**Part 2: Multi-layer Perceptron Neural Network**

**INPUT VARIABLES USED FOR MLP EXCHANGE RATE FORCASTING**

MLP exchange rate forecasting is one of the most difficult time-series problems due to the numerous factors that can contribute to the price of an individual stock. There are many current methods to try and tackle this forecasting issue which I will now name.

Other inputs that are used for time-series are normally technical inputs such as the moving average, relative strength index, and other financial statistics. These paired with lagged inputs and prices, can be used for a deeper technical analysis and better predictions in time-series forecasting. As well as technical inputs, fundamental inputs can also be used. These include, consumer price index, foreign reserve, GDP, export, and import volume.

There are many variations of inputs, which include multivariate – the combination of lagged inputs and other fundamental or technical data (always a combination of technical and fundamental), for example using the GDPs of the countries along side their lagged prices to be able to predict future prices using todays price and GDP, this would be multivariate and may allow the neural network to make more accurate predictions based off of the increased patterns that can be found within these relationships. Overall, multivariate will use a combination of technical and fundamental inputs to try and forecast the exchange rates.

Univariate is another method, this relies solely on technical data, using statistical analysis of previous price points, the data is not external to past prices, and so it is relying heavily on the past patterns of the price index. This is used in AR (Auto Regression), which uses lagged inputs as different prices at intervals to try to predict future prices. This assumes there is a direct relation to prices in the past and the future prices.

Overall, both approaches are promising and necessary for a problem which we are still far from solving. The multivariate approach may sound intuitive but the univariate approach is more widely adopted among predictive models. There applicability may be varied and needs to be understood and further researched, however, in this report we will be building our first univariate based AR model to predict future prices based off past prices.

References:

* Huang, Wei & Lai, Kin Keung & Nakamori, Yoshiteru & Wang, Shouyang. (2004). Forecasting Foreign Exchange Rates With Artificial Neural Networks: A Review. International Journal of Information Technology and Decision Making. 3. 145-165. 10.1142/S0219622004000969.
* Dancker, J. (2023). *A Brief Introduction to Time Series Forecasting Using Statistical Methods*. [online] Medium. Available at: <https://towardsdatascience.com/a-brief-introduction-to-time-series-forecasting-using-statistical-methods-d4ec849658c3>.
* Abbate, A. and Marcellino, M. (2016). Point, Interval and Density Forecasts of Exchange Rates with Time-Varying Parameter Models. *SSRN Electronic Journal*. doi:https://doi.org/10.2139/ssrn.2801984.
* Lee, J., Takizawa, H. and Hauner, D. (2011). In Which Exchange Rate Models Do Forecasters Trust? *IMF Working Papers*, 11(116), p.1. doi:https://doi.org/10.5089/9781455262397.001.

‌

**LOADING DEPENDENCIES AND DATA**

At the beginning of our code, we first install and call the dependencies that will be used later…

library(neuralnet)

library(readxl)

library(grid)

library(MASS)

library(caret)

Now we will read the data into a data frame then into a vector for the price information.

################

#load data

financial = read\_excel("ExchangeUSD.xlsx")

head(financial)

View(financial)

#selecting the usd/eur price column

eu\_usd = financial$`USD/EUR`

eu\_usd

Now we can split the data and normalise **according to the min and max of only the train data!** This is so that any data that can be used with the NN Model later can be scaled with the min and max of the training data set, this is to help the model perform more accurately as its all scaled to the train data. Therefore, we will normalise the test dataset according to the min and max of the training set, so that there is no leakage of data in the normalisation.

## Splitting the data set into a test and train

data\_train = eu\_usd[1:400]

data\_test = eu\_usd[401:500]

tmax = max(data\_train)

tmin = min(data\_train)

normalise <- function(x) {

return((x - tmin) / (tmax - tmin))

}

norm\_dt = normalise(data\_train)

norm\_td = normalise(data\_test)

**I/O MATRIX**

We will need to now create I/O matrices for different versions of inputs – a different number of time series inputs. To be able to do this for any number of inputs, we will create a function that can create this time-series matrix just from the input of the dataset and the number of “lags” or steps backward.

lagged\_matrix\_inputs = function(data, lags) {

no\_lags = length(lags)

no\_rows = length(data) - max(lags)

input\_matrix=matrix(0, nrow = no\_rows, ncol = no\_lags)

for (i in 1:no\_rows) {

for (j in 1:no\_lags) {

input\_matrix[i, j] = data[i + lags[j] - 1]

}

}

return(input\_matrix)

}

The function will take the argument of the data, and the number of lags and then will create a new matrix based on the size of the number of lags. It then iterates through every row filling out all the columns one step along until the next row which will start one step in the future.

This matrix is called in a later function like so

train\_inputs <- lagged\_matrix\_inputs(norm\_dt, lags)

test\_inputs <- lagged\_matrix\_inputs(norm\_td, lags)

training\_outputs = norm\_dt[(max(lags)+1):length(norm\_dt)]

testing\_outputs = norm\_td[(max(lags)+1):length(norm\_td)]

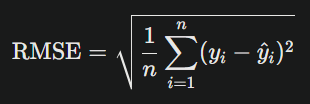
The outputs are created by taking the max lag + 1 (as we turned the lags into a vector from 1 to max(lags)), to the length of the series to slice out the nth + 1 entry in the dataset (where n = max(lags)), which will create a series of the numbers just after the final input eg. The next entry in the price data.

**TESTING VARIOUS NN PARAMETERS**

Testing out the different NN parameters like the hidden networks, the layers, and linear.output was done through a function which automatically ran each variation and added it to a table with its relevant RMSE, MAE, MAPE, and sMAPE stats listed, this allowed me to quickly test various parameters efficiently.

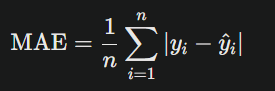
**4 STATISTICS DISCUSSION**

**RMSE – ROOT MEAN SQUARED**



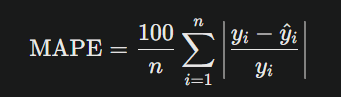
Is the square root of the average of the square of the differences. A higher number shows more error and its more sensitive to larger errors as squaring errors before averaging will make the larger ones stand out more in the analysis.

**MAE – MEAN ABSOLUTE ERROR**



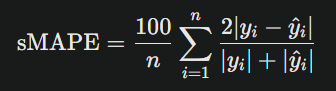
MAE dies not square the differences between the actual and predicted meaning that it treats all errors equally. Overall, it is a useful statistic for evaluating the error in a balanced manor.

**MAPE – MEAN ABSOLUTE PERCENTAGE ERROR**



This is very similar to MAE however it finds the sum off all the percentages of error and then finds then multiplies it by 100/n which will again find a total percentage of all the differences as a percentage between 0% and 200%, though it can have errors when the actual is close to zero as the errors can become infinitely large. Overall, it is useful to get a percentage of errors of the dataset.

**sMAPE – SYMMETRIC MEAN ABSOLUTE PERCENTAGE ERROR**



sMAPE is like MAPE as it gives a percentage between 0% and 200%, however it also equally penalises over and under forecast which makes it powerful metric for understanding the accuracy of a prediction model.

**TRAINING VARIOUS NNs FUNCTION**

This function took the arguments as the number of lags, the number of hidden layers and corresponding nodes, and the logistic or tanh activation functions.

# Function to train and evaluate MLP

train\_mlp <- function(n\_lags, n\_hidden, act\_fun, lin\_out) {

# Create lagged inputs

lags <- 1:n\_lags

train\_inputs <- lagged\_matrix\_inputs(norm\_dt, lags)

test\_inputs <- lagged\_matrix\_inputs(norm\_td, lags)

training\_outputs = norm\_dt[(max(lags)+1):length(norm\_dt)]

testing\_outputs = norm\_td[(max(lags)+1):length(norm\_td)

# Convert training\_outputs to a matrix with a single column

training\_outputs <- as.matrix(training\_outputs)

# Create a data frame combining train\_inputs and training\_outputs

train\_data <- cbind(train\_inputs, training\_outputs)

colnames(train\_data) <- c(paste0("input", 1:ncol(train\_inputs)), "output")

#Train MLP

mlp\_model <- neuralnet(output ~ .,

data = train\_data,

hidden = n\_hidden,

act.fct = act\_fun,

linear.output = lin\_out)

# Evaluate on test set

test\_preds <- compute(mlp\_model, test\_inputs)$net.result

# De-normalize predictions

test\_preds <- test\_preds\*(tmax - tmin) + tmin

test\_dn <- testing\_outputs\*(tmax - tmin) + tmin

#return(cbind(test\_preds, test\_dn))

# Calculate metrics

rmse <- sqrt(mean((test\_preds - testing\_outputs)^2))

mae <- mean(abs(test\_preds - testing\_outputs))

mape <- mean(abs((test\_preds - testing\_outputs)/testing\_outputs))\*100

smape <- (1/length(testing\_outputs))\*sum(2\*abs(test\_preds - testing\_outputs)/(abs(test\_preds) + abs(testing\_outputs)))\*100

return(c(rmse, mae, mape, smape))

**}**

I then set up a data frame which could hold the results of the 4 statistics. And created a new row from calling the function.

# Train multiple models

mlp\_results <- data.frame(matrix(nrow = 12, ncol = 6))

colnames(mlp\_results) <- c("Structure", "RMSE", "MAE", "MAPE", "SMAPE", "Description")

mlp\_results[1,] <- c("MLP1", train\_mlp(4, 5, "logistic", FALSE), "4 lags, 1 Hidden, 5 Neurons, Logistic, non-linear")

mlp\_results[2,] <- c("MLP2", train\_mlp(4, 10, "logistic", TRUE), "4 lags, 1 Hidden, 10 Neurons, Logistic, linear")

mlp\_results[3,] <- c("MLP3", train\_mlp(4, 5, "tanh", FALSE), "4 lags, 1 Hidden, 5 Neurons, TanH, non-linear")

mlp\_results[4,] <- c("MLP4", train\_mlp(4, 10, "tanh", TRUE), "4 lags, 1 Hidden, 10 Neurons, TanH, linear")

mlp\_results[5,] <- c("MLP5", train\_mlp(3, c(5,3), "logistic", TRUE), "3 lags, 2 Hidden, 5-3 Neurons, Logistic, linear")

mlp\_results[6,] <- c("MLP6", train\_mlp(3, c(8,5), "logistic", FALSE), "3 lags, 2 Hidden, 8-5 Neurons, Logistic, non-linear")

mlp\_results[7,] <- c("MLP7", train\_mlp(3, c(5,3), "tanh", TRUE), "3 lags, 2 Hidden, 8-5 Neurons, TanH, linear")

mlp\_results[8,] <- c("MLP8", train\_mlp(3, c(5,3), "tanh", FALSE), "3 lags, 2 Hidden, 5-3 Neurons, TanH, non-linear")

mlp\_results[9,] <- c("MLP9", train\_mlp(4, 6, "logistic", TRUE), "4 lags, 1 Hidden, 6 Neurons, Logistic, linear")

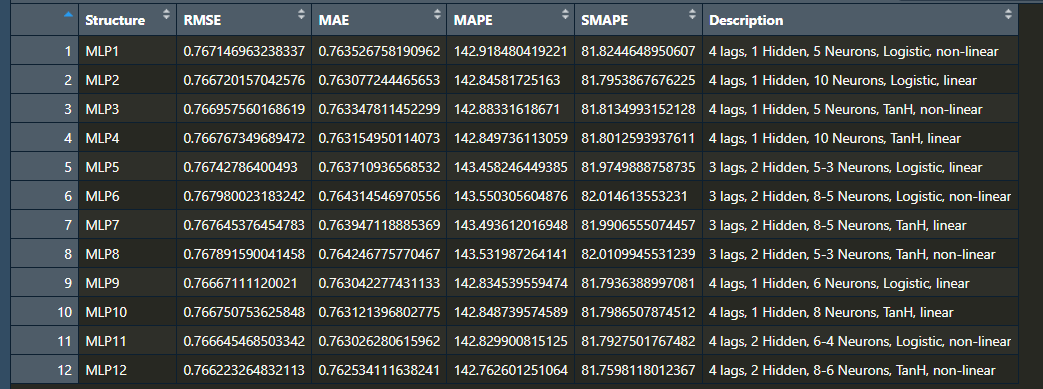
mlp\_results[10,] <- c("MLP10", train\_mlp(4, 8, "tanh", TRUE), "4 lags, 1 Hidden, 8 Neurons, TanH, linear")

mlp\_results[11,] <- c("MLP11", train\_mlp(4, c(6,4), "logistic", FALSE), "4 lags, 2 Hidden, 6-4 Neurons, Logistic, non-linear")

mlp\_results[12,] <- c("MLP12", train\_mlp(4, c(8,6), "tanh", FALSE), "4 lags, 2 Hidden, 8-6 Neurons, TanH, non-linear")

View(mlp\_results)

**NN COMPARISON TABLE**



**BEST MODEL EFFICIENCY**

Best one hidden layer -> MLP9: 4 lags, 1 hidden, 6 neurones, Logistic, linear

No. parameters = (4 \* 6) + 6 + (6 \* 1) + 1 = 37

Best 2 hidden layers -> MLP12: 4 lags, 2 hidden, 8-6 neurones, TanH, non-linear

No. parameters = (4 \* 8) + 8 + (8 \* 6) + 6 + (6 \* 1) + 1 = 101

Overall, MLP9 has less than half the parameters of MLP12, so in terms of efficiency I would prefer MLP9 although MLP12 has slightly better scores in the table, it is very negligible compared to the increase in parameters, which leads me to prefer MLP9.

**MAPPING MLP9**

As this is our preferred model, we will map its performance using a line graph.

|  |  |  |  |
| --- | --- | --- | --- |
| RMSE | MAE | MAPE | sMape |
| 0.76667111120021 | 0.763042277431133 | 142.834539559474 | 81.7936388997081 |

mlp\_result = train\_mlp(4, 6, "logistic", TRUE)

View(mlp\_result)

testing\_output = mlp\_result[, 2]

pred = mlp\_result[, 1]

y\_min <- min(c(testing\_output, pred))

y\_max <- max(c(testing\_output, pred))

x = 1:length(testing\_output)

plot(x, testing\_output, col = "red", type = "l", lwd=2,

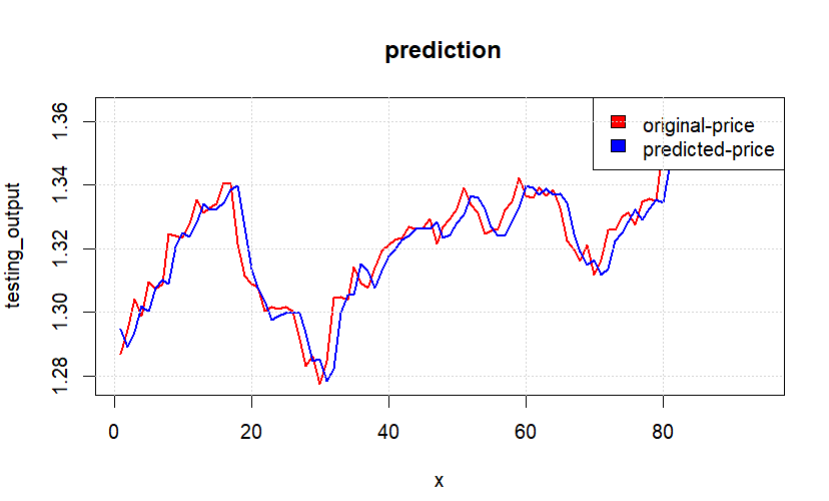
main = " prediction", ylim = c(y\_min, y\_max))

lines(x, pred, col = "blue", lwd=2)

legend("topright", legend = c("original-price", "predicted-price"),

fill = c("red", "blue"), col = 2:3, adj = c(0, 0.6))

grid()



**APPENDIX**

install.packages("neuralnet")

install.packages("grid")

install.packages("MASS")

install.packages("readxl")

install.packages("caret")

library(neuralnet)

library(readxl)

library(grid)

library(MASS)

library(caret)

################

#load data

financial = read\_excel("ExchangeUSD.xlsx")

head(financial)

View(financial)

#selecting the usd/eur price column

eu\_usd = financial$`USD/EUR`

eu\_usd

summary(eu\_usd)

##normalising the dataset NOTE: SCALING ENTIRE DATA BEFORE SPLITTING CAUSES DATA LEAKAGE

# eu\_usd = scale(eu\_usd)

# summary(eu\_usd)

# View(eu\_usd)

## Splitting the data set into a test and train

data\_train = eu\_usd[1:400]

data\_test = eu\_usd[401:500]

# #Convert to dataframe for comformality purposes

# data\_train\_df <- data.frame(price = data\_train)

# data\_test\_df <- data.frame(price = data\_test)

# ##DO THIS TO PREVENT SKEWED NORMALISATION IN THE TRAINING AND TESTING PHASE!!

#

# norm\_fit = preProcess(data\_train\_df, method = "range")

# norm\_dt = predict(norm\_fit, data\_train\_df)#data\_train

# norm\_td = predict(norm\_fit, data\_test\_df) #data\_test

tmax = max(data\_train)

tmin = min(data\_train)

tmax

tmin

normalise <- function(x) {

return((x - tmin) / (tmax - tmin))

}

norm\_dt = normalise(data\_train)

norm\_td = normalise(data\_test)

########################################

##CREATE LAGGED INPUTS FOR TIME-SERIES ANALYSIS

lagged\_matrix\_inputs = function(data, lags) {

no\_lags = length(lags)

no\_rows = length(data) - max(lags)

input\_matrix=matrix(0, nrow = no\_rows, ncol = no\_lags)

for (i in 1:no\_rows) {

for (j in 1:no\_lags) {

input\_matrix[i, j] = data[i + lags[j] - 1]

}

}

return(input\_matrix)

}

# #TESTING Creating the I/O matrices

# lags = 1:6

# train\_inputs <- lagged\_matrix\_inputs(norm\_dt, lags)

# View(norm\_dt)

# class(norm\_dt)

# length(norm\_dt)

#

# train\_inputs <- lagged\_matrix\_inputs(data\_train, lags)

# test\_inputs <- lagged\_matrix\_inputs(data\_train, lags)

# training\_outputs = data\_train[(max(lags)+1):length(data\_train)]

# testing\_outputs = data\_test[(max(lags)+1):length(data\_test)]

#

# View(train\_inputs)

# training\_outputs

# dim(train\_inputs)

# dim(training\_outputs)

# Function to train and evaluate MLP

train\_mlp <- function(n\_lags, n\_hidden, act\_fun, lin\_out) {

# Create lagged inputs

lags <- 1:n\_lags

train\_inputs <- lagged\_matrix\_inputs(norm\_dt, lags)

test\_inputs <- lagged\_matrix\_inputs(norm\_td, lags)

training\_outputs = norm\_dt[(max(lags)+1):length(norm\_dt)]

testing\_outputs = norm\_td[(max(lags)+1):length(norm\_td)]

# Convert training\_outputs to a matrix with a single column

training\_outputs <- as.matrix(training\_outputs)

# Create a data frame combining train\_inputs and training\_outputs

train\_data <- cbind(train\_inputs, training\_outputs)

colnames(train\_data) <- c(paste0("input", 1:ncol(train\_inputs)), "output")

# dim1 = length(training\_outputs)

# dim2 = dim(train\_inputs)

# #

# class1 = class(train\_outputs\_norm)

# class2 = class(train\_inputs\_norm)

#

# dim(train\_inputs\_norm)

# train\_outputs\_norm

# train\_inputs\_norm

#

# return(c(dim1, dim2, class1, class2)) ##Error handling arrays not comformable

#return(dim(train\_inputs))

#Train MLP

mlp\_model <- neuralnet(output ~ .,

data = train\_data,

hidden = n\_hidden,

act.fct = act\_fun,

linear.output = lin\_out)

# Evaluate on test set

test\_preds <- compute(mlp\_model, test\_inputs)$net.result

# De-normalize predictions

test\_preds <- test\_preds\*(tmax - tmin) + tmin

test\_dn <- testing\_outputs\*(tmax - tmin) + tmin

#return(cbind(test\_preds, test\_dn))

# Calculate metrics

rmse <- sqrt(mean((test\_preds - testing\_outputs)^2))

mae <- mean(abs(test\_preds - testing\_outputs))

mape <- mean(abs((test\_preds - testing\_outputs)/testing\_outputs))\*100

smape <- (1/length(testing\_outputs))\*sum(2\*abs(test\_preds - testing\_outputs)/(abs(test\_preds) + abs(testing\_outputs)))\*100

return(c(rmse, mae, mape, smape))

}

mlp\_result = train\_mlp(4, 6, "logistic", TRUE)

View(mlp\_result)

testing\_output = mlp\_result[, 2]

pred = mlp\_result[, 1]

y\_min <- min(c(testing\_output, pred))

y\_max <- max(c(testing\_output, pred))

x = 1:length(testing\_output)

plot(x, testing\_output, col = "red", type = "l", lwd=2,

main = " prediction", ylim = c(y\_min, y\_max))

lines(x, pred, col = "blue", lwd=2)

legend("topright", legend = c("original-price", "predicted-price"),

fill = c("red", "blue"), col = 2:3, adj = c(0, 0.6))

grid()

# Train multiple models

mlp\_results <- data.frame(matrix(nrow = 12, ncol = 6))

colnames(mlp\_results) <- c("Structure", "RMSE", "MAE", "MAPE", "SMAPE", "Description")

mlp\_results[1,] <- c("MLP1", train\_mlp(4, 5, "logistic", FALSE), "4 lags, 1 Hidden, 5 Neurons, Logistic, non-linear")

mlp\_results[2,] <- c("MLP2", train\_mlp(4, 10, "logistic", TRUE), "4 lags, 1 Hidden, 10 Neurons, Logistic, linear")

mlp\_results[3,] <- c("MLP3", train\_mlp(4, 5, "tanh", FALSE), "4 lags, 1 Hidden, 5 Neurons, TanH, non-linear")

mlp\_results[4,] <- c("MLP4", train\_mlp(4, 10, "tanh", TRUE), "4 lags, 1 Hidden, 10 Neurons, TanH, linear")

mlp\_results[5,] <- c("MLP5", train\_mlp(3, c(5,3), "logistic", TRUE), "3 lags, 2 Hidden, 5-3 Neurons, Logistic, linear")

mlp\_results[6,] <- c("MLP6", train\_mlp(3, c(8,5), "logistic", FALSE), "3 lags, 2 Hidden, 8-5 Neurons, Logistic, non-linear")

mlp\_results[7,] <- c("MLP7", train\_mlp(3, c(5,3), "tanh", TRUE), "3 lags, 2 Hidden, 8-5 Neurons, TanH, linear")

mlp\_results[8,] <- c("MLP8", train\_mlp(3, c(5,3), "tanh", FALSE), "3 lags, 2 Hidden, 5-3 Neurons, TanH, non-linear")

mlp\_results[9,] <- c("MLP9", train\_mlp(4, 6, "logistic", TRUE), "4 lags, 1 Hidden, 6 Neurons, Logistic, linear")

mlp\_results[10,] <- c("MLP10", train\_mlp(4, 8, "tanh", TRUE), "4 lags, 1 Hidden, 8 Neurons, TanH, linear")

mlp\_results[11,] <- c("MLP11", train\_mlp(4, c(6,4), "logistic", FALSE), "4 lags, 2 Hidden, 6-4 Neurons, Logistic, non-linear")

mlp\_results[12,] <- c("MLP12", train\_mlp(4, c(8,6), "tanh", FALSE), "4 lags, 2 Hidden, 8-6 Neurons, TanH, non-linear")

View(mlp\_results)