mathrm{w}$ (10)

Elman Recurrent Neural Network

The architecture of Elman RNN is almost similar to FFNN. Only in Elman RNN the outputs of hidden layer also become input units. Figure 2 describes the general network topology for Elman RNN.

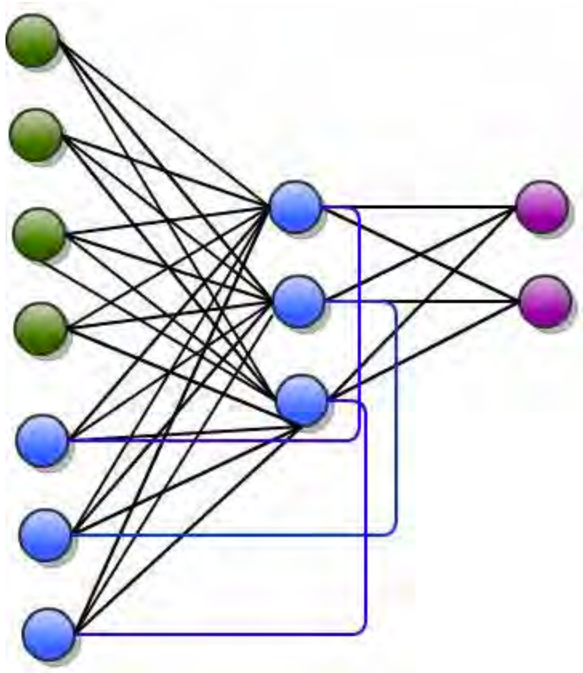


Figure 2 General network topology for Elman RNN

To update the weights associated with input units coming from the output of hidden unit, we can modified equation (9) as follows: $\Delta w_{ij}(t) = \alpha \delta_j net_j(t-1)$. This method is called first order back propagation through time **[2]**.

Autoregressive Integrated Moving Average (ARIMA)

ARIMA model is based on linear equation that consist of 3 parameters p, d, q that respectively refer to the number of autoregressive term, the number of nonseasonal differences, and the number of lagged forecast errors in the prediction equation **[7]**.

"A common obstacle for many people in using Autoregressive Integrated Moving Average (ARIMA) models for forecasting is that the order selection process is usually considered subjective and difficult to apply". Therefore Hyndman and Khandakar suggest function auto.arima() that is already implemented in R. This function can determine the best value of p, d and q **[6].**

Implementations

Tasks to do in order to evaluate the performance of FFNN, FFNN Hybrid, RNN and RNN Hybrid are as follows:

1. Data set normalization.

   In order to use the data set as inputs for FFNN and RNN, we need to normalize the data containing large decimal number. Here I used the min-max data normalization technique by Priddy and Keller [6] and create normalized data in the range of [0.2, 0.8] based on Tang and Fishwick experiment [4].

2. Implementing FFNN for time series forecasting using Java.

   Java classes related to this implementation are FFNN.java and TimeseriesNN.java (Appendix).

3. Implementing RNN for time series forecasting using Java.

   Java classes related to this implementation are RNN.java and TimeseriesRNN.java (Appendix).

4. Implementing FFNN Hybrid and RNN Hybrid by applying ARIMA methods.

   I use function built in R called auto.arima() to get the prediction results of ARIMA method **[5]**. This function is then called from Java using rcaller which is from http://code.google.com/p/rcaller/. Java codes related to this implementation are TimeSeriesHybridRNN.java, TimeSeriesHybridFFNN.java and ARIMA.java (Appendix).

5. Measuring forecasting performance on those models based on:

   a. Root Mean Square Error (RMSE)

   b. Mean Absolute Error (MAE)

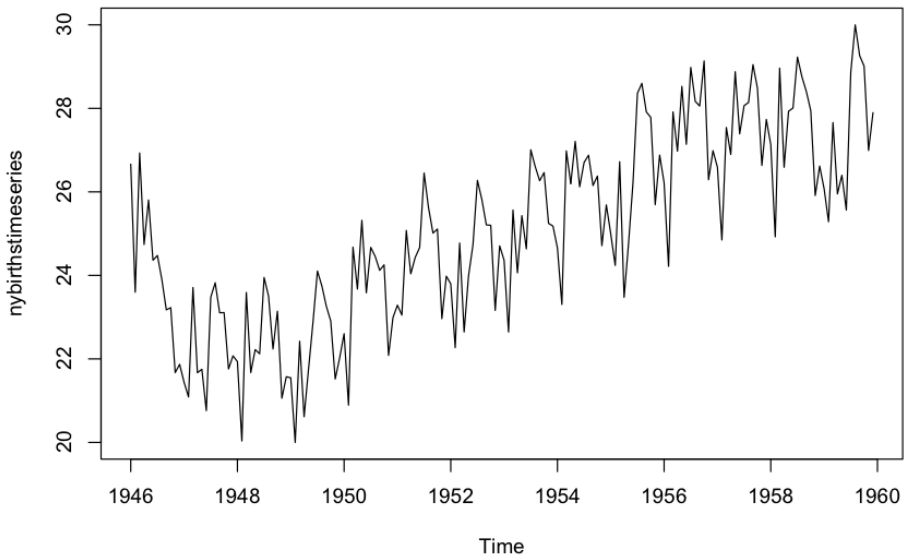
   c. Mean Absolute Percentage Error (MAPE)

   These forecast errors are calculated after de-normalized forecast outputs from each methods. Java code related to the comparison of forecast error is Comparison.java (Appendix).

Experimental Design and Testing

To analyze the performance of FFNN, Hybrid FFNN, RNN and Hybrid RNN, I run 30 trials on 7 different network topologies using the same learning rate (eta), alpha and training length for each data set. Weights are initialized between -0.5 and 0.5. The purpose of these trials is to gather RMSE, MAE, and MAPE and then apply t-test on these forecast errors for each network topology to find the best approach. The forecast error data related to these experiments can be found in the attached Forecast Error folder. The statistical testing is conducted using t.test() function available in R. Time series Data set used in this experiment is taken from http://datamarket.com. I choose 3 data sets, which have different pattern: New York Birth (NYB), Milk Production and Skirt Diameter Data Set.

New York Birth Data Testing



Data Training : 1946 - 1953

Data Testing : 1954 - 1956

Data Validation : 1957 - 1959

Experiment 1 (Network topology: 1-6-1)

Number of input unit	:	1
Number of hidden unit	:	6
Number of output unit	:	1
Learning rate (eta)	:	0.25

Alpha : 0.25

Training Length (epoch) : 500

t – test on RMSE between FFNN and FFNN Hybrid 1-6-1

```
> d = read.table("/Users/Ega/Desktop/NYB1/E1.txt", sep=",");
> #compare FFNN and FFNN Hybrid
> t.test(d$V1, d$V2, alternative = "greater")

	Welch Two Sample t-test

data:  d$V1 and d$V2
t = 190.7631, df = 52.678, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.8664302       Inf
sample estimates:
mean of x mean of y
 1.970276  1.096174
```

Since the p-value is really low we can reject the null hypothesis meaning that RMSE of FFNN is greater than FFNN Hybrid.

t – test on MAE between FFNN and FFNN Hybrid 1-6-1

```
> t.test(d$V5, d$V6, alternative = "greater")

	Welch Two Sample t-test

data:  d$V5 and d$V6
t = 159.5027, df = 49.097, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.6898026       Inf
sample estimates:
mean of x mean of y
1.6356341 0.9385042
```

Since the p-value is really low we can reject the null hypothesis meaning that MAE of FFNN is greater than FFNN Hybrid.

t – test on MAPE between FFNN and FFNN Hybrid 1-6-1

```
> t.test(d$V9, d$V10, alternative = "greater")

	Welch Two Sample t-test

data:  d$V9 and d$V10
t = 150.1756, df = 47.245, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 2.366649      Inf
sample estimates:
mean of x mean of y 
 5.797315  3.403926 
```

Since the p-value is really low we can reject the null hypothesis meaning that MAPE of FFNN is greater than FFNN Hybrid.

t – test on RMSE between RNN and RNN Hybrid 1-6-1

```
> t.test(d$V3, d$V4, alternative = "greater")

	Welch Two Sample t-test

data:  d$V3 and d$V4
t = 4.1639, df = 45.075, p-value = 6.963e-05
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.2644599       Inf
sample estimates:
mean of x mean of y 
 1.669661  1.226446 
```

The p-value is low, therefore we can reject the null hypothesis meaning that RMSE of RNN is greater than RNN Hybrid.

t – test on MAE between RNN and RNN Hybrid 1-6-1

```
> t.test(d$V7, d$V8, alternative = "greater")

	Welch Two Sample t-test

data:  d$V7 and d$V8
t = 4.2904, df = 42.772, p-value = 4.999e-05
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.249821      Inf
sample estimates:
mean of x mean of y 
 1.439933  1.029136 
```

The p-value is low, therefore we can reject the null hypothesis meaning that MAE of RNN is greater than RNN Hybrid.

t – test on MAPE between RNN and RNN Hybrid 1-6-1

```
> t.test(d$V11, d$V12, alternative = "greater")

    Welch Two Sample t-test

data:  d$V11 and d$V12
t = 4.0682, df = 45.77, p-value = 9.249e-05
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.8194611       Inf
sample estimates:
mean of x mean of y 
 5.168773  3.773542 
```

The p-value is low, therefore we can reject the null hypothesis meaning that MAPE of RNN is greater than RNN Hybrid.

t-test on RMSE between FFNN and RNN 1-6-1

```
> t.test(d$V1, d$V3, alternative = "greater")

    Welch Two Sample t-test

data:  d$V1 and d$V3
t = 3.2219, df = 29.048, p-value = 0.001567
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.1420891       Inf
sample estimates:
mean of x mean of y 
 1.970276  1.669661 
```

Here we can conclude that RMSE of FFNN is greater than RNN using topology 1-6-1

t-test on MAE between FFNN and RNN 1-6-1

```
> t.test(d$V5, d$V7, alternative = "greater")

    Welch Two Sample t-test

data:  d$V5 and d$V7
t = 2.2867, df = 29.043, p-value = 0.01484
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.05029384        Inf
sample estimates:
mean of x mean of y 
 1.635634  1.439933 
```

Here we can conclude that MAE of FFNN is greater than RNN using topology 1-6-1

t-test on MAPE between FFNN and RNN 1-6-1

```
> t.test(d$V9, d$V11, alternative = "greater")

	Welch Two Sample t-test

data:  d$V9 and d$V11
t = 2.1036, df = 29.043, p-value = 0.02209
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.120881      Inf
sample estimates:
mean of x mean of y 
 5.797315  5.168773 
```

Here we can conclude that MAPE of FFNN is greater than RNN using topology 1-6-1

t-test on RMSE between FFNN Hybrid and RNN Hybrid 1-6-1

```
> t.test(d$V2, d$V4, alternative = "greater")

	Welch Two Sample t-test

data:  d$V2 and d$V4
t = -2.5329, df = 29.305, p-value = 0.9915
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.2176323        Inf
sample estimates:
mean of x mean of y 
 1.096174  1.226446 
```

Here we can conclude that RMSE of RNN Hybrid is greater than FFNN Hybrid using topology 1-6-1

t-test on MAE between FFNN Hybrid and RNN Hybrid 1-6-1

```
> t.test(d$V6, d$V8, alternative = "greater")

	Welch Two Sample t-test

data:  d$V6 and d$V8
t = -2.1001, df = 29.427, p-value = 0.9778
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.1639226        Inf
sample estimates:
mean of x mean of y 
0.9385042 1.0291360 
```

Here we can conclude that MAE of RNN Hybrid is greater than FFNN Hybrid using topology 1-6-1.

t-test on MAPE between FFNN Hybrid and RNN Hybrid 1-6-1

```
> t.test(d$V10, d$V12, alternative = "greater")

	Welch Two Sample t-test

data:  d$V10 and d$V12
t = -2.1857, df = 29.383, p-value = 0.9815
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.656827       Inf
sample estimates:
mean of x mean of y 
 3.403926  3.773542 
```

Here we can conclude that MAPE of RNN Hybrid is greater than FFNN Hybrid using topology 1-6-1.

Experiment 2 (Network topology: 6-6-1)

Number of input unit	:	6
Number of hidden unit	:	6
Number of output unit	:	1
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 6-6-1

```
> t.test(d2$V1, d2$V2, alternative = "greater")

	Welch Two Sample t-test

data:  d2$V1 and d2$V2
t = 165.1326, df = 37.288, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.741367      Inf
sample estimates:
mean of x mean of y 
 1.796072  1.047054 
```

Since the p-value is really low we can reject the null hypothesis meaning that RMSE of FFNN is greater than FFNN Hybrid.

t – test on MAE between FFNN and FFNN Hybrid 6-6-1

```
> t.test(d2$V5, d2$V6, alternative = "greater")

	Welch Two Sample t-test

data:  d2$V5 and d2$V6
t = 153.9741, df = 46.815, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.710428      Inf
sample estimates:
mean of x mean of y
1.4637463 0.7454905
```

Since the p-value is really low we can reject the null hypothesis meaning that MAE of FFNN is greater than FFNN Hybrid.

t – test on MAPE between FFNN and FFNN Hybrid 6-6-1

```
	Welch Two Sample t-test

data:  d2$V9 and d2$V10
t = 148.4666, df = 49.628, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 2.471982      Inf
sample estimates:
mean of x mean of y
 5.186273  2.686064
```

Since the p-value is really low we can reject the null hypothesis meaning that MAPE of FFNN is greater than FFNN Hybrid.

t – test on RMSE between RNN and RNN Hybrid 6-6-1

```
> t.test(d2$V3, d2$V4, alternative = "greater")

	Welch Two Sample t-test

data:  d2$V3 and d2$V4
t = 2.7769, df = 56.273, p-value = 0.00372
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.1142494       Inf
sample estimates:
mean of x mean of y
 1.523717  1.236476
```

The p-value is low, therefore we can reject the null hypothesis meaning that RMSE of RNN is greater than RNN Hybrid.

t – test on MAE between RNN and RNN Hybrid 6-6-1

```
> t.test(d2$V7, d2$V8, alternative = "greater")

	Welch Two Sample t-test

data:  d2$V7 and d2$V8
t = 2.9137, df = 54.043, p-value = 0.002592
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.1254485       Inf
sample estimates:
mean of x mean of y
1.2566920 0.9619607
```

The p-value is low; therefore we can reject the null hypothesis meaning that MAE of RNN is greater than RNN Hybrid.

t – test on MAPE between RNN and RNN Hybrid 6-6-1

```
> t.test(d2$V11, d2$V12, alternative = "greater")

	Welch Two Sample t-test

data:  d2$V11 and d2$V12
t = 2.6137, df = 51.388, p-value = 0.005862
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.3480237       Inf
sample estimates:
mean of x mean of y
 4.495568  3.526451
```

The p-value is low; therefore we can reject the null hypothesis meaning that MAPE of RNN is greater than RNN Hybrid.

t-test on RMSE between FFNN and RNN 6-6-1

```
> t.test(d2$V1, d2$V3, alternative = "greater")

	Welch Two Sample t-test

data:  d2$V1 and d2$V3
t = 4.0916, df = 29.236, p-value = 0.000154
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.1592847       Inf
sample estimates:
mean of x mean of y
 1.796072  1.523717
```

Here we can conclude that RMSE of FFNN is greater than RNN using topology 6-6-1 *t-test on MAE between FFNN and RNN 6-6-1*

```
> t.test(d2$V5, d2$V7, alternative = "greater")

	Welch Two Sample t-test

data:  d2$V5 and d2$V7
t = 3.3822, df = 29.252, p-value = 0.00103
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.1030657       Inf
sample estimates:
mean of x mean of y 
 1.463746  1.256692 
```

Here we can conclude that MAE of FFNN is greater than RNN using topology 6-6-1

t-test on MAPE between FFNN and RNN 6-6-1

```
> t.test(d2$V9, d2$V11, alternative = "greater")

	Welch Two Sample t-test

data:  d2$V9 and d2$V11
t = 3.2822, df = 29.263, p-value = 0.001334
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.3332522       Inf
sample estimates:
mean of x mean of y 
 5.186273  4.495568 
```

Here we can conclude that MAPE of FFNN is greater than RNN using topology 6-6-1

t-test on RMSE between FFNN Hybrid and RNN Hybrid 6-6-1

```
> t.test(d2$V2, d2$V4, alternative = "greater")

	Welch Two Sample t-test

data:  d2$V2 and d2$V4
t = -2.3884, df = 29.024, p-value = 0.9882
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.324172       Inf
sample estimates:
mean of x mean of y 
 1.047054  1.236476 
```

Here we can conclude that RMSE of RNN Hybrid is greater than FFNN Hybrid on topology 6-6-1.

t-test on MAE between FFNN Hybrid and RNN Hybrid 6-6-1

```
> t.test(d2$V6, d2$V8, alternative = "greater")

	Welch Two Sample t-test

data:  d2$V6 and d2$V8
t = -2.6838, df = 29.05, p-value = 0.9941
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.3535122        Inf
sample estimates:
mean of x mean of y 
0.7454905 0.9619607 
```

Here we can conclude that MAE of RNN Hybrid is greater than FFNN Hybrid on topology 6-6-1.

t-test on MAPE between FFNN Hybrid and RNN Hybrid 6-6-1

```
> t.test(d2$V10, d2$V12, alternative = "greater")

	Welch Two Sample t-test

data:  d2$V10 and d2$V12
t = -2.7486, df = 29.052, p-value = 0.9949
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -1.359872       Inf
sample estimates:
mean of x mean of y 
 2.686064  3.526451 
```

Here we can conclude that MAPE of RNN Hybrid is greater than FFNN Hybrid using topology 6-6-1.

Experiment 3 (Network topology: 6-6-4)

Number of input unit	:	6
Number of hidden unit	:	6
Number of output unit	:	4
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 6-6-4

```
> t.test(d3$V1, d3$V2, alternative = "greater")

    Welch Two Sample t-test

data:  d3$V1 and d3$V2
t = 156.4493, df = 29.762, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 1.15503     Inf
sample estimates:
mean of x mean of y
 2.394459  1.226758
```

Since the p-value is really low we can reject the null hypothesis meaning that RMSE of FFNN is greater than FFNN Hybrid.

t – test on MAE between FFNN and FFNN Hybrid 6-6-4

```
> t.test(d3$V5, d3$V6, alternative = "greater")

    Welch Two Sample t-test

data:  d3$V5 and d3$V6
t = 115.6518, df = 29.846, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.9999293       Inf
sample estimates:
mean of x mean of y
 2.026619  1.011794
```

Since the p-value is really low we can reject the null hypothesis meaning that MAE of FFNN is greater than FFNN Hybrid.

t – test on MAPE between FFNN and FFNN Hybrid 6-6-4

```
> t.test(d3$V9, d3$V10, alternative = "greater")

    Welch Two Sample t-test

data:  d3$V9 and d3$V10
t = 111.2358, df = 30.07, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 3.453701      Inf
sample estimates:
mean of x mean of y
 7.186347  3.679136
```

Since the p-value is really low we can reject the null hypothesis meaning that MAPE of FFNN is greater than FFNN Hybrid.

t – test on RMSE between RNN and RNN Hybrid 6-6-4

```
> t.test(d3$V3, d3$V4, alternative = "greater")

	Welch Two Sample t-test

data:  d3$V3 and d3$V4
t = 6.1922, df = 52.974, p-value = 4.477e-08
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.4241271       Inf
sample estimates:
mean of x mean of y 
 2.196317  1.615035 
```

The p-value is low, therefore we can reject the null hypothesis meaning that RMSE of RNN is greater than RNN Hybrid.

t – test on MAE between RNN and RNN Hybrid 6-6-4

```
> t.test(d3$V7, d3$V8, alternative = "greater")

	Welch Two Sample t-test

data:  d3$V7 and d3$V8
t = 7.1593, df = 57.623, p-value = 8.177e-10
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.408182      Inf
sample estimates:
mean of x mean of y 
 1.812157  1.279626 
```

The p-value is low, therefore we can reject the null hypothesis meaning that MAE of RNN is greater than RNN Hybrid.

t – test on MAPE between RNN and RNN Hybrid 6-6-4

```
> t.test(d3$V11, d3$V12, alternative = "greater")

	Welch Two Sample t-test

data:  d3$V11 and d3$V12
t = 6.5521, df = 57.914, p-value = 8.299e-09
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 1.3122     Inf
sample estimates:
mean of x mean of y 
 6.450100  4.688466 
```

The p-value is low, therefore we can reject the null hypothesis meaning that MAPE of RNN is greater than RNN Hybrid.

t-test on RMSE between FFNN and RNN 6-6-4

```
> t.test(d3$V1, d3$V3, alternative = "greater")

    Welch Two Sample t-test

data:  d3$V1 and d3$V3
t = 3.5565, df = 30.046, p-value = 0.0006344
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.1035879       Inf
sample estimates:
mean of x mean of y
 2.394459  2.196317
```

Here we can conclude that RMSE of FFNN is greater than RNN using topology 6-6-4

t-test on MAE between FFNN and RNN 6-6-4

```
> t.test(d3$V5, d3$V7, alternative = "greater")

    Welch Two Sample t-test

data:  d3$V5 and d3$V7
t = 3.8731, df = 30.471, p-value = 0.0002651
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.1205261       Inf
sample estimates:
mean of x mean of y
 2.026619  1.812157
```

Here we can conclude that MAE of FFNN is greater than RNN using topology 6-6-4

t-test on MAPE between FFNN and RNN 6-6-4

```
> t.test(d3$V9, d3$V11, alternative = "greater")

    Welch Two Sample t-test

data:  d3$V9 and d3$V11
t = 3.7516, df = 30.507, p-value = 0.000369
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.4033367       Inf
sample estimates:
mean of x mean of y
 7.186347  6.450100
```

Here we can conclude that MAPE of FFNN is greater than RNN using topology 6-6-4 *t-test on RMSE between FFNN Hybrid and RNN Hybrid 6-6-4*

```
> t.test(d3$V2, d3$V4, alternative = "greater")

    Welch Two Sample t-test

data:  d3$V2 and d3$V4
t = -5.1143, df = 29.007, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.517274       Inf
sample estimates:
mean of x mean of y 
 1.226758  1.615035 
```

Here we can conclude that RMSE of RNN Hybrid is greater than FFNN Hybrid using topology 6-6-4

t-test on MAE between FFNN Hybrid and RNN Hybrid 6-6-4

```
> t.test(d3$V6, d3$V8, alternative = "greater")

    Welch Two Sample t-test

data:  d3$V6 and d3$V8
t = -5.3103, df = 29.025, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.3535262        Inf
sample estimates:
mean of x mean of y 
 1.011794  1.279626 
```

Here we can conclude that MAE of RNN Hybrid is greater than FFNN Hybrid using topology 6-6-4.

t-test on MAPE between FFNN Hybrid and RNN Hybrid 6-6-4

```
> t.test(d3$V10, d3$V12, alternative = "greater")

    Welch Two Sample t-test

data:  d3$V10 and d3$V12
t = -5.413, df = 29.03, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -1.326146       Inf
sample estimates:
mean of x mean of y 
 3.679136  4.688466 
```

Here we can conclude that MAPE of RNN Hybrid is greater than FFNN Hybrid using topology 6-6-4.

Experiment 4 (Network topology: 6-6-6)

Number of input unit	:	6
Number of hidden unit	:	6
Number of output unit	:	6
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 6-6-6

```
> t.test(d4$V1, d4$V2, alternative = "greater")

    Welch Two Sample t-test

data:  d4$V1 and d4$V2
t = 135.9076, df = 29.911, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 1.343322      Inf
sample estimates:
mean of x mean of y
 2.596172  1.235860
```

Since the p-value is really low we can reject the null hypothesis meaning that RMSE of FFNN is greater than FFNN Hybrid.

t – test on MAE between FFNN and FFNN Hybrid 6-6-6

```
> t.test(d4$V5, d4$V6, alternative = "greater")

    Welch Two Sample t-test

data:  d4$V5 and d4$V6
t = 107.9837, df = 29.502, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 1.182974      Inf
sample estimates:
mean of x mean of y
 2.247475  1.045600
```

Since the p-value is really low we can reject the null hypothesis meaning that MAE of FFNN is greater than FFNN Hybrid.

t – test on MAPE between FFNN and FFNN Hybrid 6-6-6

```
> t.test(d4$V9, d4$V10, alternative = "greater")

	Welch Two Sample t-test

data:  d4$V9 and d4$V10
t = 103.1395, df = 29.52, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 4.09683     Inf
sample estimates:
mean of x mean of y
 7.991801  3.826390
```

Since the p-value is really low we can reject the null hypothesis meaning that MAPE of FFNN is greater than FFNN Hybrid.

t – test on RMSE between RNN and RNN Hybrid 6-6-6

```
> t.test(d4$V3, d4$V4, alternative = "greater")

	Welch Two Sample t-test

data:  d4$V3 and d4$V4
t = 25.2845, df = 57.848, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 1.275237      Inf
sample estimates:
mean of x mean of y
 2.711535  1.346021
```

The p-value is low, therefore we can reject the null hypothesis meaning that RMSE of RNN is greater than RNN Hybrid.

t – test on MAE between RNN and RNN Hybrid 6-6-6

```
> t.test(d4$V7, d4$V8, alternative = "greater")

	Welch Two Sample t-test

data:  d4$V7 and d4$V8
t = 24.2924, df = 47.354, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 1.130059      Inf
sample estimates:
mean of x mean of y
 2.322481  1.108589
```

The p-value is low, therefore we can reject the null hypothesis meaning that MAE of RNN is greater than RNN Hybrid.

t – test on MAPE between RNN and RNN Hybrid 6-6-6

```
> t.test(d4$V11, d4$V12, alternative = "greater")

	Welch Two Sample t-test

data:  d4$V11 and d4$V12
t = 23.2302, df = 48.653, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 3.907534      Inf
sample estimates:
mean of x mean of y 
 8.277659  4.066132 
```

The p-value is low, therefore we can reject the null hypothesis meaning that MAPE of RNN is greater than RNN Hybrid.

t-test on RMSE between FFNN and RNN 6-6-6

```
> t.test(d4$V1, d4$V3, alternative = "greater")

	Welch Two Sample t-test

data:  d4$V1 and d4$V3
t = -2.8559, df = 32.716, p-value = 0.9963
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.1837429        Inf
sample estimates:
mean of x mean of y 
 2.596172  2.711535 
```

RMSE of RNN is greater than FFNN.

t-test on MAE between FFNN and RNN 6-6-6

```
> t.test(d4$V5, d4$V7, alternative = "greater")

	Welch Two Sample t-test

data:  d4$V5 and d4$V7
t = -1.6928, df = 32.853, p-value = 0.95
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.1500028        Inf
sample estimates:
mean of x mean of y 
 2.247475  2.322481 
```

MAE of RNN is greater than FFNN.

t-test on MAPE between FFNN and RNN 6-6-6

```
> t.test(d4$V9, d4$V11, alternative = "greater")

	Welch Two Sample t-test

data:  d4$V9 and d4$V11
t = -1.7988, df = 32.948, p-value = 0.9594
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.55481      Inf
sample estimates:
mean of x mean of y 
 7.991801  8.277659 
```

MAPE of RNN is greater than FFNN.

t-test on RMSE between FFNN Hybrid and RNN Hybrid 6-6-6

```
> t.test(d4$V2, d4$V4, alternative = "greater")

	Welch Two Sample t-test

data:  d4$V2 and d4$V4
t = -2.96, df = 29.065, p-value = 0.997
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.1733915        Inf
sample estimates:
mean of x mean of y 
 1.235860  1.346021 
```

RMSE of RNN Hybrid is greater than FFNN Hybrid .

t-test on MAE between FFNN Hybrid and RNN Hybrid 6-6-6

```
> t.test(d4$V6, d4$V8, alternative = "greater")

	Welch Two Sample t-test

data:  d4$V6 and d4$V8
t = -2.4563, df = 29.094, p-value = 0.9899
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.106556       Inf
sample estimates:
mean of x mean of y 
 1.045600  1.108589 
```

Here we can conclude that MAE of RNN Hybrid is greater than FFNN Hybrid using topology 6-6-6.

t-test on MAPE between FFNN Hybrid and RNN Hybrid 6-6-6

```
> t.test(d4$V10, d4$V12, alternative = "greater")

    Welch Two Sample t-test

data:  d4$V10 and d4$V12
t = -2.4933, df = 29.091, p-value = 0.9907
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.403101       Inf
sample estimates:
mean of x mean of y 
 3.826390  4.066132 
```

Here we can conclude that MAPE of RNN Hybrid is greater than FFNN Hybrid using topology 6-6-6.

Experiment 5 (Network topology: 12-6-1)

Number of input unit	:	12
Number of hidden unit	:	6
Number of output unit	:	1
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 12-6-1

```
> t.test(d5$V1, d5$V2, alternative = "greater")

    Welch Two Sample t-test

data:  d5$V1 and d5$V2
t = 51.3173, df = 31.567, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.3454042       Inf
sample estimates:
mean of x mean of y 
1.1849537 0.8277541 
```

Since the p-value is really low we can reject the null hypothesis meaning that RMSE of FFNN is greater than FFNN Hybrid.

t – test on MAE between FFNN and FFNN Hybrid 12-6-1

```
> t.test(d5$V5, d5$V6, alternative = "greater")

	Welch Two Sample t-test

data:  d5$V5 and d5$V6
t = 49.0362, df = 52.23, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.3340649       Inf
sample estimates:
mean of x mean of y 
0.9109726 0.5650962 
```

Since the p-value is really low we can reject the null hypothesis meaning that MAE of FFNN is greater than FFNN Hybrid.

t – test on MAPE between FFNN and FFNN Hybrid 12-6-1

```
> t.test(d5$V9, d5$V10, alternative = "greater")

	Welch Two Sample t-test

data:  d5$V9 and d5$V10
t = 50.5829, df = 54.075, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 1.199267      Inf
sample estimates:
mean of x mean of y 
 3.240436  2.000133 
```

Since the p-value is really low we can reject the null hypothesis meaning that MAPE of FFNN is greater than FFNN Hybrid.

t – test on RMSE between RNN and RNN Hybrid 12-6-1

```
> t.test(d5$V3, d5$V4, alternative = "greater")

	Welch Two Sample t-test

data:  d5$V3 and d5$V4
t = -5.9537, df = 32.687, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -1.36974      Inf
sample estimates:
mean of x mean of y 
 1.385712  2.452210 
```

Here we can see clearly that the RMSE of RNN Hybrid is greater than RNN.

t – test on MAE between RNN and RNN Hybrid 12-6-1

```
> t.test(d5$V7, d5$V8, alternative = "greater")

	Welch Two Sample t-test

data:  d5$V7 and d5$V8
t = -5.9498, df = 31.566, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -1.385477       Inf
sample estimates:
mean of x mean of y 
 1.152150  2.230497
```

Obviously the MAE of RNN Hybrid is greater than RNN.

t – test on MAPE between RNN and RNN Hybrid 12-6-1

```
> t.test(d5$V11, d5$V12, alternative = "greater")

	Welch Two Sample t-test

data:  d5$V11 and d5$V12
t = -5.9473, df = 31.867, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -5.142108       Inf
sample estimates:
mean of x mean of y 
 4.293635  8.295736
```

MAPE of RNN Hybrid is greater than RNN.

t-test on RMSE between FFNN and RNN 12-6-1

```
> t.test(d5$V1, d5$V3, alternative = "greater")

	Welch Two Sample t-test

data:  d5$V1 and d5$V3
t = -4.5211, df = 30.397, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.2760922        Inf
sample estimates:
mean of x mean of y 
 1.184954  1.385712
```

RMSE of RNN is greater than FFNN.

t-test on MAE between FFNN and RNN 12-6-1

```
> t.test(d5$V5, d5$V7, alternative = "greater")

    Welch Two Sample t-test

data:  d5$V5 and d5$V7
t = -6.3834, df = 30.378, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.3052772        Inf
sample estimates:
mean of x mean of y
0.9109726 1.1521496
```

MAE of RNN is greater than FFNN using topology 12-6-1.

t-test on MAPE between FFNN and RNN 12-6-1

```
> t.test(d5$V9, d5$V11, alternative = "greater")

    Welch Two Sample t-test

data:  d5$V9 and d5$V11
t = -7.14, df = 30.035, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -1.303548       Inf
sample estimates:
mean of x mean of y
 3.240436  4.293635
```

MAPE of RNN is greater than FFNN using topology 12-6-1.

t-test on RMSE between FFNN Hybrid and RNN Hybrid 12-6-1

```
> t.test(d5$V2, d5$V4, alternative = "greater")

    Welch Two Sample t-test

data:  d5$V2 and d5$V4
t = -9.3531, df = 29.004, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -1.919562       Inf
sample estimates:
mean of x mean of y
0.8277541 2.4522103
```

RMSE of RNN Hybrid is greater than FFNN Hybrid.

t-test on MAE between FFNN Hybrid and RNN Hybrid 12-6-1

```
> t.test(d5$V6, d5$V8, alternative = "greater")

	Welch Two Sample t-test

data:  d5$V6 and d5$V8
t = -9.3878, df = 29.031, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -1.966817       Inf
sample estimates:
mean of x mean of y 
0.5650962 2.2304973 
```

MAE of RNN Hybrid is greater than FFNN Hybrid.

t-test on MAPE between FFNN Hybrid and RNN Hybrid 12-6-1

```
> t.test(d5$V10, d5$V12, alternative = "greater")

	Welch Two Sample t-test

data:  d5$V10 and d5$V12
t = -9.5821, df = 29.03, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -7.411925       Inf
sample estimates:
mean of x mean of y 
 2.000133  8.295736 
```

MAPE of RNN Hybrid is greater than FFNN Hybrid.

Experiment 6 (Network topology: 12-12-4)

Number of input unit	:	12
Number of hidden unit	:	12
Number of output unit	:	4
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 12-6-4

```
> t.test(d6$V1, d6$V2, alternative = "greater")

	Welch Two Sample t-test

data:  d6$V1 and d6$V2
t = 21.2394, df = 52.271, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.3408243       Inf
sample estimates:
mean of x mean of y 
1.3346251 0.9646301 
```

Since the p-value is really low we can reject the null hypothesis meaning that RMSE of FFNN is greater than FFNN Hybrid.

t – test on MAE between FFNN and FFNN Hybrid 12-12-4

```
> t.test(d6$V5, d6$V6, alternative = "greater")

	Welch Two Sample t-test

data:  d6$V5 and d6$V6
t = 22.7157, df = 52.595, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.2889488       Inf
sample estimates:
mean of x mean of y 
 1.044747  0.732805 
```

Since the p-value is really low we can reject the null hypothesis meaning that MAE of FFNN is greater than FFNN Hybrid.

t – test on MAPE between FFNN and FFNN Hybrid 12-12-4

```
> t.test(d6$V9, d6$V10, alternative = "greater")

	Welch Two Sample t-test

data:  d6$V9 and d6$V10
t = 21.1542, df = 48.71, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.9794932       Inf
sample estimates:
mean of x mean of y 
 3.713653  2.649839 
```

Since the p-value is really low we can reject the null hypothesis meaning that MAPE of FFNN is greater than FFNN Hybrid.

t – test on RMSE between RNN and RNN Hybrid 12-12-4

```
> t.test(d6$V3, d6$V4, alternative = "greater")

    Welch Two Sample t-test

data:  d6$V3 and d6$V4
t = -8.9018, df = 34.467, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -2.211949       Inf
sample estimates:
mean of x mean of y
 1.780404  3.639371
```

Here we can see clearly that the RMSE of RNN Hybrid is greater than RNN.

t – test on MAE between RNN and RNN Hybrid 12-12-4

```
> t.test(d6$V7, d6$V8, alternative = "greater")

    Welch Two Sample t-test

data:  d6$V7 and d6$V8
t = -6.961, df = 32.917, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -1.84501       Inf
sample estimates:
mean of x mean of y
 1.465718  2.949874
```

Obviously the MAE of RNN Hybrid is greater than RNN.

t – test on MAPE between RNN and RNN Hybrid 12-12-4

```
> t.test(d6$V11, d6$V12, alternative = "greater")

    Welch Two Sample t-test

data:  d6$V11 and d6$V12
t = -6.9562, df = 33.106, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -6.837249       Inf
sample estimates:
mean of x mean of y
 5.425307 10.924738
```

MAPE of RNN Hybrid is greater than RNN.

t-test on RMSE between FFNN and RNN 12-12-4

```
> t.test(d6$V1, d6$V3, alternative = "greater")

	Welch Two Sample t-test

data:  d6$V1 and d6$V3
t = -7.1481, df = 30.553, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.5515645        Inf
sample estimates:
mean of x mean of y 
 1.334625  1.780404 
```

RMSE of RNN is greater than FFNN.

t-test on MAE between FFNN and RNN 12-12-4

```
> t.test(d6$V5, d6$V7, alternative = "greater")

	Welch Two Sample t-test

data:  d6$V5 and d6$V7
t = -7.7474, df = 30.286, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -0.5131685        Inf
sample estimates:
mean of x mean of y 
 1.044747  1.465718 
```

MAE of RNN is greater than FFNN.

t-test on MAPE between FFNN and RNN 12-12-4

```
> t.test(d6$V9, d6$V11, alternative = "greater")

	Welch Two Sample t-test

data:  d6$V9 and d6$V11
t = -8.329, df = 29.995, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -2.060451       Inf
sample estimates:
mean of x mean of y 
 3.713653  5.425307 
```

MAPE of RNN is greater than FFNN.

t-test on RMSE between FFNN Hybrid and RNN Hybrid 12-12-4

```
> t.test(d6$V2, d6$V4, alternative = "greater")

	Welch Two Sample t-test

data:  d6$V2 and d6$V4
t = -13.3696, df = 29.294, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -3.014557       Inf
sample estimates:
mean of x mean of y
0.9646301 3.6393714
```

RMSE of RNN Hybrid is greater than FFNN Hybrid.

t-test on MAE between FFNN Hybrid and RNN Hybrid 12-12-4

```
> t.test(d6$V6, d6$V8, alternative = "greater")

	Welch Two Sample t-test

data:  d6$V6 and d6$V8
t = -10.7298, df = 29.17, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -2.568086       Inf
sample estimates:
mean of x mean of y
 0.732805  2.949874
```

MAE of RNN Hybrid is greater than FFNN Hybrid.

t-test on MAPE between FFNN Hybrid and RNN Hybrid 12-12-4

```
> t.test(d6$V10, d6$V12, alternative = "greater")

	Welch Two Sample t-test

data:  d6$V10 and d6$V12
t = -10.8161, df = 29.181, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -9.57456      Inf
sample estimates:
mean of x mean of y
 2.649839 10.924738
```

MAPE of RNN Hybrid is greater than FFNN Hybrid.

Experiment 7 (Network topology: 12-18-12)

Number of input unit : 12

Number of hidden unit : 18

Number of output unit	:	12
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 12-18-12

```
> t.test(d7$V1, d7$V2, alternative = "greater")

    Welch Two Sample t-test

data:  d7$V1 and d7$V2
t = 20.5816, df = 52.005, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.7248158       Inf
sample estimates:
mean of x mean of y
 2.152039  1.363022
```

Since the p-value is really low we can reject the null hypothesis meaning that RMSE of FFNN is greater than FFNN Hybrid.

t – test on MAE between FFNN and FFNN Hybrid 12-18-12

```
> t.test(d7$V5, d7$V6, alternative = "greater")

    Welch Two Sample t-test

data:  d7$V5 and d7$V6
t = 27.7006, df = 55.806, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.727785      Inf
sample estimates:
mean of x mean of y
 1.797346  1.022792
```

Since the p-value is really low we can reject the null hypothesis meaning that MAE of FFNN is greater than FFNN Hybrid.

t – test on MAPE between FFNN and FFNN Hybrid 12-18-12

```
> t.test(d7$V9, d7$V10, alternative = "greater")

	Welch Two Sample t-test

data:  d7$V9 and d7$V10
t = 26.232, df = 55.351, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 2.567924      Inf
sample estimates:
mean of x mean of y
 6.535117  3.792278
```

Since the p-value is really low we can reject the null hypothesis meaning that MAPE of FFNN is greater than FFNN Hybrid.

t – test on RMSE between RNN and RNN Hybrid 12-18-12

```
> t.test(d7$V3, d7$V4, alternative = "greater")

	Welch Two Sample t-test

data:  d7$V3 and d7$V4
t = 7.9644, df = 57.167, p-value = 3.863e-11
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.7196576       Inf
sample estimates:
mean of x mean of y
 4.149893  3.239017
```

RMSE of RNN is greater than RNN Hybrid.

t – test on MAE between RNN and RNN Hybrid 12-18-12

```
> t.test(d7$V7, d7$V8, alternative = "greater")

	Welch Two Sample t-test

data:  d7$V7 and d7$V8
t = 14.4542, df = 57.698, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 1.13715     Inf
sample estimates:
mean of x mean of y
 3.457599  2.171733
```

MAE of RNN is greater than RNN Hybrid.

t – test on MAPE between RNN and RNN Hybrid 12-18-12

```
> t.test(d7$V11, d7$V12, alternative = "greater")

    Welch Two Sample t-test

data:  d7$V11 and d7$V12
t = 13.875, df = 57.874, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 3.996516      Inf
sample estimates:
mean of x mean of y
 12.64849   8.10453
```

MAPE of RNN is greater than RNN Hybrid.

t-test on RMSE between FFNN and RNN 12-12-18

```
> t.test(d7$V1, d7$V3, alternative = "greater")

    Welch Two Sample t-test

data:  d7$V1 and d7$V3
t = -25.2997, df = 33.86, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -2.131398       Inf
sample estimates:
mean of x mean of y
 2.152039  4.149893
```

RMSE of RNN is greater than FFNN.

t-test on MAE between FFNN and RNN 12-12-18

```
> t.test(d7$V5, d7$V7, alternative = "greater")

    Welch Two Sample t-test

data:  d7$V5 and d7$V7
t = -24.595, df = 33.261, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -1.774468       Inf
sample estimates:
mean of x mean of y
 1.797346  3.457599
```

MAE of RNN is greater than FFNN.

t-test on MAPE between FFNN and RNN 12-12-18

```
> t.test(d7$V9, d7$V11, alternative = "greater")

	Welch Two Sample t-test

data:  d7$V9 and d7$V11
t = -24.8746, df = 33.387, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -6.529158       Inf
sample estimates:
mean of x mean of y
 6.535117 12.648487
```

MAPE of RNN is greater than FFNN.

t-test on RMSE between FFNN Hybrid and RNN Hybrid 12-12-18

```
> t.test(d7$V2, d7$V4, alternative = "greater")

	Welch Two Sample t-test

data:  d7$V2 and d7$V4
t = -20.5743, df = 36.651, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -2.029864       Inf
sample estimates:
mean of x mean of y
 1.363022  3.239017
```

RMSE of RNN Hybrid is greater than FFNN Hybrid.

t-test on MAE between FFNN Hybrid and RNN Hybrid 12-12-18

```
> t.test(d7$V6, d7$V8, alternative = "greater")

	Welch Two Sample t-test

data:  d7$V6 and d7$V8
t = -17.8581, df = 36.283, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -1.257539       Inf
sample estimates:
mean of x mean of y
 1.022792  2.171733
```

MAE of RNN Hybrid is greater than FFNN Hybrid.

t-test on MAPE between FFNN Hybrid and RNN Hybrid 12-12-18

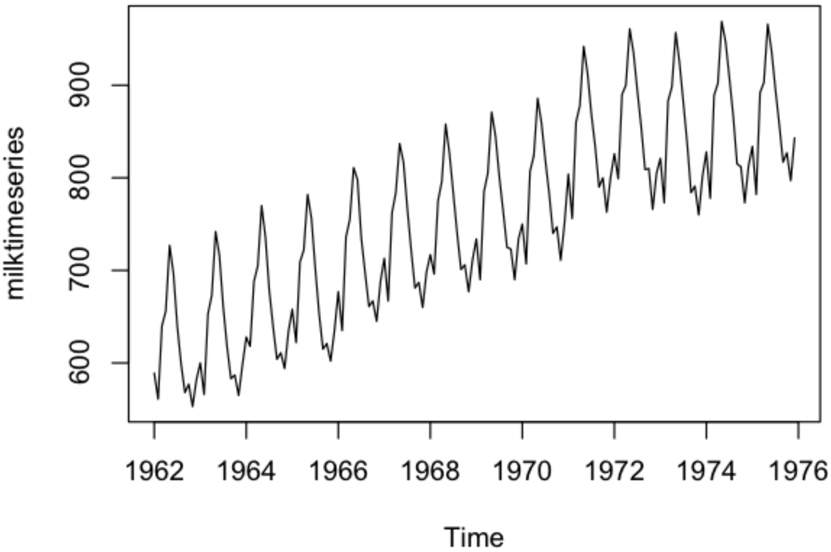
```
> t.test(d7$V10, d7$V12, alternative = "greater")

	Welch Two Sample t-test

data:  d7$V10 and d7$V12
t = -17.9392, df = 36.433, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -4.717959       Inf
sample estimates:
mean of x mean of y
 3.792278  8.104530
```

MAPE of RNN Hybrid is greater than FFNN Hybrid.

Milk Production Data Testing



Data Training : 1962 - 1969

Data Testing : 1970 - 1972

Data Validation : 1973 - 1975

Experiment 1 (Network topology: 1-6-1)

Number of input unit	:	1
Number of hidden unit	:	6
Number of output unit	:	1
Learning rate (eta)	:	0.25
Alpha	:	0.25

Training Length (epoch)   :   500

t – test on RMSE between FFNN and FFNN Hybrid 1-6-1

```
> t.test(m$V1, m$V2, alternative = "greater");

	Welch Two Sample t-test

data:  m$V1 and m$V2
t = 90.156, df = 32.498, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 21.0648      Inf
sample estimates:
mean of x mean of y 
 58.94170  37.47374 
```

FFNN Hybrid is better than FFNN.

t – test on RMSE between RNN and RNN Hybrid 1-6-1

```
> t.test(m$V3, m$V4, alternative = "greater");

	Welch Two Sample t-test

data:  m$V3 and m$V4
t = 2.1344, df = 56.695, p-value = 0.01857
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 1.950401      Inf
sample estimates:
mean of x mean of y 
 61.24295  52.23711 
```

RNN Hybrid is better than RNN.

t – test on RMSE between FFNN and RNN 1-6-1

```
> t.test(m$V1, m$V3, alternative = "greater");

	Welch Two Sample t-test

data:  m$V1 and m$V3
t = -0.8345, df = 29.411, p-value = 0.7946
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -6.984591       Inf
sample estimates:
mean of x mean of y 
 58.94170  61.24295 
```

FFNN is better than RNN.

t – test on RMSE between FFNN Hybrid and RNN Hybrid 1-6-1

```
> t.test(m$V2, m$V4, alternative = "greater");

	Welch Two Sample t-test

data:  m$V2 and m$V4
t = -4.6102, df = 29.018, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -20.20445       Inf
sample estimates:
mean of x mean of y 
 37.47374  52.23711
```

FFNN Hybrid is better than RNN Hybrid

Experiment 2 (Network topology: 6-6-1)

Number of input unit	:	6
Number of hidden unit	:	6
Number of output unit	:	1
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 6-6-1

```
> t.test(m2$V1, m2$V2, alternative = "greater");

	Welch Two Sample t-test

data:  m2$V1 and m2$V2
t = 82.2397, df = 35.299, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 25.07697      Inf
sample estimates:
mean of x mean of y 
 53.11377  27.51092
```

FFNN Hybrid is better than FFNN.

t – test on RMSE between RNN and RNN Hybrid 6-6-1

```
> t.test(m2$V3, m2$V4, alternative = "greater");

	Welch Two Sample t-test

data:  m2$V3 and m2$V4
t = 6.7995, df = 29.921, p-value = 7.754e-08
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 32.92915      Inf
sample estimates:
mean of x mean of y 
 77.98251  34.09821 
```

RNN Hybrid is better than RNN.

t – test on RMSE between FFNN and RNN 6-6-1

```
> t.test(m2$V1, m2$V3, alternative = "greater");

	Welch Two Sample t-test

data:  m2$V1 and m2$V3
t = -3.8795, df = 29.124, p-value = 0.9997
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -35.75901       Inf
sample estimates:
mean of x mean of y 
 53.11377  77.98251 
```

FFNN is better than RNN.

t – test on RMSE between FFNN Hybrid and RNN Hybrid 6-6-1

```
> t.test(m2$V2, m2$V4, alternative = "greater");

	Welch Two Sample t-test

data:  m2$V2 and m2$V4
t = -8.1036, df = 29.855, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -7.967177       Inf
sample estimates:
mean of x mean of y 
 27.51092  34.09821 
```

FFNN Hybrid is better than RNN Hybrid.

Experiment 3 (Network topology: 6-6-4)

Number of input unit : 6

Number of hidden unit	:	6
Number of output unit	:	4
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 6-6-4

```
> t.test(m3$V1, m3$V2, alternative = "greater");

    Welch Two Sample t-test

data:  m3$V1 and m3$V2
t = 27.6689, df = 33.109, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 50.43305      Inf
sample estimates:
mean of x mean of y
 83.72554  30.00712
```

FFNN Hybrid is better than FFNN.

t – test on RMSE between RNN and RNN Hybrid 6-6-4

```
> t.test(m3$V3, m3$V4, alternative = "greater");

    Welch Two Sample t-test

data:  m3$V3 and m3$V4
t = 13.8627, df = 32.76, p-value = 1.468e-15
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 90.13265      Inf
sample estimates:
mean of x mean of y
145.97203  43.30282
```

RNN Hybrid is better than RNN.

t – test on RMSE between FFNN and RNN 6-6-4

```
> t.test(m3$V1, m3$V3, alternative = "greater");

	Welch Two Sample t-test

data:  m3$V1 and m3$V3
t = -8.392, df = 32.945, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -74.8   Inf
sample estimates:
mean of x mean of y
 83.72554 145.97203
```

FFNN is better than RNN.

t – test on RMSE between FFNN Hybrid and RNN Hybrid 6-6-4

```
> t.test(m3$V2, m3$V4, alternative = "greater");

	Welch Two Sample t-test

data:  m3$V2 and m3$V4
t = -7.0044, df = 33.311, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -16.50725       Inf
sample estimates:
mean of x mean of y
 30.00712  43.30282
```

FFNN Hybrid is better than RNN Hybrid

Experiment 4 (Network topology: 6-6-6)

Number of input unit	:	6
Number of hidden unit	:	6
Number of output unit	:	6
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 6-6-6

```
> t.test(m4$V1, m4$V2, alternative = "greater");

	Welch Two Sample t-test

data:  m4$V1 and m4$V2
t = 30.9988, df = 33.875, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 57.15042      Inf
sample estimates:
mean of x mean of y
 92.85432  32.40624
```

FFNN Hybrid is better than FFNN.

t – test on RMSE between RNN and RNN Hybrid 6-6-6

```
> t.test(m4$V3, m4$V4, alternative = "greater");

	Welch Two Sample t-test

data:  m4$V3 and m4$V4
t = 2.29, df = 57.907, p-value = 0.01284
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 6.361956      Inf
sample estimates:
mean of x mean of y
 98.98282  75.42480
```

RNN Hybrid is better than RNN.

t – test on RMSE between FFNN and RNN 6-6-6

```
> t.test(m4$V1, m4$V3, alternative = "greater");

	Welch Two Sample t-test

data:  m4$V1 and m4$V3
t = -0.8317, df = 32.984, p-value = 0.7942
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -18.59961      Inf
sample estimates:
mean of x mean of y
 92.85432  98.98282
```

FFNN is better than RNN.

t – test on RMSE between FFNN Hybrid and RNN Hybrid 6-6-6

```
> t.test(m4$V2, m4$V4, alternative = "greater");

	Welch Two Sample t-test

data:  m4$V2 and m4$V4
t = -5.7834, df = 29.313, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -55.65266       Inf
sample estimates:
mean of x mean of y
 32.40624  75.42480
```

FFNN Hybrid is better than RNN Hybrid

Experiment 5 (Network topology: 12-6-1)

Number of input unit	:	12
Number of hidden unit	:	6
Number of output unit	:	1
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 12-6-1

```
> t.test(m5$V1, m5$V2, alternative = "greater");

	Welch Two Sample t-test

data:  m5$V1 and m5$V2
t = 27.8691, df = 39.361, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 10.0153     Inf
sample estimates:
mean of x mean of y
 21.99446  11.33485
```

FFNN Hybrid is better than FFNN.

t – test on RMSE between RNN and RNN Hybrid 12-6-1

```
> t.test(m5$V3, m5$V4, alternative = "greater");

	Welch Two Sample t-test

data:  m5$V3 and m5$V4
t = -2.0731, df = 39.619, p-value = 0.9776
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -34.76548       Inf
sample estimates:
mean of x mean of y
 48.38854  67.57036
```

RNN is better than RNN Hybrid.

t – test on RMSE between FFNN and RNN 12-6-1

```
> t.test(m5$V1, m5$V3, alternative = "greater");

	Welch Two Sample t-test

data:  m5$V1 and m5$V3
t = -7.1121, df = 29.525, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -32.69612       Inf
sample estimates:
mean of x mean of y
 21.99446  48.38854
```

FFNN is better than RNN.

t – test on RMSE between FFNN Hybrid and RNN Hybrid 12-6-1

```
> t.test(m5$V2, m5$V4, alternative = "greater");

	Welch Two Sample t-test

data:  m5$V2 and m5$V4
t = -6.6281, df = 29.018, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -70.65131       Inf
sample estimates:
mean of x mean of y
 11.33485  67.57036
```

FFNN Hybrid is better than RNN Hybrid

Experiment 6 (Network topology: 12-12-4)

Number of input unit	:	12
Number of hidden unit	:	12

Number of output unit	:	4
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 12-12-4

```
> t.test(m6$V1, m6$V2, alternative = "greater");

	Welch Two Sample t-test

data:  m6$V1 and m6$V2
t = 32.9177, df = 32.401, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 14.74236      Inf
sample estimates:
mean of x mean of y
 27.29597  11.75415
```

FFNN Hybrid is better than FFNN.

t – test on RMSE between RNN and RNN Hybrid 12-12-4

```
> t.test(m6$V3, m6$V4, alternative = "greater");

	Welch Two Sample t-test

data:  m6$V3 and m6$V4
t = -10.3531, df = 36.395, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -71.19975       Inf
sample estimates:
mean of x mean of y
 54.22643 115.44591
```

RNN is better than RNN Hybrid.

t – test on RMSE between FFNN and RNN 12-12-4

```
> t.test(m6$V1, m6$V3, alternative = "greater");

	Welch Two Sample t-test

data:  m6$V1 and m6$V3
t = -13.1045, df = 32.035, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -30.41138       Inf
sample estimates:
mean of x mean of y
 27.29597  54.22643
```

FFNN is better than RNN.

t – test on RMSE between FFNN Hybrid and RNN Hybrid 12-12-4

```
> t.test(m6$V2, m6$V4, alternative = "greater");

	Welch Two Sample t-test

data:  m6$V2 and m6$V4
t = -18.6342, df = 29.023, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -113.1465       Inf
sample estimates:
mean of x mean of y
 11.75415 115.44591
```

FFNN Hybrid is better than RNN Hybrid.

Experiment 7 (Network topology: 12-12-4)

Number of input unit	:	12
Number of hidden unit	:	12
Number of output unit	:	4
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 12-18-12

```
> t.test(m7$V1, m7$V2, alternative = "greater");

	Welch Two Sample t-test

data:  m7$V1 and m7$V2
t = 34.0023, df = 30.472, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 27.20842      Inf
sample estimates:
mean of x mean of y
 43.60101  14.96384
```

FFNN Hybrid is better than FFNN.

t – test on RMSE between RNN and RNN Hybrid 12-18-12

```
> t.test(m7$V3, m7$V4, alternative = "greater");

	Welch Two Sample t-test

data:  m7$V3 and m7$V4
t = -4.7293, df = 34.404, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -35.50064       Inf
sample estimates:
mean of x mean of y
 56.10933  82.26224
```

RNN is better than RNN Hybrid.

t – test on RMSE between FFNN and RNN 12-18-12

```
> t.test(m7$V1, m7$V3, alternative = "greater");

	Welch Two Sample t-test

data:  m7$V1 and m7$V3
t = -6.8654, df = 43.279, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -15.5707       Inf
sample estimates:
mean of x mean of y
 43.60101  56.10933
```

FFNN is better than RNN.

t – test on RMSE between FFNN Hybrid and RNN Hybrid 12-18-12

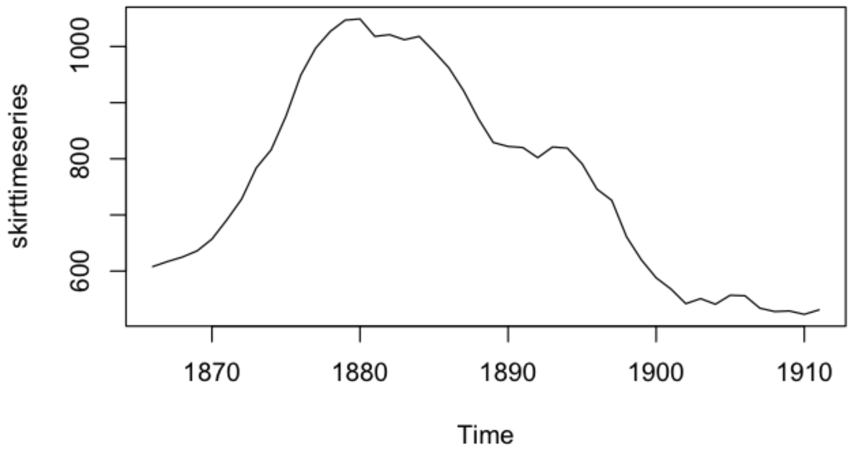
```
> t.test(m7$V2, m7$V4, alternative = "greater");

	Welch Two Sample t-test

data:  m7$V2 and m7$V4
t = -12.7249, df = 29.036, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -76.28426       Inf
sample estimates:
mean of x mean of y
 14.96384  82.26224
```

FFNN Hybrid is better than RNN Hybrid.

Skirt Diameter Size Data Testing



Data Training : 1866 - 1889

Data Testing : 1890 - 1899

Data Validation : 1900 - 1910

Experiment 1 (Network topology: 1-5-1)

Number of input unit	:	1
Number of hidden unit	:	5
Number of output unit	:	1
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 1-5-1

```
> t.test(s$V1, s$V2, alternative = "greater");

	Welch Two Sample t-test

data:  s$V1 and s$V2
t = -23.8918, df = 45.255, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -36.76074       Inf
sample estimates:
mean of x mean of y
 37.47821  71.82491
```

FFNN is better than FFNN Hybrid.

t – test on RMSE between RNN and RNN Hybrid 1-5-1

```
> t.test(s$V3, s$V4, alternative = "greater");

    Welch Two Sample t-test

data:  s$V3 and s$V4
t = -4.3218, df = 42.069, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -144.9605       Inf
sample estimates:
mean of x mean of y
 184.9941  289.3450
```

RNN is better than RNN Hybrid.

t – test on RMSE between FFNN and RNN 1-5-1

```
> t.test(s$V1, s$V3, alternative = "greater");

    Welch Two Sample t-test

data:  s$V1 and s$V3
t = -13.9019, df = 29.251, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -165.5405       Inf
sample estimates:
mean of x mean of y
 37.47821 184.99408
```

FFNN is better than RNN.

t – test on RMSE between FFNN Hybrid and RNN Hybrid 1-5-1

```
> t.test(s$V2, s$V4, alternative = "greater");

    Welch Two Sample t-test

data:  s$V2 and s$V4
t = -10.0073, df = 29.195, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -254.4444       Inf
sample estimates:
mean of x mean of y
 71.82491 289.34504
```

FFNN Hybrid is better than RNN Hybrid

Experiment 2 (Network topology: 3-5-1)

Number of input unit : 3

Number of hidden unit	:	5
Number of output unit	:	1
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 3-5-1

```
> t.test(s2$V1, s2$V2, alternative = "greater");

    Welch Two Sample t-test

data:  s2$V1 and s2$V2
t = -11.4511, df = 36.02, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -41.56047       Inf
sample estimates:
mean of x mean of y
 47.68896  83.90933
```

FFNN is better than FFNN Hybrid.

t – test on RMSE between RNN and RNN Hybrid 3-5-1

```
> t.test(s2$V3, s2$V4, alternative = "greater");

    Welch Two Sample t-test

data:  s2$V3 and s2$V4
t = -5.3096, df = 38.04, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -132.5617       Inf
sample estimates:
mean of x mean of y
 187.8544  288.4686
```

RNN is better than RNN Hybrid.

t – test on RMSE between FFNN and RNN 3-5-1

```
> t.test(s2$V1, s2$V3, alternative = "greater");

    Welch Two Sample t-test

data:  s2$V1 and s2$V3
t = -19.7079, df = 30.282, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -152.233      Inf
sample estimates:
mean of x mean of y
 47.68896 187.85436
```

FFNN is better than RNN.

t – test on RMSE between FFNN Hybrid and RNN Hybrid 3-5-1

```
> t.test(s2$V2, s2$V4, alternative = "greater");

	Welch Two Sample t-test

data:  s2$V2 and s2$V4
t = -11.4619, df = 30.668, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -234.8291       Inf
sample estimates:
mean of x mean of y
 83.90933 288.46864
```

FFNN Hybrid is better than RNN Hybrid

Experiment 3 (Network topology: 3-5-3)

Number of input unit	:	3
Number of hidden unit	:	5
Number of output unit	:	3
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 3-5-3

```
> t.test(s3$V1, s3$V2, alternative = "greater");

	Welch Two Sample t-test

data:  s3$V1 and s3$V2
t = -11.8829, df = 39.493, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -40.68215       Inf
sample estimates:
mean of x mean of y
 78.20819 113.83968
```

FFNN is better than FFNN Hybrid.

t – test on RMSE between RNN and RNN Hybrid 3-5-3

```
> t.test(s3$V3, s3$V4, alternative = "greater");

	Welch Two Sample t-test

data:  s3$V3 and s3$V4
t = -9.6858, df = 33.623, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -256.9884       Inf
sample estimates:
mean of x mean of y
 212.4153  431.1974
```

RNN is better than RNN Hybrid.

t – test on RMSE between FFNN and RNN 3-5-3

```
> t.test(s3$V1, s3$V3, alternative = "greater");

	Welch Two Sample t-test

data:  s3$V1 and s3$V3
t = -21.4053, df = 31.168, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -144.8359       Inf
sample estimates:
mean of x mean of y
 78.20819 212.41528
```

FFNN is better than RNN.

t – test on RMSE between FFNN Hybrid and RNN Hybrid 3-5-3

```
> t.test(s3$V2, s3$V4, alternative = "greater");

	Welch Two Sample t-test

data:  s3$V2 and s3$V4
t = -14.487, df = 29.93, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -354.5414       Inf
sample estimates:
mean of x mean of y
 113.8397  431.1974
```

FFNN Hybrid is better than RNN Hybrid

Experiment 4 (Network topology: 6-12-1)

Number of input unit	:	6
Number of hidden unit	:	12

Number of output unit	:	1
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 6-12-1

```
> t.test(s4$V1, s4$V2, alternative = "greater");

	Welch Two Sample t-test

data:  s4$V1 and s4$V2
t = -24.4345, df = 47.737, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -42.96749       Inf
sample estimates:
mean of x mean of y
 61.86009 102.06740
```

FFNN is better than FFNN Hybrid.

t – test on RMSE between RNN and RNN Hybrid 6-12-1

```
> t.test(s4$V3, s4$V4, alternative = "greater");

	Welch Two Sample t-test

data:  s4$V3 and s4$V4
t = -15.2933, df = 37.938, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -324.3316       Inf
sample estimates:
mean of x mean of y
 305.3445  597.4705
```

RNN is better than RNN Hybrid.

t – test on RMSE between FFNN and RNN 6-12-1

```
> t.test(s4$V1, s4$V3, alternative = "greater");

	Welch Two Sample t-test

data:  s4$V1 and s4$V3
t = -34.2639, df = 29.846, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -255.5474       Inf
sample estimates:
mean of x mean of y
 61.86009 305.34454
```

FFNN is better than RNN.

t – test on RMSE between FFNN Hybrid and RNN Hybrid 6-12-1

```
> t.test(s4$V2, s4$V4, alternative = "greater");

	Welch Two Sample t-test

data:  s4$V2 and s4$V4
t = -27.8211, df = 29.365, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -525.6466       Inf
sample estimates:
mean of x mean of y
 102.0674  597.4705
```

FFNN Hybrid is better than RNN Hybrid.

Experiment 5 (Network topology: 6-12-4)

Number of input unit	:	6
Number of hidden unit	:	12
Number of output unit	:	4
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 6-12-4

```
> t.test(s5$V1, s5$V2, alternative = "greater");

	Welch Two Sample t-test

data:  s5$V1 and s5$V2
t = -70.4955, df = 45.679, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -188.9757       Inf
sample estimates:
mean of x mean of y
 226.1736  410.7534
```

FFNN is better than FFNN Hybrid.

t – test on RMSE between RNN and RNN Hybrid 6-12-4

```
> t.test(s5$V3, s5$V4, alternative = "greater");

	Welch Two Sample t-test

data:  s5$V3 and s5$V4
t = -24.4484, df = 33.199, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -338.4728       Inf
sample estimates:
mean of x mean of y
 386.0682  702.6317
```

RNN is better than RNN Hybrid.

t – test on RMSE between FFNN and RNN 6-12-4

```
> t.test(s5$V1, s5$V3, alternative = "greater");

	Welch Two Sample t-test

data:  s5$V1 and s5$V3
t = -44.3114, df = 37.229, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -165.9814       Inf
sample estimates:
mean of x mean of y
 226.1736  386.0682
```

FFNN is better than RNN.

t – test on RMSE between FFNN Hybrid and RNN Hybrid 6-12-4

```
> t.test(s5$V2, s5$V4, alternative = "greater");

	Welch Two Sample t-test

data:  s5$V2 and s5$V4
t = -22.9682, df = 30.931, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -313.4263       Inf
sample estimates:
mean of x mean of y
 410.7534  702.6317
```

FFNN Hybrid is better than RNN Hybrid.

Experiment 6 (Network topology: 4-6-1)

Number of input unit : 4

Number of hidden unit	:	6
Number of output unit	:	1
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 4-6-1

```
> t.test(s6$V1, s6$V2, alternative = "greater");

	Welch Two Sample t-test

data:  s6$V1 and s6$V2
t = -4.6542, df = 43.785, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -22.60242       Inf
sample estimates:
mean of x mean of y
 54.69559  71.30220
```

FFNN is better than FFNN Hybrid.

t – test on RMSE between RNN and RNN Hybrid 4-6-1

```
> t.test(s6$V3, s6$V4, alternative = "greater");

	Welch Two Sample t-test

data:  s6$V3 and s6$V4
t = -6.7377, df = 38.016, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -167.944       Inf
sample estimates:
mean of x mean of y
 203.0088  337.3397
```

RNN is better than RNN Hybrid.

t – test on RMSE between FFNN and RNN 4-6-1

```
> t.test(s6$V1, s6$V3, alternative = "greater");

    Welch Two Sample t-test

data:  s6$V1 and s6$V3
t = -19.5776, df = 31.899, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -161.1467       Inf
sample estimates:
mean of x mean of y
 54.69559 203.00875
```

FFNN is better than RNN.

t – test on RMSE between FFNN Hybrid and RNN Hybrid 4-6-1

```
> t.test(s6$V2, s6$V4, alternative = "greater");

    Welch Two Sample t-test

data:  s6$V2 and s6$V4
t = -14.163, df = 30.689, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -297.896       Inf
sample estimates:
mean of x mean of y
  71.3022  337.3397
```

FFNN Hybrid is better than RNN Hybrid.

Experiment 7 (Network topology: 6-4-3)

Number of input unit	:	6
Number of hidden unit	:	4
Number of output unit	:	3
Learning rate (eta)	:	0.25
Alpha	:	0.25
Training Length (epoch)	:	500

t – test on RMSE between FFNN and FFNN Hybrid 6-4-3

```
> t.test(s7$V1, s7$V2, alternative = "greater");

	Welch Two Sample t-test

data:  s7$V1 and s7$V2
t = -22.012, df = 54.513, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -133.0005       Inf
sample estimates:
mean of x mean of y
 185.8330  309.4375
```

FFNN is better than FFNN Hybrid.

t – test on RMSE between RNN and RNN Hybrid 6-4-3

```
> t.test(s7$V3, s7$V4, alternative = "greater");

	Welch Two Sample t-test

data:  s7$V3 and s7$V4
t = -17.6838, df = 36.32, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -236.8054       Inf
sample estimates:
mean of x mean of y
 350.7377  566.9096
```

RNN is better than RNN Hybrid.

t – test on RMSE between FFNN and RNN 6-4-3

```
> t.test(s7$V1, s7$V3, alternative = "greater");

	Welch Two Sample t-test

data:  s7$V1 and s7$V3
t = -30.7447, df = 56.157, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -173.8751       Inf
sample estimates:
mean of x mean of y
 185.8330  350.7377
```

FFNN is better than RNN.

t – test on RMSE between FFNN Hybrid and RNN Hybrid 6-4-3

```
> t.test(s7$V2, s7$V4, alternative = "greater");

	Welch Two Sample t-test

data:  s7$V2 and s7$V4
t = -20.8703, df = 37.462, p-value = 1
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -278.2788       Inf
sample estimates:
mean of x mean of y
 309.4375  566.9096
```

FFNN Hybrid is better than RNN Hybrid

Data Testing Summary

Table 1, 2, and 3 shows the comparison result of 3 different data sets on seven network topologies.

Table 1 FFNN, FFNN Hybrid, RNN and RNN Hybrid comparison on New York Birth data set using different topologies

		Network Topology (NYB data set)						
		1-6-1	6-6-1	6-6-4	6-6-6	12-6-1	12-12-4	12-18-12
Comparison	FFNN vs FFNN Hybrid	FFFN Hybrid	FFFN Hybrid	FFFN Hybrid	FFFN Hybrid	FFFN Hybrid	FFFN Hybrid	FFFN Hybrid
	RNN vs RNN Hybrid	RNN Hybrid	RNN Hybrid	RNN Hybrid	RNN Hybrid	RNN	RNN	RNN Hybrid
	FFNN vs RNN	RNN	RNN	RNN	FFNN	FFNN	FFNN	FFNN
	FFNN Hybrid vs RNN Hybrid	FFNN Hybrid	FFNN Hybrid	FFNN Hybrid	FFNN Hybrid	FFNN Hybrid	FFNN Hybrid	FFNN Hybrid

Table 2 FFNN, FFNN Hybrid, RNN and RNN Hybrid comparison on Milk Production data set using different topologies

		Network Topology (Milk Production data set)						
		1-6-1	6-6-1	6-6-4	6-6-6	12-6-1	12-12-4	12-18-12
Comparison	FFNN vs FFNN Hybrid	FFFN Hybrid	FFFN Hybrid	FFFN Hybrid	FFFN Hybrid	FFFN Hybrid	FFFN Hybrid	FFFN Hybrid
	RNN vs RNN Hybrid	RNN Hybrid	RNN Hybrid	RNN Hybrid	RNN Hybrid	RNN	RNN	RNN
	FFNN vs RNN	FFNN	FFNN	FFNN	FFNN	FFNN	FFNN	FFNN
	FFNN Hybrid vs RNN Hybrid	FFNN Hybrid	FFNN Hybrid	FFNN Hybrid	FFNN Hybrid	FFNN Hybrid	FFNN Hybrid	FFNN Hybrid

Table 3 FFNN, FFNN Hybrid, RNN and RNN Hybrid comparison on Skirt Size data set using different topologies

		Network Topology (Skirt Size Data Set)						
		1-5-1	3-5-1	3-5-3	16-12-1	6-12-4	4-6-1	6-4-3
Comparison	FFNN vs FFNN Hybrid	FFFN Hybrid	FFFN Hybrid	FFFN Hybrid	FFFN Hybrid	FFFN Hybrid	FFFN Hybrid	FFFN Hybrid
	RNN vs RNN Hybrid	RNN	RNN	RNN	RNN	RNN	RNN	RNN
	FFNN vs RNN	FFNN	FFNN	FFNN	FFNN	FFNN	FFNN	FFNN
	FFNN Hybrid vs RNN Hybrid	FFNN Hybrid	FFNN Hybrid	FFNN Hybrid	FFNN Hybrid	FFNN Hybrid	FFNN Hybrid	FFNN Hybrid

Conclusion

Based on the data testing results, we can conclude that for time series forecasting, FFNN Hybrid is better than FFNN, RNN and RNN Hybrid especially when the time series data contains trends and seasonal pattern like in NYB and Milk Production data set. However when the data set shows irregularity like in Skirt Size data set, FNNN performs better than the other methods. Combining neural network and ARIMA on irregular time series data does not improve the performance.

The performance of each method also depends on the network topology and the characteristic of time series data. For time series with frequently up and down pattern like NYB, FFNN hybrid yields the best performance in term of forecast error. However if we need to shorten the computation time we can consider using regular FFNN or RNN. RNN can outperform FFNN when the number of input and output unit is relatively small. We can consider using RNN than FFNN if we only have small number of input unit and we want to do single step forecast or short multistep forecast. However, when the number of input unit is large, FFNN outperforms RNN. RNN Hybrid will work better than RNN when the number of input unit is small but when the number of input unit is larger, combining RNN and ARIMA does not seem as better approach.

Different result comes from time series with regular seasonal pattern like Milk Production data set. In this case FFNN Hybrid also has the smallest RMSE, MAE and MAPE compare to the other approaches. However if we want to compare FFNN or RNN without adding linear method then FFNN outperforms RNN. Although adding ARIMA works better in FFNN, it is not the case if we combine it with RNN. RNN hybrid outperforms RNN only when it is applied to single or short multistep forecast with small number of input unit.

Future Works

In this experiment, I only use one set of training length, learning rate and alpha. For future works, we can expand this experiment by comparing the result from different epoch and learning parameter. Applying FFNN, FFNN Hybrid, RNN and RNN Hybrid to more various time series data is also necessary to substantiate the conclusion. We can also examine the performance of these 4 approaches when applied to step-wise forecast **[4].**

Since the recurrent neural used in this experiment is simple recurrent network with first order back propagation through time algorithm, we can try to combine ARIMA with more sophisticated type of recurrent neural network.

References

[1]Engelbrecht A. P., "Supervised Learning Neural Networks," in *Computational Intelligence An Introduction*, 2'nd ed: John Wiley and Sons, Ltd, 2007, pp. 27-54.

[2]Haykin S., "Dynamically Driven Recurrent Networks," in *Neural Networks and Learning Machines,* 3'rd ed New Jersey: Pearson Education, Inc., 2008.

[3]Purwanto, Eswaran C. , and Longeswaran R., "An enhanced hybrid method for time series prediction using linear and neural network models," *Applied Intelligence*, vol. 37, pp. 511-519, 2012.

[4]Tang Z. and Fishwick P. A., "Feedforward Neural Nets as Models for Time Series Forecasting," *ORSA Journal on Computing*, vol. 5, pp. 374-385, 1993.

[5]Hyndman J. Rob and Khandakar Yeasmin, "Automatic Time Series Forecasting: The Forecast Package for R," *Journal of Statistical Software*, vol. 27, 2008.

[6]Priddy Kevin L. and Keller Paul E., "Min-Max Normalization," in *Artificial neural networks : an introduction*, Bellingham, Washington: SPIE- The International Society for Optical Engineering, 2005, p. 16.

[7](05/01/2013). *Introduction to ARIMA: nonseasonal models*. Available: http://people.duke.edu/~rnau/411arim.htm

Appendix

Comparison.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package timeseriesforecast;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintStream;
import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;
import rcaller.RCaller;
import rcaller.RCode;

/**
 *
 * @author Ega
 */
public class Comparison {

    private int numOfInputUnit;
    private int numOfHiddenUnit;
    private int numOfOutputUnit;
    private double[][] V_FFNN;
    private double[][] W_FFNN;
    private double[][] V_RNN;
    private double[][] W_RNN;
    private double[][] U_RNN;
    private double[][] V_FFNNH;
    private double[][] W_FFNNH;
    private double[][] V_RNNH;
    private double[][] W_RNNH;
    private double[][] U_RNNH;
    private double eta;
    private double alpha;
    private int maxEpoch;
    private double maxError;
    private int weightInitialization;

    public Comparison(int weightInitialization, int numOfInputUnit, int numOfHiddenUnit, int numOfOutputUnit,
double eta, double alpha, int maxEpoch, double maxError) {
        this.numOfInputUnit = numOfInputUnit;
        this.numOfHiddenUnit = numOfHiddenUnit;
        this.numOfOutputUnit = numOfOutputUnit;
        this.eta = eta;
        this.alpha = alpha;
        this.maxEpoch = maxEpoch;
        this.maxError = maxError;
        this.weightInitialization = weightInitialization;
    }

    public Vector getDataSetNYB() {
        Vector results = new Vector();
        double[] trainingSet;
        double[] testingSet;
        double[] validationSet;
        try {
            /* Creating a RCaller */
            RCaller caller = new RCaller();
```

```
            caller.setRscriptExecutable("/usr/bin/Rscript");

            /* Creating a source code */
            RCode code = new RCode();
            code.clear();

            // add libraries needed to load data set
            code.addRCode("library(zoo)");
            code.addRCode("library(timeSeries)");
            code.addRCode("library(rdatamarket)");

            //get data for training, testing and validation set
            code.addRCode("dminit(\"be0123ef782e49348a7ed53c2444c08c\")");
            code.addRCode("dataSet <- dmlist(\"22nv\")");
            code.addRCode("trainingSet <- dataSet[1:96,2]");
            code.addRCode("testingSet <- dataSet[97:132,2]");
            code.addRCode("validationSet <- dataSet[133:168,2]");
            code.addRCode("results<-list(trainSet = trainingSet, testSet = testingSet, validSet = validationSet, data =
dataSet[,2])");
            caller.setRCode(code);
            System.out.println("script exe" + caller.getRscriptExecutable());
            caller.runAndReturnResult("results");
            trainingSet = caller.getParser().getAsDoubleArray("trainSet");
            testingSet = caller.getParser().getAsDoubleArray("testSet");
            validationSet = caller.getParser().getAsDoubleArray("validSet");
            double[] allData = caller.getParser().getAsDoubleArray("data");
            results.addElement(trainingSet);
            results.addElement(testingSet);
            results.addElement(validationSet);
            results.addElement(allData);
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        return results;
    }

    public double [] HybridFFNN(double[][] V, double[][] W) {
        TimeSeriesHybridFFNN tsnn = new TimeSeriesHybridFFNN(V, W);
        Vector dataSet = getDataSetNYB();
        tsnn.setMinMax((double[]) dataSet.elementAt(3));

        double[] trainingSet = tsnn.normalizeData((double[]) dataSet.elementAt(0));
        double[] testingSet = tsnn.normalizeData((double[]) dataSet.elementAt(1));
        double[] validationSet = tsnn.normalizeData((double[]) dataSet.elementAt(2));

        double[] RMSE = tsnn.TrainingNN(weightInitialization, trainingSet, testingSet, numOfInputUnit,
numOfHiddenUnit, numOfOutputUnit, eta, alpha, maxEpoch, maxError);
        try {
         RCaller caller = new RCaller();
         caller.setRscriptExecutable("/usr/bin/Rscript");
         caller.cleanRCode();
         File file;
         String[] arr = new String[1];
         arr[0] = "New York Birth";
         file = caller.startPlot();
         caller.addDoubleArray("RMSE.hybrid.ffnn", RMSE);
         caller.addStringArray("arr", arr);
         caller.addRCode("a = arr[1]");
         caller.addRCode("plot.ts(RMSE.hybrid.ffnn, main=a)");
         caller.endPlot();
         caller.runOnly();
         caller.showPlot(file);
         } catch (IOException ex) {
         Logger.getLogger(FFNN.class.getName()).log(Level.SEVERE, null, ex);
         }
```

```
        Vector forecastError = tsnn.generalizationNN(validationSet,numOfInputUnit, numOfHiddenUnit,
numOfOutputUnit,eta, alpha);
        double [] SE = (double[]) forecastError.elementAt(0);
        double [] absE = (double[]) forecastError.elementAt(1);
        double [] absPE = (double[]) forecastError.elementAt(2);
        double RMSEValidation = tsnn.calculateForecastError(SE);
        double MAE = tsnn.calculateMAE(absE);
        double MAPE = tsnn.calculateMAPE(absPE);
        //System.out.println("Forecast RMSE = "+RMSEValidation);
        double [] result = new double[3];
        result[0] = RMSEValidation;
        result[1] = MAE;
        result[2] = MAPE;
        return result;
    }

    public double [] RNN(double[][] V, double[][] W, double[][] U) {
        TimeSeriesRNN tsnn = new TimeSeriesRNN(V, W, U);
        Vector dataSet = getDataSetNYB();
        tsnn.setMinMax((double[]) dataSet.elementAt(3));
        double[] trainingSet = tsnn.normalizeData((double[]) dataSet.elementAt(0));
        double[] testingSet = tsnn.normalizeData((double[]) dataSet.elementAt(1));
        double[] validationSet = tsnn.normalizeData((double[]) dataSet.elementAt(2));
        double[] RMSE = tsnn.TrainingNN(weightInitialization, trainingSet, testingSet, numOfInputUnit,
numOfHiddenUnit, numOfOutputUnit, eta, alpha, maxEpoch, maxError);

         try {
            RCaller caller = new RCaller();
            caller.setRscriptExecutable("/usr/bin/Rscript");
            caller.cleanRCode();
            File file;
            String[] arr = new String[1];
            arr[0] = "New York Birth";
            file = caller.startPlot();
            caller.addDoubleArray("RMSE.rnn", RMSE);
            caller.addStringArray("arr", arr);
            caller.addRCode("a = arr[1]");
            caller.addRCode("plot.ts(RMSE.rnn, main=a)");
            caller.endPlot();
            caller.runOnly();
            caller.showPlot(file);
        } catch (IOException ex) {
            Logger.getLogger(FFNN.class.getName()).log(Level.SEVERE, null, ex);
        }


        Vector forecastError = tsnn.generalizationNN(validationSet,numOfInputUnit, numOfHiddenUnit,
numOfOutputUnit,eta, alpha);
        double [] SE = (double[]) forecastError.elementAt(0);
        double [] absE = (double[]) forecastError.elementAt(1);
        double [] absPE = (double[]) forecastError.elementAt(2);
        double RMSEValidation = tsnn.calculateForecastError(SE);
        double MAE = tsnn.calculateMAE(absE);
        double MAPE = tsnn.calculateMAPE(absPE);
        //System.out.println("Forecast RMSE = "+RMSEValidation);
        double [] result = new double[3];
        result[0] = RMSEValidation;
        result[1] = MAE;
        result[2] = MAPE;
        return result;
    }

    public double[] FFNN(double[][] V, double[][] W) {
        TimeSeriesNN tsnn = new TimeSeriesNN(V, W);
        Vector dataSet = getDataSetNYB();
        tsnn.setMinMax((double[]) dataSet.elementAt(3));
```

```
        double[] trainingSet = tsnn.normalizeData((double[]) dataSet.elementAt(0));
        double[] testingSet = tsnn.normalizeData((double[]) dataSet.elementAt(1));
        double[] validationSet = tsnn.normalizeData((double[]) dataSet.elementAt(2));

        double[] RMSE = tsnn.TrainingNN(weightInitialization, trainingSet, testingSet, numOfInputUnit,
numOfHiddenUnit, numOfOutputUnit, eta, alpha, maxEpoch, maxError);

        try {
            RCaller caller = new RCaller();
            caller.setRscriptExecutable("/usr/bin/Rscript");
            caller.cleanRCode();
            File file;
            String[] arr = new String[1];
            arr[0] = "New York Birth";
            file = caller.startPlot();
            caller.addDoubleArray("RMSE.ffnn", RMSE);
            caller.addStringArray("arr", arr);
            caller.addRCode("a = arr[1]");
            caller.addRCode("plot.ts(RMSE.ffnn, main=a)");
            caller.endPlot();
            caller.runOnly();
            caller.showPlot(file);
        } catch (IOException ex) {
            Logger.getLogger(FFNN.class.getName()).log(Level.SEVERE, null, ex);
        }


        Vector forecastError = tsnn.generalizationNN(validationSet,numOfInputUnit, numOfHiddenUnit,
numOfOutputUnit,eta, alpha);
        double [] SE = (double[]) forecastError.elementAt(0);
        double [] absE = (double[]) forecastError.elementAt(1);
        double [] absPE = (double[]) forecastError.elementAt(2);
        double RMSEValidation = tsnn.calculateForecastError(SE);
        double MAE = tsnn.calculateMAE(absE);
        double MAPE = tsnn.calculateMAPE(absPE);
        //System.out.println("Forecast RMSE = "+RMSEValidation);
        double [] result = new double[3];
        result[0] = RMSEValidation;
        result[1] = MAE;
        result[2] = MAPE;
        return result;
    }

    public double [] HybridRNN(double[][] V, double[][] W, double[][] U) {
        TimeSeriesHybridRNN tsnn = new TimeSeriesHybridRNN(V, W, U);
        Vector dataSet = getDataSetNYB();
        tsnn.setMinMax((double[]) dataSet.elementAt(3));
        double[] trainingSet = tsnn.normalizeData((double[]) dataSet.elementAt(0));
        double[] testingSet = tsnn.normalizeData((double[]) dataSet.elementAt(1));
        double[] validationSet = tsnn.normalizeData((double[]) dataSet.elementAt(2));

        double[] RMSE = tsnn.TrainingNN(weightInitialization, trainingSet, testingSet, numOfInputUnit,
numOfHiddenUnit, numOfOutputUnit, eta, alpha, maxEpoch, maxError);

        try {
         RCaller caller = new RCaller();
         caller.setRscriptExecutable("/usr/bin/Rscript");
         caller.cleanRCode();
         File file;
         String [] arr = new String[1];
         arr[0] = "New York Birth";
         file = caller.startPlot();
         caller.addDoubleArray("RMSE.hybrid.rnn", RMSE);
         caller.addStringArray("arr", arr);
         caller.addRCode("a = arr[1]");
         caller.addRCode("plot.ts(RMSE.hybrid.rnn, main=a)");
         caller.endPlot();
```

```
        caller.runOnly();
        caller.showPlot(file);
        } catch (IOException ex) {
        Logger.getLogger(FFNN.class.getName()).log(Level.SEVERE, null, ex);
        }


        Vector forecastError = tsnn.generalizationNN(validationSet,numOfInputUnit, numOfHiddenUnit,
numOfOutputUnit,eta, alpha);
        double [] SE = (double[]) forecastError.elementAt(0);
        double [] absE = (double[]) forecastError.elementAt(1);
        double [] absPE = (double[]) forecastError.elementAt(2);
        double RMSEValidation = tsnn.calculateForecastError(SE);
        double MAE = tsnn.calculateMAE(absE);
        double MAPE = tsnn.calculateMAPE(absPE);
        double [] result = new double[3];
        result[0] = RMSEValidation;
        result[1] = MAE;
        result[2] = MAPE;
        return result;
    }

    public static void main(String[] args) {
        int I = 12;
        int J = 18;
        int K = 12;
        double eta = 0.25;
        double alpha = 0.25;
        int maxEpoch = 500;
        double maxError = 0.5;
        int choice = 1;
        double[][] experiment = new double[30][12];

        for (int i = 0; i < 30; i++) {
            WeightsInitialization weight0 = new WeightsInitialization();
            Comparison result = new Comparison(choice, I, J, K, eta, alpha, maxEpoch, maxError);
            double[][] V = weight0.useRandomWeight(I, J);
            double[][] W = weight0.useRandomWeight(J, K);
            double[][] U = weight0.useRandomWeightForU(J, J);
            double [] E_RNN = result.RNN(V, W, U);
            double [] E_RNNH = result.HybridRNN(V, W, U);
            double [] E_FFNN = result.FFNN(V, W);
            double [] E_FFNNH = result.HybridFFNN(V, W);
            experiment[i][0] = E_FFNN [0];
            experiment[i][1] = E_FFNNH [0];
            experiment[i][2] = E_RNN[0];
            experiment[i][3] = E_RNNH[0];
            experiment[i][4] = E_FFNN [1];
            experiment[i][5] = E_FFNNH [1];
            experiment[i][6] = E_RNN[1];
            experiment[i][7] = E_RNNH[1];
            experiment[i][8] = E_FFNN [2];
            experiment[i][9] = E_FFNNH [2];
            experiment[i][10] = E_RNN[2];
            experiment[i][11] = E_RNNH[2];
        }


        try (
                PrintStream output = new PrintStream(new File("/Users/Ega/Desktop/E7.txt"));) {

            for (int i = 0; i < 30; i++) {
                String sc = "";
                for (int j = 0; j < experiment[i].length; j++) {
                    if (j < 11) {
                        sc += Double.toString(experiment[i][j]) + ", ";
                    } else {
```

```
                sc += Double.toString(experiment[i][j]);
              }

            }
            output.println(sc);
          }
          output.close();

        } catch (FileNotFoundException e) {
          e.printStackTrace();
        }

    }
}
```

FFNN.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package timeseriesforecast;

/**
 *
 * @author Ega
 */
public class FFNN {

    private double[][] V;       // Weights between input and hidden layer
    private double[][] W;        // Weights between hidden and ouput layer
    private double[][] deltaV;
    private double[][] deltaW;
    private int numOfInputUnit;
    private int numOfHiddenUnit;
    private int numOfOutputUnit;
    //private int epoch;
    private double eta;
    private double alpha;

    public FFNN(int I, int J, int K, double eta, double alpha, double [][] V, double[][] W) {
        this.numOfInputUnit = I;
        this.numOfHiddenUnit = J;
        this.numOfOutputUnit = K;
        this.V = V;
        this.deltaV = new double[J][I + 1];
        this.W = W;
        this.deltaW = new double[K][J + 1];
        //this.epoch = epoch;
        this.eta = eta;
        this.alpha = alpha;
    }

    public void initializeRandomWeight(int choice) {
        /*
         * if choice = 1 generate random number between -0.5 and 0.5 for weights initialization
         */
        WeightsInitialization initialWeight = new WeightsInitialization();
        int I = numOfInputUnit;
        int J = numOfHiddenUnit;
        int K = numOfOutputUnit;

        //initialization for delta
        for (int j = 0; j < J; j++) {
            for (int i = 0; i <= I; i++) {
```

```
                deltaV[j][i] = 0.0;
            }
        }
    for (int k = 0; k < K; k++) {
        for (int j = 0; j <= J; j++) {
            deltaW[k][j] = 0.0;
        }
    }
}

public void showWeights(double[][] Weights) {
    for (int j = 0; j < Weights.length; j++) {
        for (int i = 0; i < Weights[0].length; i++) {
            int x = j + 1;
            int y = i + 1;
            System.out.printf(x + "," + y + ": " + Weights[j][i] + " ");
        }
        System.out.println("");
    }
}

public void showVector(double[] vector) {
    for (int i = 0; i < vector.length; i++) {
        System.out.print(" " + vector[i]);
    }
}

private void showTrainingSet(double[][] dataSet) {
    int m = dataSet.length;
    int n = dataSet[0].length;
    for (int j = 0; j < m; j++) {
        for (int i = 0; i < n; i++) {
            System.out.print(dataSet[j][i] + " ");
        }
        System.out.println("");
    }
}

public double[] feedforward(double[] dataPoint) {
    //forward propagation from input to hidden layer
    double[] input = createAugmentedVector(dataPoint); //z
    double[][] v = getV();
    int m = v.length;

    int n = v[0].length;
    double[] sup = sumOfProduct(input,v);
    double[] nonLinear = calculateNonLinearVector(sup);

    //forward propagation from hidden to output layer
    double[] inputForOutput = createAugmentedVector(nonLinear); //y
    double[][] w = getW();

    double[] supOutput = sumOfProduct(inputForOutput, w);
    double[] output = calculateNonLinearVector(supOutput); //O
    return output;

}

private double[] sumOfProduct2(double[] augmentedVector, double[][] weights) {
    double[] sup = new double[weights.length];
    int m = weights.length;
    int n = weights[0].length;

        for (int j = 0; j < m; j++) {
            for (int i = 0; i < n; i++) {
                sup[j] += weights[j][i] * augmentedVector[i];
            }
```

```
        }
    return sup;
}

public double[] gradientDescent(double[] dataPoint, double[] target, double eta, double alpha) {

    //forward propagation from input to hidden layer
    double[] input = createAugmentedVector(dataPoint); //z
    double[][] v = getV();
    double[] sup = sumOfProduct(input, v);
    double[] nonLinear = calculateNonLinearVector(sup);

    //forward propagation from hidden to output layer
    double[] inputForOutput = createAugmentedVector(nonLinear); //y
    double[][] w = getW();

    double[] supOutput = sumOfProduct(inputForOutput, w);
    double[] output = calculateNonLinearVector(supOutput); //O

    //backward propagation

    double[] gammaO = calculateGammaO(target, output);
    double[][] deltaWeight0 = getDeltaWeightO(gammaO, inputForOutput, eta);
    updateWeight0(alpha, deltaWeight0);

    double[] gammaY = calculateGammaY(gammaO, inputForOutput, w);
    double[][] deltaWeightY = getDeltaWeightY(gammaY, input, eta);
    updateWeightY(alpha, deltaWeightY);

    return output;
}

private void updateWeight0(double alpha, double[][] deltaWeightO) {
    double [][] weightW = getW();
    for (int k = 0; k < deltaWeightO.length; k++) {
        for (int j = 0; j < deltaWeightO[0].length; j++) {
            weightW[k][j] = weightW[k][j] + deltaWeightO[k][j] + alpha * deltaW[k][j];
            deltaW[k][j] = deltaWeightO[k][j];
        }
    }
    setW(weightW);
}

private double[][] getDeltaWeightO(double[] gammaO, double[] y, double eta) {
    double[][] deltaWeightO = new double[gammaO.length][y.length];
    for (int k = 0; k < gammaO.length; k++) {
        for (int j = 0; j < y.length; j++) {
            deltaWeightO[k][j] = -1.0 * eta * gammaO[k] * y[j];
        }
    }
    return deltaWeightO;
}

private double[] calculateGammaO(double[] target, double[] output) {
    double[] gammaO = new double[target.length];
    for (int k = 0; k < target.length; k++) {
        gammaO[k] = -1*(target[k] - output[k]) * sigmoidDerivate(output[k]);
    }
    return gammaO;
}

private void updateWeightY(double alpha, double[][] deltaWeightY) {
    double [][] weightV = getV();
    for (int j = 0; j < deltaWeightY.length - 1; j++) {
        for (int i = 0; i < deltaWeightY[0].length; i++) {
            weightV[j][i] = weightV[j][i] + deltaWeightY[j][i] + alpha * deltaV[j][i];
            deltaV[j][i] = deltaWeightY[j][i];
```

```
            }
        }
        setV(weightV);
    }

    private double[][] getDeltaWeightY(double[] gammaY, double[] z, double eta) {
        double[][] deltaWeightY = new double[gammaY.length][z.length];
        for (int j = 0; j < gammaY.length; j++) {
            for (int i = 0; i < z.length; i++) {
                deltaWeightY[j][i] = -1.0 * eta * gammaY[j] * z[i];
            }
        }
        return deltaWeightY;
    }

    private double[] calculateGammaY(double[] gammaO, double[] y, double[][] w) {
        double[] gammaY = new double[w[0].length];
        for (int j = 0; j < w[0].length; j++) {
            for (int k = 0; k < w.length; k++) {
                gammaY[j] += gammaO[k] * w[k][j] * sigmoidDerivate(y[j]);
            }
        }
        return gammaY;
    }

    private double[] calculateNonLinearVector(double[] sup) {
        double[] nonLinearVector = new double[sup.length];
        for (int i = 0; i < sup.length; i++) {
            nonLinearVector[i] = sigmoid(sup[i]);
        }
        return nonLinearVector;

    }

    private double sigmoid(double x) {
        double f = 0;
        f = 1 / (1 + Math.exp(-x));
        //System.out.println("sigmoid val" +f);
        return f;
    }

    private double sigmoidDerivate(double x) {
        double f = 0;
        f = (1 - x) * x;
        return f;
    }

   private double[] sumOfProduct(double[] augmentedVector, double[][] weights) {

        double[] sup = new double[weights.length];
        double[][] weightsT = transposeWeight(weights);
        int m = weightsT.length;
        int n = weightsT[0].length;
        if (m != augmentedVector.length) {
            System.out.println("We can not multiply this vector and matrix");
        } else {
            for (int j = 0; j < n; j++) {
                for (int i = 0; i < m; i++) {
                    sup[j] += weightsT[i][j] * augmentedVector[i];
                }
            }

        }
        return sup;
    }
```

```
private double[][] transposeWeight(double[][] weights) {
    int m = weights.length;
    int n = weights[0].length;

    double[][] weightsT = new double[n][m];
    for (int j = 0; j < m; j++) {
        for (int i = 0; i < n; i++) {
            weightsT[i][j] = weights[j][i];
        }
    }
    return weightsT;
}

private double[] createAugmentedVector(double[] originalVector) {
    int n = originalVector.length;
    double[] augmentedVector = new double[n + 1];
    for (int i = 0; i < n; i++) {
        augmentedVector[i] = originalVector[i];
    }
    augmentedVector[n] = -1;
    return augmentedVector;
}


public double calculateSE(double[] output, double[] target) {
    double SE = 0;
    int K = output.length;
    for (int k = 0; k < K; k++) {
        double e = target[k] - output[k];
        SE += e * e;

    }
    SE = SE/K;
    return SE;
}

public double calculateAbsoluteError(double[] output, double[] target) {
    double absE = 0;

    int K = output.length;
    for (int k = 0; k < K; k++) {
        double e = Math.abs(target[k] - output[k]);
        absE += e;
    }
    absE = absE/K;
    return absE;
}

public double calculateAbsolutePercentageError(double[] output, double[] target) {
    double absPE = 0;

    int K = output.length;
    for (int k = 0; k < K; k++) {
        double e = 100*(Math.abs(target[k] - output[k]))/target[k];
        absPE += e;
    }
    absPE = absPE/K;
    return absPE;
}

/**
 * @return the V
 */
public double[][] getV() {
    return V;
}
```

```
    /**
     * @param V the V to set
     */
    public void setV(double[][] V) {
        this.V = V;
    }

    /**
     * @return the W
     */
    public double[][] getW() {
        return W;
    }

    /**
     * @param W the W to set
     */
    public void setW(double[][] W) {
        this.W = W;
    }

}
```

RNN.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package timeseriesforecast;

/**
 *
 * @author Ega
 */
public class RNN {
    private double[][] V;       // Weights between input and hidden layer
    private double[][] W;        // Weights between hidden and ouput layer
    private double [][] U;
    private double[][] deltaV;
    private double[][] deltaW;
    private double[][] deltaU;
    private int numOfInputUnit;
    private int numOfHiddenUnit;
    private int numOfOutputUnit;
    //private int epoch;
    private double eta;
    private double alpha;
    private double [] hiddenUnitOutput;

    public RNN(int I, int J, int K, double eta, double alpha, double [][] V, double [][] W, double [][] U) {
        this.numOfInputUnit = I;
        this.numOfHiddenUnit = J;
        this.numOfOutputUnit = K;
        this.V = V;
        this.deltaV = new double[J][I + 1];
        this.W = W;
        this.deltaW = new double[K][J + 1];
        this.U = U;
        //System.out.println("Initial U"+U.length+"x"+U[0].length);
        this.deltaU = new double[J][J];
        this.eta = eta;
        this.alpha = alpha;
        this.hiddenUnitOutput = new double [J];
    }
```

```java
public void initializeInputFromHiddenNeuronRNN(int J){
    double[] hiddenOutput = new double [J];
    for (int j = 0; j<J; j++){
        hiddenOutput[j] = 0;
    }
    setHiddenUnitOutput(hiddenOutput);
}

public void initializeRandomWeight(int choice) {
    /*
     * if choice = 1 generate random number between -0.5 and 0.5 for weights initialization
     */

    int I = numOfInputUnit;
    int J = numOfHiddenUnit;
    int K = numOfOutputUnit;

    //initialization for delta
    for (int j = 0; j < J; j++) {
            for (int i = 0; i <= I; i++) {
                deltaV[j][i] = 0.0;
            }
        }

    for (int j = 0; j < J; j++) {
            for (int i = 0; i < J; i++) {
                deltaU[j][i] = 0.0;
            }
        }

    for (int k = 0; k < K; k++) {
        for (int j = 0; j <= J; j++) {
            deltaW[k][j] = 0.0;
        }
    }
}

public void showWeights(double[][] Weights) {
    for (int j = 0; j < Weights.length; j++) {
        for (int i = 0; i < Weights[0].length; i++) {
            int x = j + 1;
            int y = i + 1;
            System.out.printf(x + "," + y + ": " + Weights[j][i] + " ");
        }
        System.out.println("");
    }
}

public void showVector(double[] vector) {
    for (int i = 0; i < vector.length; i++) {
        System.out.print(" " + vector[i]);
    }
}

private void showTrainingSet(double[][] dataSet) {
    int m = dataSet.length;
    int n = dataSet[0].length;
    for (int j = 0; j < m; j++) {
        for (int i = 0; i < n; i++) {
            System.out.print(dataSet[j][i] + " ");
        }
        System.out.println("");
    }
}

public double[] forward(double[] dataPoint) {
    double[] output = new double [numOfOutputUnit];
```

```
    //forward propagation from input to hidden layer
    double[] input = createAugmentedVector(dataPoint); //z
    double[][] v = getV();
    double[][] u = getU();

    double [] hiddenOutput = getHiddenUnitOutput();
    //System.out.println("H" +hiddenOutput.length);
    //System.out.println("I" +input.length);
    double[] sup = sumOfProductHidden(input,hiddenOutput,v, u);
    double[] nonLinear = calculateNonLinearVector(sup);

    //forward propagation from hidden to output layer
    double[] inputForOutput = createAugmentedVector(nonLinear); //y
    double[][] w = getW();

    double[] supOutput = sumOfProduct(inputForOutput, w);
    output = calculateNonLinearVector(supOutput); //O
    return output;

}

public double[] gradientDescent(double[] dataPoint, double[] target, double eta, double alpha) {

    //forward propagation from input to hidden layer
    double[] input = createAugmentedVector(dataPoint); //z
    double[][] v = getV();
    double[][] u = getU();
    double [] hiddenOutput = getHiddenUnitOutput();
    double[] sup = sumOfProductHidden(input,hiddenOutput, v, u);
    double[] nonLinear = calculateNonLinearVector(sup);
    setHiddenUnitOutput(nonLinear);

    //forward propagation from hidden to output layer
    double[] inputForOutput = createAugmentedVector(nonLinear); //y
    double[][] w = getW();

    double[] supOutput = sumOfProduct(inputForOutput, w);
    double[] output = calculateNonLinearVector(supOutput); //O

    //backward propagation

    double[] gammaO = calculateGammaO(target, output);
    double[][] deltaWeight0 = getDeltaWeightO(gammaO, inputForOutput, eta);
    updateWeight0(alpha, deltaWeight0);

    double[] gammaY = calculateGammaY(gammaO, inputForOutput, w);
    double[][] deltaWeightY = getDeltaWeightY(gammaY, input, eta);
    //System.out.println("W " +w.length+"x"+w[0].length);
    //System.out.println("V " +v.length+"x"+v[0].length);
    //System.out.println("U " +u.length+"x"+u[0].length);
    //System.out.println("Gamma O"+gammaO.length);
    //System.out.println("Gamma Y"+gammaY.length);
    //System.out.println("deltaWeight Y " + deltaWeightY.length+ "x" +deltaWeightY[0].length);
    updateWeightY(alpha, deltaWeightY);

    double[][] deltaWeightU = getDeltaWeightY(gammaY, getHiddenUnitOutput(), eta);
    //System.out.println("deltaWeight U " + deltaWeightU.length+ "x" +deltaWeightU[0].length);
    updateWeightU(alpha, deltaWeightU);

    return output;
}

private void updateWeight0(double alpha, double[][] deltaWeightO) {
    double [][] weightW = getW();
    for (int k = 0; k < deltaWeightO.length; k++) {
        for (int j = 0; j < deltaWeightO[0].length; j++) {
```

```
                weightW[k][j] = weightW[k][j] + deltaWeightO[k][j] + alpha * deltaW[k][j];
                deltaW[k][j] = deltaWeightO[k][j];
            }
        }
        setW(weightW);
    }

    private double[][] getDeltaWeightO(double[] gammaO, double[] y, double eta) {
        double[][] deltaWeightO = new double[gammaO.length][y.length];
        for (int k = 0; k < gammaO.length; k++) {
            for (int j = 0; j < y.length; j++) {
                deltaWeightO[k][j] = -1.0 * eta * gammaO[k] * y[j];
            }
        }
        return deltaWeightO;
    }

    private double[] calculateGammaO(double[] target, double[] output) {
        double[] gammaO = new double[target.length];
        for (int k = 0; k < target.length; k++) {
            gammaO[k] = -1*(target[k] - output[k]) * sigmoidDerivate(output[k]);
        }
        return gammaO;
    }

    private void updateWeightY(double alpha, double[][] deltaWeightY) {
        double [][] weightV = getV();
        for (int j = 0; j < deltaWeightY.length - 1; j++) {
            for (int i = 0; i < deltaWeightY[0].length; i++) {
                weightV[j][i] = weightV[j][i] + deltaWeightY[j][i] + alpha * deltaV[j][i];
                deltaV[j][i] = deltaWeightY[j][i];
            }
        }
        setV(weightV);
    }

    private void updateWeightU(double alpha, double[][] deltaWeightU) {
        double [][] weightU = getU();
        for (int j = 0; j < deltaWeightU.length-1; j++) {
            for (int i = 0; i < deltaWeightU[0].length; i++) {
                weightU[j][i] = weightU[j][i] + deltaWeightU[j][i] + alpha * deltaU[j][i];
                deltaU[j][i] = deltaWeightU[j][i];
            }
        }
        setU(weightU);
    }

    private double[][] getDeltaWeightY(double[] gammaY, double[] z, double eta) {
        double[][] deltaWeightY = new double[gammaY.length][z.length];
        for (int j = 0; j < gammaY.length; j++) {
            for (int i = 0; i < z.length; i++) {
                deltaWeightY[j][i] = -1.0 * eta * gammaY[j] * z[i];
            }
        }
        return deltaWeightY;
    }

    private double[] calculateGammaY(double[] gammaO, double[] y, double[][] w) {
        double[] gammaY = new double[w[0].length];
        for (int j = 0; j < w[0].length; j++) {
            for (int k = 0; k < w.length; k++) {
                gammaY[j] += gammaO[k] * w[k][j] * sigmoidDerivate(y[j]);
            }
        }
        return gammaY;
    }
```

```
    private double[] calculateNonLinearVector(double[] sup) {
        double[] nonLinearVector = new double[sup.length];
        for (int i = 0; i < sup.length; i++) {
            nonLinearVector[i] = sigmoid(sup[i]);
        }
        return nonLinearVector;

    }

    private double sigmoid(double x) {
        double f = 0;
        f = 1 / (1 + Math.exp(-x));
        //System.out.println("sigmoid val" +f);
        return f;
    }

    private double sigmoidDerivate(double x) {
        double f = 0;
        f = (1 - x) * x;
        return f;
    }

   private double[] sumOfProduct(double[] augmentedVector, double[][] weights) {

        double[] sup = new double[weights.length];
        double[][] weightsT = transposeWeight(weights);
        int m = weightsT.length;
        int n = weightsT[0].length;
        if (m != augmentedVector.length) {
            System.out.println("We can not multiply this vector and matrix");
        } else {
            for (int j = 0; j < n; j++) {
                for (int i = 0; i < m; i++) {
                    sup[j] += weightsT[i][j] * augmentedVector[i];
                }
            }

        }
        return sup;
    }

   private double[] sumOfProductHidden(double[] augmentedVector, double[] hiddenOutput, double[][] weightsW,
double[][] weightsU) {
        double[] sup = new double[weightsW.length];
        double[][] weightsT = transposeWeight(weightsW);
        int m = weightsT.length;
        int n = weightsT[0].length;
        if (m != augmentedVector.length) {
            System.out.println("We can not multiply this vector and matrix");
        } else {
            for (int j = 0; j < n; j++) {
                for (int i = 0; i < m; i++) {
                    sup[j] += weightsT[i][j] * augmentedVector[i];
                }
            }

        }

        //System.out.println("sup.length" +sup.length +" hidden output length" +hiddenOutput.length);

        double[][] weightsTU = transposeWeight(weightsU);
        //System.out.println("U.i " +weightsTU.length+" Uj"+weightsTU[0].length);
        for (int j = 0; j < sup.length; j++) {
                for (int i = 0; i < weightsTU[0].length; i++) {
                    sup[j] += weightsTU[i][j] * hiddenOutput[i];
                }
            }
```

```
    return sup;
}


private double[][] transposeWeight(double[][] weights) {
    int m = weights.length;
    int n = weights[0].length;

    double[][] weightsT = new double[n][m];
    for (int j = 0; j < m; j++) {
        for (int i = 0; i < n; i++) {
            weightsT[i][j] = weights[j][i];
        }
    }
    return weightsT;
}

private double[] createAugmentedVector(double[] originalVector) {
    int n = originalVector.length;
    double[] augmentedVector = new double[n + 1];
    for (int i = 0; i < n; i++) {
        augmentedVector[i] = originalVector[i];
    }
    augmentedVector[n] = -1;
    return augmentedVector;
}


public double calculateSE(double[] output, double[] target) {
    double SE = 0;
    int K = output.length;
    for (int k = 0; k < K; k++) {
        double e = target[k] - output[k];
        SE += e * e;

    }
    SE = SE/K;
    return SE;
}

public double calculateAbsoluteError(double[] output, double[] target) {
    double absE = 0;

    int K = output.length;
    for (int k = 0; k < K; k++) {
        double e = Math.abs(target[k] - output[k]);
        absE += e;
    }
    absE = absE/K;
    return absE;
}

public double calculateAbsolutePercentageError(double[] output, double[] target) {
    double absPE = 0;

    int K = output.length;
    for (int k = 0; k < K; k++) {
        double e = 100*(Math.abs(target[k] - output[k]))/target[k];
        absPE += e;
    }
    absPE = absPE/K;
    return absPE;
}
/**
 * @return the V
 */
public double[][] getV() {
```

```
        return V;
    }

    /**
     * @param V the V to set
     */
    public void setV(double[][] V) {
        this.V = V;
    }

    /**
     * @return the W
     */
    public double[][] getW() {
        return W;
    }

    /**
     * @param W the W to set
     */
    public void setW(double[][] W) {
        this.W = W;
    }

    /**
     * @return the U
     */
    public double[][] getU() {
        return U;
    }

    /**
     * @param U the U to set
     */
    public void setU(double[][] U) {
        this.U = U;
    }

    /**
     * @return the hiddenUnitOuput
     */
    public double[] getHiddenUnitOutput() {
        return hiddenUnitOutput;
    }

    /**
     * @param hiddenUnitOuput the hiddenUnitOuput to set
     */
    public void setHiddenUnitOutput(double[] hiddenUnitOutput) {
        this.hiddenUnitOutput = hiddenUnitOutput;
    }

}
```

TimeSeriesNN.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package timeseriesforecast;

import java.util.Vector;
import rcaller.RCaller;
import rcaller.RCode;
```

```
/**
 *
 * @author Ega
 */
public class TimeSeriesNN {

    private double[][] weightsV;
    private double[][] weightsW;
    private double minValue;
    private double maxValue;

    public TimeSeriesNN(double [][]V, double [][]W){
        this.weightsV = V;
        this.weightsW = W;
    }

    public double[] TrainingNN(int choiceW, double[] trainingSet, double[] testingSet,int numOfInputUnit, int
numOfHiddenUnit, int numOfOutputUnit, double eta, double alpha, int maxEpoch, double maxError) {
        /*
         * Learning in one-step forecast or multi-step forecast
         */
        FFNN ffnn = new FFNN(numOfInputUnit, numOfHiddenUnit, numOfOutputUnit, eta, alpha, getWeightsV(),
getWeightsW());
        ffnn.initializeRandomWeight(choiceW);
        //System.out.println("");
        //ffnn.showWeights(ffnn.getV());
        //System.out.println("");
        //ffnn.showWeights(ffnn.getW());
        //System.out.println("");
        boolean stopCondition = false;
        double forecastError = 9999999.99999;
        int epoch = 0;
        double[] SE = new double[trainingSet.length - numOfOutputUnit - numOfInputUnit +1];

        double[] RMSEtemp = new double[maxEpoch + 1];
        double[] output = new double[numOfOutputUnit];

        while (stopCondition != true) {

            for (int l = 0; l < (trainingSet.length - numOfOutputUnit - numOfInputUnit +1); l++) {
                double[] dataPoint = new double[numOfInputUnit];
                double[] target = new double[numOfOutputUnit];
                //set value for input units
                for (int p = 0; p < numOfInputUnit; p++) {
                    dataPoint[p] = trainingSet[p + l];
                }
                //set value for target unit  for one step forecast or multi step forecast
                for (int p = 0; p < numOfOutputUnit; p++) {
                    if ((numOfInputUnit + p + l) < trainingSet.length) {
                        target[p] = trainingSet[numOfInputUnit + p + l];
                    } else {
                        break;
                    }
                }

                ffnn.gradientDescent(dataPoint, target, eta, alpha);
                setWeightsV(ffnn.getV());
                setWeightsW(ffnn.getW());
            }

            SE = TestingNN(testingSet, numOfInputUnit, numOfHiddenUnit, numOfOutputUnit, eta, alpha);

            RMSEtemp[epoch] = calculateForecastError(SE);
            epoch += 1;
            if(epoch>maxEpoch){
                stopCondition= true;
```

```
            }

        }

        double[] RMSE = new double[epoch];
        for(int i=0; i<epoch; i++){
            RMSE[i] = RMSEtemp[i];
        }
        //System.out.println("rmse testing" +RMSE[epoch-1]);
        return RMSE;
    }

    public double[] TestingNN(double[] testingSet, int numOfInputUnit, int numOfHiddenUnit, int
numOfOutputUnit, double eta, double alpha) {
        /*
         * Testing one-step forecast or multi-step forecast
         */
        FFNN ffnn = new FFNN(numOfInputUnit, numOfHiddenUnit, numOfOutputUnit, eta, alpha, getWeightsV(),
getWeightsW());
        ffnn.setV(getWeightsV());
        ffnn.setW(getWeightsW());

        double[][] outputSet = new double[testingSet.length][numOfOutputUnit];
        double[][] targetSet = new double[testingSet.length][numOfOutputUnit];
        double[] output = new double[numOfOutputUnit];
        double[] SE = new double[(testingSet.length - numOfOutputUnit - numOfInputUnit)+1];
        for (int l = 0; l < (testingSet.length - numOfOutputUnit - numOfInputUnit +1); l++) {
            double[] dataPoint = new double[numOfInputUnit];
            double[] target = new double[numOfOutputUnit];
            //set value for input units
            for (int p = 0; p < numOfInputUnit; p++) {
                dataPoint[p] = testingSet[p + l];
            }
            output = ffnn.feedforward(dataPoint);
            for (int p = 0; p < numOfOutputUnit; p++) {
                outputSet[l][p] = output[p];
            }
            //set testing set
            for (int p = 0; p < numOfOutputUnit; p++) {
                if ((numOfInputUnit + p + l) < testingSet.length) {
                    targetSet[l][p] = testingSet[numOfInputUnit + p + l];
                    target[p] = targetSet[l][p];
                } else {
                    break;
                }
            }
            output = ffnn.feedforward(dataPoint);
            double[] outputD = denormalizeData(output);
            double[] targetD = denormalizeData(target);
            SE[l] = ffnn.calculateSE(outputD, targetD);
        }
        return SE;
    }

    public Vector generalizationNN(double[] validationSet, int numOfInputUnit, int numOfHiddenUnit, int
numOfOutputUnit, double eta, double alpha) {
        /*
         * Testing one-step forecast or multi-step forecast
         */

        FFNN ffnn = new FFNN(numOfInputUnit, numOfHiddenUnit, numOfOutputUnit, eta, alpha, getWeightsV(),
getWeightsW());
        ffnn.setV(getWeightsV());
        ffnn.setW(getWeightsW());

        double[][] outputSet = new double[validationSet.length][numOfOutputUnit];
        double[][] targetSet = new double[validationSet.length][numOfOutputUnit];
```

```
    double[] output = new double[numOfOutputUnit];
    double[] SE = new double[(validationSet.length - numOfOutputUnit - numOfInputUnit)+1];
    double[] absE = new double[(validationSet.length - numOfOutputUnit - numOfInputUnit)+1];
    double[] absPE = new double[(validationSet.length - numOfOutputUnit - numOfInputUnit)+1];
    for (int l = 0; l < (validationSet.length - numOfOutputUnit - numOfInputUnit + 1); l++) {
        double[] dataPoint = new double[numOfInputUnit];
        double[] target = new double[numOfOutputUnit];
        //set value for input units
        for (int p = 0; p < numOfInputUnit; p++) {
            dataPoint[p] = validationSet[p + l];
        }
        output = ffnn.feedforward(dataPoint);
        for (int p = 0; p < numOfOutputUnit; p++) {
            outputSet[l][p] = output[p];
        }
        //set testing set
        for (int p = 0; p < numOfOutputUnit; p++) {
            if ((numOfInputUnit + p + l) < validationSet.length) {
                targetSet[l][p] = validationSet[numOfInputUnit + p + l];
                target[p] = targetSet[l][p];
            } else {
                break;
            }
        }
        output = ffnn.feedforward(dataPoint);
        double[] outputD = denormalizeData(output);
        double[] targetD = denormalizeData(target);
        //System.out.println("output "+outputD[0]+" target "+targetD[0]);
        SE[l] = ffnn.calculateSE(outputD, targetD);
        absE[l] = ffnn.calculateAbsoluteError(outputD, targetD);
        absPE[l] = ffnn.calculateAbsolutePercentageError(outputD, targetD);
    }
    Vector forecastError = new Vector();
    forecastError.add(SE);
    forecastError.add(absE);
    forecastError.add(absPE);
    return forecastError;
}

public double calculateForecastError(double[] squaredError) {
    double SSE = 0.0;
    for (int i = 0; i < squaredError.length; i++) {
        SSE += squaredError[i];
    }
    double MSE = SSE / squaredError.length;
    double RMSE = Math.sqrt(MSE);
    //System.out.println("rmse length" +squaredError.length);
    //System.out.println("rmse " + RMSE);
    return RMSE;
}

public double calculateMAE (double[] absError) {
    double MAE = 0.0;
    for (int i = 0; i < absError.length; i++) {
        MAE += absError[i];
    }
    MAE = MAE/absError.length;
    //System.out.println("rmse length" +squaredError.length);
    //System.out.println("rmse " + RMSE);
    return MAE;
}

public double calculateMAPE (double[] absPError) {
    double MAPE = 0.0;
    for (int i = 0; i < absPError.length; i++) {
        MAPE += absPError[i];
    }
```

```
    MAPE = MAPE/absPError.length;
    //System.out.println("rmse length" +squaredError.length);
    //System.out.println("rmse " + RMSE);
    return MAPE;
}

public void setMinMax(double[] dataSet) {
    RCaller caller = new RCaller();
    caller.setRscriptExecutable("/usr/bin/Rscript");

    RCode code = new RCode();
    code.clear();

    /*get maximum and minimum value from data set*/
    code.addDoubleArray("dataSet", dataSet);
    code.addRCode("maxVal <- max(dataSet)");
    code.addRCode("minVal <- min(dataSet)");
    code.addRCode("results <-list(max = maxVal, min=minVal)");
    caller.setRCode(code);
    caller.runAndReturnResult("results");
    double[] max = caller.getParser().getAsDoubleArray("max");
    double[] min = caller.getParser().getAsDoubleArray("min");
    setMaxValue(max[0]);
    setMinValue(min[0]);
}

public double[] normalizeData(double[] dataSet) {
    /*
     * Normalize Data Set between 0.2 and 0.8
     */
    double[] normalizedData = new double[dataSet.length];
    double maxD = getMaxValue();
    double minD = getMinValue();
    /*Normalized Data*/
    for (int i = 0; i < dataSet.length; i++) {
        normalizedData[i] = (0.8 - 0.2) * ((dataSet[i] - minD) / (maxD - minD)) + 0.2;
    }
    return normalizedData;
}

public double[] denormalizeData(double[] normalizedData) {
    /*
     * get original value of data set
     */
    double[] originalData = new double[normalizedData.length];
    double maxD = getMaxValue();
    double minD = getMinValue();
    /*Denormalized Data*/
    for (int i = 0; i < normalizedData.length; i++) {

        originalData[i] = (((normalizedData[i] - 0.2) / (0.8 - 0.2)) * (maxD - minD)) + minD;
    }
    return originalData;
}

public double[] normalizeDataUsingSigmoid(double[] dataSet) {
    /*
     * Normalize Data Set between 0.2 and 0.8
     */
    double[] normalizedData = new double[dataSet.length];

    /*Normalized Data*/
    for (int i = 0; i < dataSet.length; i++) {
        normalizedData[i] = 1 / (1 + Math.exp(-1 * dataSet[i]));
    }
    return normalizedData;
}
```

```
public double[] denormalizeDataInverseSigmoid(double[] normalizedData) {
    /*
     * get original value of data set
     */
    double[] originalData = new double[normalizedData.length];
    double maxD = getMaxValue();
    double minD = getMinValue();
    /*Denormalized Data*/
    for (int i = 0; i < normalizedData.length; i++) {
        //originalData[i] = (inverseSigmoid((normalizedData[i] - 0.2)/(0.8-0.2))*(maxD-minD)) + minD;
        originalData[i] = inverseSigmoid(normalizedData[i]);
    }
    return originalData;
}

/**
 * @return the weightsV
 */
public double inverseSigmoid(double x) {
    double inverse = Math.log(x) - Math.log(1 - x);
    return inverse;
}

public double[][] getWeightsV() {
    return weightsV;
}

/**
 * @param weightsV the weightsV to set
 */
public void setWeightsV(double[][] weightsV) {
    this.weightsV = weightsV;
}

/**
 * @return the weightsW
 */
public double[][] getWeightsW() {
    return weightsW;
}

/**
 * @param weightsW the weightsW to set
 */
public void setWeightsW(double[][] weightsW) {
    this.weightsW = weightsW;
}

/**
 * @return the minValue
 */
public double getMinValue() {
    return minValue;
}

/**
 * @param minValue the minValue to set
 */
public void setMinValue(double minValue) {
    this.minValue = minValue;
}

/**
 * @return the maxValue
 */
public double getMaxValue() {
```

```
        return maxValue;
    }

    /**
     * @param maxValue the maxValue to set
     */
    public void setMaxValue(double maxValue) {
        this.maxValue = maxValue;
    }
}
```

TimeSeriesRNN.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package timeseriesforecast;

import java.util.Vector;
import rcaller.RCaller;
import rcaller.RCode;

/**
 *
 * @author Ega
 */
public class TimeSeriesRNN {

    private double[][] weightsV;
    private double[][] weightsW;
    private double [][] weightsU;
    private double minValue;
    private double maxValue;

    public TimeSeriesRNN(double [][] V, double [][] W, double [][] U){
        this.weightsV = V;
        this.weightsW = W;
        this.weightsU = U;
    }

    public double[] TrainingNN(int choiceW, double[] trainingSet, double[] testingSet,int numOfInputUnit, int
numOfHiddenUnit, int numOfOutputUnit, double eta, double alpha, int maxEpoch, double maxError) {
        /*
         * Learning in one-step forecast or multi-step forecast
         */
        RNN rnn = new RNN(numOfInputUnit, numOfHiddenUnit, numOfOutputUnit, eta, alpha,
getWeightsV(),getWeightsW(), getWeightsU());
        rnn.initializeRandomWeight(choiceW);
        rnn.initializeInputFromHiddenNeuronRNN(numOfHiddenUnit);
        boolean stopCondition = false;
        double forecastError = 9999999.99999;
        int epoch = 0;
        double[] SE = new double[trainingSet.length - numOfOutputUnit - numOfInputUnit +1];

        double[] RMSEtemp = new double[maxEpoch + 1];
        double[] output = new double[numOfOutputUnit];

        while (stopCondition != true) {

            for (int l = 0; l < (trainingSet.length - numOfOutputUnit - numOfInputUnit +1); l++) {
                double[] dataPoint = new double[numOfInputUnit];
                double[] target = new double[numOfOutputUnit];
                //set value for input units
```

```
                for (int p = 0; p < numOfInputUnit; p++) {
                    dataPoint[p] = trainingSet[p + l];
                }
                //set value for target unit  for one step forecast or multi step forecast
                for (int p = 0; p < numOfOutputUnit; p++) {
                    if ((numOfInputUnit + p + l) < trainingSet.length) {
                        target[p] = trainingSet[numOfInputUnit + p + l];
                    } else {
                        break;
                    }
                }

                rnn.gradientDescent(dataPoint, target, eta, alpha);
                setWeightsV(rnn.getV());
                setWeightsW(rnn.getW());
                setWeightsU(rnn.getU());
            }

            SE = TestingNN(testingSet, numOfInputUnit, numOfHiddenUnit, numOfOutputUnit, eta, alpha);

            RMSEtemp[epoch] = calculateForecastError(SE);
            epoch += 1;
            if(epoch>maxEpoch){
                stopCondition= true;
            }

        }
        //System.out.println("epoch fin" + epoch);
        //System.out.println("min value" + getMinValue());
        //System.out.println("max value" + getMaxValue());

        //System.out.println("testing set size "+testingSet.length);
        double[] RMSE = new double[epoch];
        for(int i=0; i<epoch; i++){
            RMSE[i] = RMSEtemp[i];
        }
        //System.out.println("rmse testing" +RMSE[epoch-1]);
        return RMSE;
    }

    public double[] TestingNN(double[] testingSet, int numOfInputUnit, int numOfHiddenUnit, int
numOfOutputUnit, double eta, double alpha) {
        /*
         * Testing one-step forecast or multi-step forecast
         */
        RNN rnn = new RNN(numOfInputUnit, numOfHiddenUnit, numOfOutputUnit, eta, alpha,
getWeightsV(),getWeightsW(), getWeightsU());
        rnn.setV(getWeightsV());
        rnn.setW(getWeightsW());
        rnn.setU(getWeightsU());

        double[][] outputSet = new double[testingSet.length][numOfOutputUnit];
        double[][] targetSet = new double[testingSet.length][numOfOutputUnit];
        double[] output = new double[numOfOutputUnit];
        double[] SE = new double[(testingSet.length - numOfOutputUnit - numOfInputUnit)+1];
        for (int l = 0; l < (testingSet.length - numOfOutputUnit - numOfInputUnit+1); l++) {
            double[] dataPoint = new double[numOfInputUnit];
            double[] target = new double[numOfOutputUnit];
            //set value for input units
            for (int p = 0; p < numOfInputUnit; p++) {
                dataPoint[p] = testingSet[p + l];
            }
            output = rnn.forward(dataPoint);
            for (int p = 0; p < numOfOutputUnit; p++) {
                outputSet[l][p] = output[p];
            }
            //set testing set
```

```
            for (int p = 0; p < numOfOutputUnit; p++) {
                if ((numOfInputUnit + p + l) < testingSet.length) {
                    targetSet[l][p] = testingSet[numOfInputUnit + p + l];
                    target[p] = targetSet[l][p];
                } else {
                    break;
                }
            }

            double[] outputD = denormalizeData(output);
            double[] targetD = denormalizeData(target);
            SE[l] = rnn.calculateSE(outputD, targetD);
        }
        return SE;
    }

    public Vector generalizationNN(double[] validationSet, int numOfInputUnit, int numOfHiddenUnit, int
numOfOutputUnit, double eta, double alpha) {
        /*
         * Testing one-step forecast or multi-step forecast
         */
        RNN rnn = new RNN(numOfInputUnit, numOfHiddenUnit, numOfOutputUnit, eta, alpha,
getWeightsV(),getWeightsW(), getWeightsU());
        rnn.setV(getWeightsV());
        rnn.setW(getWeightsW());
        rnn.setU(getWeightsU());

        double[][] outputSet = new double[validationSet.length][numOfOutputUnit];
        double[][] targetSet = new double[validationSet.length][numOfOutputUnit];
        double[] output = new double[numOfOutputUnit];
        double[] SE = new double[(validationSet.length - numOfOutputUnit - numOfInputUnit)+1];
        double[] absE = new double[(validationSet.length - numOfOutputUnit - numOfInputUnit)+1];
        double[] absPE = new double[(validationSet.length - numOfOutputUnit - numOfInputUnit)+1];
        for (int l = 0; l < (validationSet.length - numOfOutputUnit - numOfInputUnit+1); l++) {
            double[] dataPoint = new double[numOfInputUnit];
            double[] target = new double[numOfOutputUnit];
            //set value for input units
            for (int p = 0; p < numOfInputUnit; p++) {
                dataPoint[p] = validationSet[p + l];
            }
            output = rnn.forward(dataPoint);
            for (int p = 0; p < numOfOutputUnit; p++) {
                outputSet[l][p] = output[p];
            }
            //set testing set
            for (int p = 0; p < numOfOutputUnit; p++) {
                if ((numOfInputUnit + p + l) < validationSet.length) {
                    targetSet[l][p] = validationSet[numOfInputUnit + p + l];
                    target[p] = targetSet[l][p];
                } else {
                    break;
                }
            }
            output = rnn.forward(dataPoint);
            double[] outputD = denormalizeData(output);
            double[] targetD = denormalizeData(target);
            SE[l] = rnn.calculateSE(outputD, targetD);
            absE[l] = rnn.calculateAbsoluteError(outputD, targetD);
            absPE[l] = rnn.calculateAbsolutePercentageError(outputD, targetD);
        }
        Vector forecastError = new Vector();
        forecastError.add(SE);
        forecastError.add(absE);
        forecastError.add(absPE);
        return forecastError;
    }
```

```
public double calculateForecastError(double[] squaredError) {
    double SSE = 0.0;
    for (int i = 0; i < squaredError.length; i++) {
        SSE += squaredError[i];
    }
    double MSE = SSE / squaredError.length;
    double RMSE = Math.sqrt(MSE);
    //System.out.println("rmse " + RMSE);
    return RMSE;
}

public double calculateMAE (double[] absError) {
    double MAE = 0.0;
    for (int i = 0; i < absError.length; i++) {
        MAE += absError[i];
    }
    MAE = MAE/absError.length;
    //System.out.println("rmse length" +squaredError.length);
    //System.out.println("rmse " + RMSE);
    return MAE;
}

public double calculateMAPE (double[] absPError) {
    double MAPE = 0.0;
    for (int i = 0; i < absPError.length; i++) {
        MAPE += absPError[i];
    }
    MAPE = MAPE/absPError.length;
    //System.out.println("rmse length" +squaredError.length);
    //System.out.println("rmse " + RMSE);
    return MAPE;
}

public void setMinMax(double[] dataSet) {
    RCaller caller = new RCaller();
    caller.setRscriptExecutable("/usr/bin/Rscript");

    RCode code = new RCode();
    code.clear();

    /*get maximum and minimum value from data set*/
    code.addDoubleArray("dataSet", dataSet);
    code.addRCode("maxVal <- max(dataSet)");
    code.addRCode("minVal <- min(dataSet)");
    code.addRCode("results <-list(max = maxVal, min=minVal)");
    caller.setRCode(code);
    caller.runAndReturnResult("results");
    double[] max = caller.getParser().getAsDoubleArray("max");
    double[] min = caller.getParser().getAsDoubleArray("min");
    setMaxValue(max[0]);
    setMinValue(min[0]);
}

public double[] normalizeData(double[] dataSet) {
    /*
     * Normalize Data Set between 0.2 and 0.8
     */
    double[] normalizedData = new double[dataSet.length];
    double maxD = getMaxValue();
    double minD = getMinValue();
    /*Normalized Data*/
    for (int i = 0; i < dataSet.length; i++) {
        normalizedData[i] = (0.8 - 0.2) * ((dataSet[i] - minD) / (maxD - minD)) + 0.2;
    }
    return normalizedData;
}
```

```
public double[] denormalizeData(double[] normalizedData) {
    /*
     * get original value of data set
     */
    double[] originalData = new double[normalizedData.length];
    double maxD = getMaxValue();
    double minD = getMinValue();
    /*Denormalized Data*/
    for (int i = 0; i < normalizedData.length; i++) {

        originalData[i] = (((normalizedData[i] - 0.2) / (0.8 - 0.2)) * (maxD - minD)) + minD;
    }
    return originalData;
}

public double[] normalizeDataUsingSigmoid(double[] dataSet) {
    /*
     * Normalize Data Set between 0.2 and 0.8
     */
    double[] normalizedData = new double[dataSet.length];

    /*Normalized Data*/
    for (int i = 0; i < dataSet.length; i++) {
        normalizedData[i] = 1 / (1 + Math.exp(-1 * dataSet[i]));
    }
    return normalizedData;
}

public double[] denormalizeDataInverseSigmoid(double[] normalizedData) {
    /*
     * get original value of data set
     */
    double[] originalData = new double[normalizedData.length];
    double maxD = getMaxValue();
    double minD = getMinValue();
    /*Denormalized Data*/
    for (int i = 0; i < normalizedData.length; i++) {
        //originalData[i] = (inverseSigmoid((normalizedData[i] - 0.2)/(0.8-0.2))*(maxD-minD)) + minD;
        originalData[i] = inverseSigmoid(normalizedData[i]);
    }
    return originalData;
}

/**
 * @return the weightsV
 */
public double inverseSigmoid(double x) {
    double inverse = Math.log(x) - Math.log(1 - x);
    return inverse;
}

public double[][] getWeightsV() {
    return weightsV;
}

/**
 * @param weightsV the weightsV to set
 */
public void setWeightsV(double[][] weightsV) {
    this.weightsV = weightsV;
}

/**
 * @return the weightsW
 */
public double[][] getWeightsW() {
    return weightsW;
```

```
    }

    /**
     * @param weightsW the weightsW to set
     */
    public void setWeightsW(double[][] weightsW) {
        this.weightsW = weightsW;
    }

    /**
     * @return the minValue
     */
    public double getMinValue() {
        return minValue;
    }

    /**
     * @param minValue the minValue to set
     */
    public void setMinValue(double minValue) {
        this.minValue = minValue;
    }

    /**
     * @return the maxValue
     */
    public double getMaxValue() {
        return maxValue;
    }

    /**
     * @param maxValue the maxValue to set
     */
    public void setMaxValue(double maxValue) {
        this.maxValue = maxValue;
    }

    /**
     * @return the weightsU
     */
    public double[][] getWeightsU() {
        return weightsU;
    }

    /**
     * @param weightsU the weightsU to set
     */
    public void setWeightsU(double[][] weightsU) {
        this.weightsU = weightsU;
    }
}
```

TimeSeriesHybridFFNN.java

```
package timeseriesforecast;

import java.util.Vector;
import rcaller.RCaller;
import rcaller.RCode;

/**
 *
 * @author Ega
 */
public class TimeSeriesHybridFFNN {
```

```
    private double[][] weightsV;
    private double[][] weightsW;
    private double minValue;
    private double maxValue;

    public TimeSeriesHybridFFNN(double [][] V, double [][]W){
        this.weightsV = V;
        this.weightsW = W;
    }

    public double[] TrainingNN(int choiceW, double[] trainingSet, double[] testingSet,int numOfInputUnit, int
numOfHiddenUnit, int numOfOutputUnit, double eta, double alpha, int maxEpoch, double maxError) {
        /*
         * Learning in one-step forecast or multi-step forecast
         */
        FFNN ffnn = new FFNN(numOfInputUnit, numOfHiddenUnit, numOfOutputUnit, eta, alpha, getWeightsV(),
getWeightsW());
        ffnn.initializeRandomWeight(choiceW);
        //System.out.println("");
        //ffnn.showWeights(ffnn.getV());
        //System.out.println("");
        //ffnn.showWeights(ffnn.getW());
        //System.out.println("");
        boolean stopCondition = false;
        double forecastError = 9999999.99999;
        int epoch = 0;

        double[] SE = new double[trainingSet.length - numOfOutputUnit - numOfInputUnit +1];

        double[] RMSEtemp = new double[maxEpoch + 1];
        double[] output = new double[numOfOutputUnit];

        while (stopCondition != true) {

            for (int l = 0; l < (trainingSet.length - numOfOutputUnit - numOfInputUnit +1); l++) {
                double[] dataPoint = new double[numOfInputUnit];
                double[] target = new double[numOfOutputUnit];
                //set value for input units
                for (int p = 0; p < numOfInputUnit; p++) {
                    dataPoint[p] = trainingSet[p + l];
                }
                //set value for target unit  for one step forecast or multi step forecast
                for (int p = 0; p < numOfOutputUnit; p++) {
                    if ((numOfInputUnit + p + l) < trainingSet.length) {
                        target[p] = trainingSet[numOfInputUnit + p + l];
                    } else {
                        break;
                    }
                }

                ffnn.gradientDescent(dataPoint, target, eta, alpha);
                setWeightsV(ffnn.getV());
                setWeightsW(ffnn.getW());
            }

            SE = TestingNN(testingSet, numOfInputUnit, numOfHiddenUnit, numOfOutputUnit, eta, alpha);

            RMSEtemp[epoch] = calculateForecastError(SE);
            epoch += 1;
            if(epoch>maxEpoch){
                stopCondition= true;
            }

        }
        double[] RMSE = new double[epoch];
        for(int i=0; i<epoch; i++){
            RMSE[i] = RMSEtemp[i];
```

```
        }
        //System.out.println("rmse testing" +RMSE[epoch-1]);
        return RMSE;
    }

    public double [] TestingNN(double[] testingSet, int numOfInputUnit, int numOfHiddenUnit, int
numOfOutputUnit, double eta, double alpha) {
        /*
         * Testing one-step forecast or multi-step forecast
         */

        FFNN ffnn = new FFNN(numOfInputUnit, numOfHiddenUnit, numOfOutputUnit, eta, alpha, getWeightsV(),
getWeightsW());
        ffnn.setV(getWeightsV());
        ffnn.setW(getWeightsW());
        int L = (testingSet.length - numOfOutputUnit - numOfInputUnit +1);
        double[][] outputSetD = new double[L][numOfOutputUnit];
        double[][] targetSetD = new double[L][numOfOutputUnit];
        double[][] targetSet = new double[L][numOfOutputUnit];
        double[] output = new double[numOfOutputUnit];
        double[] SE = new double[L];
        double [][] errorForInput = new double[L][numOfOutputUnit];
        for (int l = 0; l < L; l++) {
            double[] dataPoint = new double[numOfInputUnit];
            double[] target = new double[numOfOutputUnit];
            //set value for input units
            for (int p = 0; p < numOfInputUnit; p++) {
                dataPoint[p] = testingSet[p + l];
            }
            output = ffnn.feedforward(dataPoint);

            //set testing set
            for (int p = 0; p < numOfOutputUnit; p++) {
                if ((numOfInputUnit + p + l) < testingSet.length) {
                    targetSet[l][p] = testingSet[numOfInputUnit + p + l];
                    target[p] = targetSet[l][p];
                } else {
                    break;
                }
            }

            output = ffnn.feedforward(dataPoint);
            double[] outputD = denormalizeData(output);
            double[] targetD = denormalizeData(target);
            for (int p = 0; p < numOfOutputUnit; p++) {
                outputSetD[l][p] = outputD[p];
                targetSetD[l][p] = targetD[p];

            }

            for (int p = 0; p < numOfOutputUnit; p++) {
                outputSetD[l][p] = outputD[p];
                targetSetD[l][p] = targetD[p];
                errorForInput[l][p] =  outputSetD[l][p] - targetSetD[l][p];
                //System.out.println("output-t-"+outputSetD[l][p]+" target-t-"+targetSetD[l][p]);
                //System.out.println("error for input"+errorForInput[l][p]);
            }

        }


        double [][] linearOutput = new double[outputSetD.length][outputSetD[0].length];



         for(int i = 0; i<errorForInput[0].length; i++){
            ARIMA arima = new ARIMA();
```

```
            double [] arimaOutput = new double [linearOutput.length];
            double [] arimaInput = new double [linearOutput.length];


            for(int j = 0; j< linearOutput.length; j++){
              arimaInput[j] = errorForInput[j][i];
            }
            arimaOutput = arima.getPredictionValueOnInputError(arimaInput);

            //System.out.println("Arima output length" + arimaOutput.length);

            for(int j = 0; j< linearOutput.length; j++){
               linearOutput[j][i] = arimaOutput[j];
              //System.out.println("arima Output "+arimaOutput[j]);
            }
        }

        // combine output of NN and ARIMA
        double[][] hybridOutput = new double[testingSet.length][numOfOutputUnit];
        for(int j = 0; j< outputSetD.length; j++){
            for(int i =0; i<outputSetD[0].length; i++){
                if(errorForInput[j][i] < 0 && linearOutput[j][i] <0){
                    hybridOutput[j][i] = outputSetD[j][i] - linearOutput[j][i];
                }
                else
                    hybridOutput[j][i] = outputSetD[j][i] + linearOutput[j][i];
            }
            SE[j] = ffnn.calculateSE(hybridOutput[j], targetSetD[j]);
        }


        return SE;
    }

    public Vector generalizationNN(double[] validationSet, int numOfInputUnit, int numOfHiddenUnit, int
numOfOutputUnit, double eta, double alpha) {
        /*
         * one-step forecast or multi-step forecast
         */
        FFNN ffnn = new FFNN(numOfInputUnit, numOfHiddenUnit, numOfOutputUnit, eta, alpha, getWeightsV(),
getWeightsW());
        ffnn.setV(getWeightsV());
        ffnn.setW(getWeightsW());

        int L = (validationSet.length - numOfOutputUnit - numOfInputUnit +1);
        double[][] outputSetD = new double[L][numOfOutputUnit];
        double[][] targetSetD = new double[L][numOfOutputUnit];
        double[][] targetSet = new double[L][numOfOutputUnit];
        double[] output = new double[numOfOutputUnit];
        double[] SE = new double[L];
        double[] absE = new double[(validationSet.length - numOfOutputUnit - numOfInputUnit)+1];
        double[] absPE = new double[(validationSet.length - numOfOutputUnit - numOfInputUnit)+1];
        double [][] errorForInput = new double[L][numOfOutputUnit];
        for (int l = 0; l < L; l++) {
            double[] dataPoint = new double[numOfInputUnit];
            double[] target = new double[numOfOutputUnit];
            //set value for input units
            for (int p = 0; p < numOfInputUnit; p++) {
                dataPoint[p] = validationSet[p + l];
            }
            output = ffnn.feedforward(dataPoint);

            //set testing set
            for (int p = 0; p < numOfOutputUnit; p++) {
                if ((numOfInputUnit + p + l) < validationSet.length) {
                    targetSet[l][p] = validationSet[numOfInputUnit + p + l];
                    target[p] = targetSet[l][p];
```

```
            } else {
                break;
            }
        }

        output = ffnn.feedforward(dataPoint);
        double[] outputD = denormalizeData(output);
        double[] targetD = denormalizeData(target);
        for (int p = 0; p < numOfOutputUnit; p++) {
            outputSetD[l][p] = outputD[p];
            targetSetD[l][p] = targetD[p];
            errorForInput[l][p] = targetSetD[l][p]- outputSetD[l][p];
            //System.out.println("output-t-"+outputSetD[l][p]+" target-t-"+targetSetD[l][p]);
            //System.out.println("error for input"+errorForInput[l][p]);

        }

    }


    double [][] linearOutput = new double[outputSetD.length][outputSetD[0].length];

    //calculate forecast using ARIMA
    //System.out.println("errorForInput[0].length "+errorForInput[0].length);
    //System.out.println("linearOutput.length "+linearOutput.length);
    for(int i = 0; i<errorForInput[0].length; i++){
        ARIMA arima = new ARIMA();
        double [] arimaOutput = new double [linearOutput.length];
        double [] arimaInput = new double [linearOutput.length];


        for(int j = 0; j< linearOutput.length; j++){
          arimaInput[j] = errorForInput[j][i];
        }
        arimaOutput = arima.getPredictionValueOnInputError(arimaInput);

        //System.out.println("Arima output length" + arimaOutput.length);

        for(int j = 0; j< linearOutput.length; j++){
            linearOutput[j][i] = arimaOutput[j];

          //System.out.println("arima Output "+arimaOutput[j]);
        }
    }

    // combine output of NN and ARIMA
    double[][] hybridOutput = new double[validationSet.length][numOfOutputUnit];
    for(int j = 0; j< outputSetD.length; j++){
        for(int i =0; i<outputSetD[0].length; i++){
            if(errorForInput[j][i] < 0){
                hybridOutput[j][i] = outputSetD[j][i] - linearOutput[j][i];
            }
            else{
                hybridOutput[j][i] = outputSetD[j][i] + linearOutput[j][i];
            }
        }
        SE[j] = ffnn.calculateSE(hybridOutput[j], targetSetD[j]);
        absE[j] = ffnn.calculateAbsoluteError(hybridOutput[j], targetSetD[j]);
        absPE[j] = ffnn.calculateAbsolutePercentageError(hybridOutput[j], targetSetD[j]);
    }
    Vector forecastError = new Vector();
    forecastError.add(SE);
    forecastError.add(absE);
    forecastError.add(absPE);
    return forecastError;
}
```

```
public double calculateSEHybrid(double[] output, double[] target) {
    double SE = 0;
    int K = output.length;
    for (int k = 0; k < K; k++) {
        double e = target[k] - output[k];
        SE += e * e;

    }
    SE = SE/K;
    return SE;
}

public double calculateForecastError(double[] squaredError) {
    double SSE = 0.0;
    for (int i = 0; i < squaredError.length; i++) {
        SSE += squaredError[i];
    }
    double MSE = SSE / squaredError.length;
    double RMSE = Math.sqrt(MSE);
    //System.out.println("rmse " + RMSE);
    return RMSE;
}

public double calculateMAE (double[] absError) {
    double MAE = 0.0;
    for (int i = 0; i < absError.length; i++) {
        MAE += absError[i];
    }
    MAE = MAE/absError.length;
    //System.out.println("rmse length" +squaredError.length);
    //System.out.println("rmse " + RMSE);
    return MAE;
}

public double calculateMAPE (double[] absPError) {
    double MAPE = 0.0;
    for (int i = 0; i < absPError.length; i++) {
        MAPE += absPError[i];
    }
    MAPE = MAPE/absPError.length;
    //System.out.println("rmse length" +squaredError.length);
    //System.out.println("rmse " + RMSE);
    return MAPE;
}

public void setMinMax(double[] dataSet) {
    RCaller caller = new RCaller();
    caller.setRscriptExecutable("/usr/bin/Rscript");

    RCode code = new RCode();
    code.clear();

    /*get maximum and minimum value from data set*/
    code.addDoubleArray("dataSet", dataSet);
    code.addRCode("maxVal <- max(dataSet)");
    code.addRCode("minVal <- min(dataSet)");
    code.addRCode("results <-list(max = maxVal, min=minVal)");
    caller.setRCode(code);
    caller.runAndReturnResult("results");
    double[] max = caller.getParser().getAsDoubleArray("max");
    double[] min = caller.getParser().getAsDoubleArray("min");
    setMaxValue(max[0]);
    setMinValue(min[0]);
}

public double[] normalizeData(double[] dataSet) {
    /*
```

```
     * Normalize Data Set between 0.2 and 0.8
     */
    double[] normalizedData = new double[dataSet.length];
    double maxD = getMaxValue();
    double minD = getMinValue();
    /*Normalized Data*/
    for (int i = 0; i < dataSet.length; i++) {
        normalizedData[i] = (0.8 - 0.2) * ((dataSet[i] - minD) / (maxD - minD)) + 0.2;
    }
    return normalizedData;
}

public double[] denormalizeData(double[] normalizedData) {
    /*
     * get original value of data set
     */
    double[] originalData = new double[normalizedData.length];
    double maxD = getMaxValue();
    double minD = getMinValue();
    /*Denormalized Data*/
    for (int i = 0; i < normalizedData.length; i++) {

        originalData[i] = (((normalizedData[i] - 0.2) / (0.8 - 0.2)) * (maxD - minD)) + minD;
    }
    return originalData;
}

public double[] normalizeDataUsingSigmoid(double[] dataSet) {
    /*
     * Normalize Data Set between 0.2 and 0.8
     */
    double[] normalizedData = new double[dataSet.length];

    /*Normalized Data*/
    for (int i = 0; i < dataSet.length; i++) {
        normalizedData[i] = 1 / (1 + Math.exp(-1 * dataSet[i]));
    }
    return normalizedData;
}

public double[] denormalizeDataInverseSigmoid(double[] normalizedData) {
    /*
     * get original value of data set
     */
    double[] originalData = new double[normalizedData.length];
    double maxD = getMaxValue();
    double minD = getMinValue();
    /*Denormalized Data*/
    for (int i = 0; i < normalizedData.length; i++) {
        //originalData[i] = (inverseSigmoid((normalizedData[i] - 0.2)/(0.8-0.2))*(maxD-minD)) + minD;
        originalData[i] = inverseSigmoid(normalizedData[i]);
    }
    return originalData;
}

/**
 * @return the weightsV
 */
public double inverseSigmoid(double x) {
    double inverse = Math.log(x) - Math.log(1 - x);
    return inverse;
}

public double[][] getWeightsV() {
    return weightsV;
}
```

```
    /**
     * @param weightsV the weightsV to set
     */
    public void setWeightsV(double[][] weightsV) {
        this.weightsV = weightsV;
    }

    /**
     * @return the weightsW
     */
    public double[][] getWeightsW() {
        return weightsW;
    }

    /**
     * @param weightsW the weightsW to set
     */
    public void setWeightsW(double[][] weightsW) {
        this.weightsW = weightsW;
    }

    /**
     * @return the minValue
     */
    public double getMinValue() {
        return minValue;
    }

    /**
     * @param minValue the minValue to set
     */
    public void setMinValue(double minValue) {
        this.minValue = minValue;
    }

    /**
     * @return the maxValue
     */
    public double getMaxValue() {
        return maxValue;
    }

    /**
     * @param maxValue the maxValue to set
     */
    public void setMaxValue(double maxValue) {
        this.maxValue = maxValue;
    }
}
```

TimeSeriesHybridRNN.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package timeseriesforecast;

import java.util.Vector;
import rcaller.RCaller;
import rcaller.RCode;

/**
 *
 * @author Ega
```

```
 */
public class TimeSeriesHybridRNN {

    private double[][] weightsV;
    private double[][] weightsW;
    private double[][] weightsU;
    private double minValue;
    private double maxValue;

    public TimeSeriesHybridRNN(double[][] V, double[][] W, double[][] U) {
        this.weightsV = V;
        this.weightsW = W;
        this.weightsU = U;
    }

    public double[] TrainingNN(int choiceW, double[] trainingSet, double[] testingSet, int numOfInputUnit, int
numOfHiddenUnit, int numOfOutputUnit, double eta, double alpha, int maxEpoch, double maxError) {
        /*
         * Learning in one-step forecast or multi-step forecast
         */
        RNN rnn = new RNN(numOfInputUnit, numOfHiddenUnit, numOfOutputUnit, eta, alpha, getWeightsV(),
getWeightsW(), getWeightsU());
        rnn.initializeRandomWeight(choiceW);
        rnn.initializeInputFromHiddenNeuronRNN(numOfHiddenUnit);
        boolean stopCondition = false;
        double forecastError = 9999999.99999;
        int epoch = 0;
        double[] SE = new double[trainingSet.length - numOfOutputUnit - numOfInputUnit + 1];

        double[] RMSEtemp = new double[maxEpoch + 1];
        double[] output = new double[numOfOutputUnit];

        while (stopCondition != true) {

            for (int l = 0; l < (trainingSet.length - numOfOutputUnit - numOfInputUnit); l++) {
                double[] dataPoint = new double[numOfInputUnit];
                double[] target = new double[numOfOutputUnit];
                //set value for input units
                for (int p = 0; p < numOfInputUnit; p++) {
                    dataPoint[p] = trainingSet[p + l];
                }
                //set value for target unit  for one step forecast or multi step forecast
                for (int p = 0; p < numOfOutputUnit; p++) {
                    if ((numOfInputUnit + p + l) < trainingSet.length) {
                        target[p] = trainingSet[numOfInputUnit + p + l];
                    } else {
                        break;
                    }
                }

                rnn.gradientDescent(dataPoint, target, eta, alpha);
                setWeightsV(rnn.getV());
                setWeightsW(rnn.getW());
                setWeightsU(rnn.getU());
            }

            SE = TestingNN(testingSet, numOfInputUnit, numOfHiddenUnit, numOfOutputUnit, eta, alpha);

            RMSEtemp[epoch] = calculateForecastError(SE);
            epoch += 1;
            if (epoch > maxEpoch) {
                stopCondition = true;
            }

        }
        //System.out.println("epoch fin" + epoch);
        //System.out.println("min value" + getMinValue());
```

```
        //System.out.println("max value" + getMaxValue());

        //System.out.println("testing set size "+testingSet.length);
        double[] RMSE = new double[epoch];
        for (int i = 0; i < epoch; i++) {
            RMSE[i] = RMSEtemp[i];
        }
        //System.out.println("rmse testing" +RMSE[epoch-1]);
        return RMSE;
    }

    public double[] TestingNN(double[] testingSet, int numOfInputUnit, int numOfHiddenUnit, int
numOfOutputUnit, double eta, double alpha) {
        /*
         * Testing one-step forecast or multi-step forecast
         */
        RNN rnn = new RNN(numOfInputUnit, numOfHiddenUnit, numOfOutputUnit, eta, alpha, getWeightsV(),
getWeightsW(), getWeightsU());
        rnn.setV(getWeightsV());
        rnn.setW(getWeightsW());
        rnn.setU(getWeightsU());

        int L = (testingSet.length - numOfOutputUnit - numOfInputUnit + 1);
        double[][] outputSetD = new double[L][numOfOutputUnit];
        double[][] targetSetD = new double[L][numOfOutputUnit];
        double[][] targetSet = new double[L][numOfOutputUnit];
        double[] output = new double[numOfOutputUnit];
        double[] SE = new double[L];
        double[][] errorForInput = new double[L][numOfOutputUnit];
        for (int l = 0; l < L; l++) {
            double[] dataPoint = new double[numOfInputUnit];
            double[] target = new double[numOfOutputUnit];
            //set value for input units
            for (int p = 0; p < numOfInputUnit; p++) {
                dataPoint[p] = testingSet[p + l];
            }
            output = rnn.forward(dataPoint);

            //set testing set
            for (int p = 0; p < numOfOutputUnit; p++) {
                if ((numOfInputUnit + p + l) < testingSet.length) {
                    targetSet[l][p] = testingSet[numOfInputUnit + p + l];
                    target[p] = targetSet[l][p];
                } else {
                    break;
                }
            }

            output = rnn.forward(dataPoint);
            double[] outputD = denormalizeData(output);
            double[] targetD = denormalizeData(target);
            for (int p = 0; p < numOfOutputUnit; p++) {
                outputSetD[l][p] = outputD[p];
                targetSetD[l][p] = targetD[p];

            }

            for (int p = 0; p < numOfOutputUnit; p++) {
                outputSetD[l][p] = outputD[p];
                targetSetD[l][p] = targetD[p];
                errorForInput[l][p] = outputSetD[l][p] - targetSetD[l][p];

            }

        }
```

```
        double[][] linearOutput = new double[outputSetD.length][outputSetD[0].length];


        for (int i = 0; i < errorForInput[0].length; i++) {
            ARIMA arima = new ARIMA();
            double[] arimaOutput = new double[linearOutput.length];
            double[] arimaInput = new double[linearOutput.length];


            for (int j = 0; j < linearOutput.length; j++) {
                arimaInput[j] = errorForInput[j][i];
            }
            arimaOutput = arima.getPredictionValueOnInputError(arimaInput);

            //System.out.println("Arima output length" + arimaOutput.length);

            for (int j = 0; j < linearOutput.length; j++) {
                linearOutput[j][i] = arimaOutput[j];
                //System.out.println("arima Output "+arimaOutput[j]);
            }
        }

        // combine output of NN and ARIMA
        double[][] hybridOutput = new double[testingSet.length][numOfOutputUnit];
        for (int j = 0; j < outputSetD.length; j++) {
            for (int i = 0; i < outputSetD[0].length; i++) {
                if (errorForInput[j][i] < 0 && linearOutput[j][i] < 0) {
                    hybridOutput[j][i] = outputSetD[j][i] - linearOutput[j][i];
                } else {
                    hybridOutput[j][i] = outputSetD[j][i] + linearOutput[j][i];
                }
            }
            SE[j] = rnn.calculateSE(hybridOutput[j], targetSetD[j]);
        }


        return SE;
    }

    public Vector generalizationNN(double[] validationSet, int numOfInputUnit, int numOfHiddenUnit, int
numOfOutputUnit, double eta, double alpha) {
        /*
         * one-step forecast or multi-step forecast
         */
        RNN rnn = new RNN(numOfInputUnit, numOfHiddenUnit, numOfOutputUnit, eta, alpha, getWeightsV(),
getWeightsW(), getWeightsU());
        rnn.setV(getWeightsV());
        rnn.setW(getWeightsW());
        rnn.setU(getWeightsU());

        int L = (validationSet.length - numOfOutputUnit - numOfInputUnit + 1);
        double[][] outputSetD = new double[L][numOfOutputUnit];
        double[][] targetSetD = new double[L][numOfOutputUnit];
        double[][] targetSet = new double[L][numOfOutputUnit];
        double[] output = new double[numOfOutputUnit];
        double[] SE = new double[L];
        double[] absE = new double[(validationSet.length - numOfOutputUnit - numOfInputUnit) + 1];
        double[] absPE = new double[(validationSet.length - numOfOutputUnit - numOfInputUnit) + 1];
        double[][] errorForInput = new double[L][numOfOutputUnit];
        for (int l = 0; l < L; l++) {
            double[] dataPoint = new double[numOfInputUnit];
            double[] target = new double[numOfOutputUnit];
            //set value for input units
            for (int p = 0; p < numOfInputUnit; p++) {
                dataPoint[p] = validationSet[p + l];
            }
```

```
        output = rnn.forward(dataPoint);

        //set testing set
        for (int p = 0; p < numOfOutputUnit; p++) {
            if ((numOfInputUnit + p + l) < validationSet.length) {
                targetSet[l][p] = validationSet[numOfInputUnit + p + l];
                target[p] = targetSet[l][p];
            } else {
                break;
            }
        }

        output = rnn.forward(dataPoint);
        double[] outputD = denormalizeData(output);
        double[] targetD = denormalizeData(target);
        for (int p = 0; p < numOfOutputUnit; p++) {
            outputSetD[l][p] = outputD[p];
            targetSetD[l][p] = targetD[p];

        }

        for (int p = 0; p < numOfOutputUnit; p++) {
            outputSetD[l][p] = outputD[p];
            targetSetD[l][p] = targetD[p];
            errorForInput[l][p] = outputSetD[l][p] - targetSetD[l][p];

        }

    }


    double[][] linearOutput = new double[outputSetD.length][outputSetD[0].length];

    for (int i = 0; i < errorForInput[0].length; i++) {
        ARIMA arima = new ARIMA();
        double[] arimaOutput = new double[linearOutput.length];
        double[] arimaInput = new double[linearOutput.length];


        for (int j = 0; j < linearOutput.length; j++) {
            arimaInput[j] = errorForInput[j][i];
        }
        arimaOutput = arima.getPredictionValueOnInputError(arimaInput);

        //System.out.println("Arima output length" + arimaOutput.length);

        for (int j = 0; j < linearOutput.length; j++) {
            linearOutput[j][i] = arimaOutput[j];
            //System.out.println("arima Output "+arimaOutput[j]);
        }
    }

    // combine output of NN and ARIMA
    double[][] hybridOutput = new double[validationSet.length][numOfOutputUnit];
    for (int j = 0; j < outputSetD.length; j++) {
        for (int i = 0; i < outputSetD[0].length; i++) {
            if (errorForInput[j][i] < 0 && linearOutput[j][i] < 0) {
                hybridOutput[j][i] = outputSetD[j][i] - linearOutput[j][i];
            } else {
                hybridOutput[j][i] = outputSetD[j][i] + linearOutput[j][i];
            }
        }
        SE[j] = rnn.calculateSE(hybridOutput[j], targetSetD[j]);
        absE[j] = rnn.calculateAbsoluteError(hybridOutput[j], targetSetD[j]);
        absPE[j] = rnn.calculateAbsolutePercentageError(hybridOutput[j], targetSetD[j]);
    }
    Vector forecastError = new Vector();
```

```java
        forecastError.add(SE);
        forecastError.add(absE);
        forecastError.add(absPE);
        return forecastError;
    }

    public double calculateForecastError(double[] squaredError) {
        double SSE = 0.0;
        for (int i = 0; i < squaredError.length; i++) {
            SSE += squaredError[i];
        }
        double MSE = SSE / squaredError.length;
        double RMSE = Math.sqrt(MSE);
        //System.out.println("rmse " + RMSE);
        return RMSE;
    }

    public double calculateMAE (double[] absError) {
        double MAE = 0.0;
        for (int i = 0; i < absError.length; i++) {
            MAE += absError[i];
        }
        MAE = MAE/absError.length;
        //System.out.println("rmse length" +squaredError.length);
        //System.out.println("rmse " + RMSE);
        return MAE;
    }

    public double calculateMAPE (double[] absPError) {
        double MAPE = 0.0;
        for (int i = 0; i < absPError.length; i++) {
            MAPE += absPError[i];
        }
        MAPE = MAPE/absPError.length;
        //System.out.println("rmse length" +squaredError.length);
        //System.out.println("rmse " + RMSE);
        return MAPE;
    }

    public void setMinMax(double[] dataSet) {
        RCaller caller = new RCaller();
        caller.setRscriptExecutable("/usr/bin/Rscript");

        RCode code = new RCode();
        code.clear();

        /*get maximum and minimum value from data set*/
        code.addDoubleArray("dataSet", dataSet);
        code.addRCode("maxVal <- max(dataSet)");
        code.addRCode("minVal <- min(dataSet)");
        code.addRCode("results <-list(max = maxVal, min=minVal)");
        caller.setRCode(code);
        caller.runAndReturnResult("results");
        double[] max = caller.getParser().getAsDoubleArray("max");
        double[] min = caller.getParser().getAsDoubleArray("min");
        setMaxValue(max[0]);
        setMinValue(min[0]);
    }

    public double[] normalizeData(double[] dataSet) {
        /*
         * Normalize Data Set between 0.2 and 0.8
         */
        double[] normalizedData = new double[dataSet.length];
        double maxD = getMaxValue();
        double minD = getMinValue();
        /*Normalized Data*/
```

```
    for (int i = 0; i < dataSet.length; i++) {
        normalizedData[i] = (0.8 - 0.2) * ((dataSet[i] - minD) / (maxD - minD)) + 0.2;
    }
    return normalizedData;
}

public double[] denormalizeData(double[] normalizedData) {
    /*
     * get original value of data set
     */
    double[] originalData = new double[normalizedData.length];
    double maxD = getMaxValue();
    double minD = getMinValue();
    /*Denormalized Data*/
    for (int i = 0; i < normalizedData.length; i++) {
        originalData[i] = (((normalizedData[i] - 0.2) / (0.8 - 0.2)) * (maxD - minD)) + minD;
    }
    return originalData;
}

/**
 * @return the weightsV
 */
public double inverseSigmoid(double x) {
    double inverse = Math.log(x) - Math.log(1 - x);
    return inverse;
}

public double[][] getWeightsV() {
    return weightsV;
}

/**
 * @param weightsV the weightsV to set
 */
public void setWeightsV(double[][] weightsV) {
    this.weightsV = weightsV;
}

/**
 * @return the weightsW
 */
public double[][] getWeightsW() {
    return weightsW;
}

/**
 * @param weightsW the weightsW to set
 */
public void setWeightsW(double[][] weightsW) {
    this.weightsW = weightsW;
}

/**
 * @return the minValue
 */
public double getMinValue() {
    return minValue;
}

/**
 * @param minValue the minValue to set
 */
public void setMinValue(double minValue) {
    this.minValue = minValue;
}
```

```
    /**
     * @return the maxValue
     */
    public double getMaxValue() {
        return maxValue;
    }

    /**
     * @param maxValue the maxValue to set
     */
    public void setMaxValue(double maxValue) {
        this.maxValue = maxValue;
    }

    /**
     * @return the weightsU
     */
    public double[][] getWeightsU() {
        return weightsU;
    }

    /**
     * @param weightsU the weightsU to set
     */
    public void setWeightsU(double[][] weightsU) {
        this.weightsU = weightsU;
    }
}
```

WeightsInitialization.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package timeseriesforecast;

/**
 *
 * @author Ega
 */
public class WeightsInitialization {
    /*
     * Here some methods to initialize weights
     */
    public double[][] useRandomWeight(int numOfSourceLayerUnit, int numOfDestinationLayerUnit) {
        /*
         * generate random number between -0.5 and 0.5 for weights initialization
         */


        int I = numOfSourceLayerUnit;
        int J = numOfDestinationLayerUnit;
        double [][] weights = new double [J][I+1];
        for (int j = 0; j < J; j++) {
            for (int i = 0; i <= I; i++) {
                weights[j][i] = Math.random() - 0.5;
            }
        }
        return weights;
    }

    public double[][] useRandomWeightForU(int numOfSourceLayerUnit, int numOfDestinationLayerUnit) {
        /*
         * generate random number between -0.5 and 0.5 for weights initialization
         */
```

```
        int I = numOfSourceLayerUnit;
        int J = numOfDestinationLayerUnit;
        double [][] weights = new double [J][I];
        for (int j = 0; j < J; j++) {
            for (int i = 0; i <I; i++) {
                weights[j][i] = Math.random() - 0.5;
            }
        }
        return weights;
    }
}
```

ARIMA.java

```
package timeseriesforecast;

import rcaller.RCaller;
import rcaller.RCode;

/**
 *
 * @author Ega
 */
public class ARIMA {

    private double[] input;
    private double[] output;

    public double[] getPredictionValueOnInputError(double[] input) {
        double [] output  = new double [input.length];
        /* Creating a RCaller */
        RCaller caller = new RCaller();
        caller.setRscriptExecutable("/usr/bin/Rscript");
        /* Creating a source code */
        RCode code = new RCode();
        code.clear();
        // add libraries needed to load data set
        code.addRCode("library(forecast)");

        //run forecast using auto.arima in R
        code.addDoubleArray("input", input);
        code.addRCode("fit <- auto.arima(input)");
        code.addRCode("y <- fitted.values(fit)");
        code.addRCode("z <- y");
        code.addRCode("y_predict <- as.matrix(z)");
        code.addRCode("results<-list(output = y_predict)");
        caller.setRCode(code);
        caller.runAndReturnResult("results");
        output = caller.getParser().getAsDoubleArray("output");
        return output;
    }
}
```