CS230

Macroeconomic and Technical Forecast of USDJPY Daily Spot Rate using Deep Neutral Network

Ren Hao Tan
Department of Computer Science
Stanford University
renhao@stanford.edu

Abstract

Daily USDJPY exchange rates are influenced by a host of factors ranging from macroeconomic trends, expectation of monetary policies and speculative investor action based on past price actions. A deep neural network trained on current and historical information is shown to be able to predict the next-day USDJPY open rate.

Index Terms—deep learning, foreign exchange rate prediction, investment science, neural network, time-series analysis (G11, G12, G14, G17)

I. Introduction

The prediction of daily USDJPY exchange rate based on past data is an interesting topic because the specification of USDJPY rate is influenced by a host of factors ranging from macroeconomic trends, expectation of monetary policies and speculative investor action based on technical factors. Despite being a complex and nonlinear problem, the set of determining factors for USDJPY rate seems to be reasonably finite; this suggests that a well-trained neural network could be effective in predicting price movements.

Understanding USDJPY dynamics is important not only in creating opportunities for profitable trades. It is also critical insofar as it informs policymakers of potential impacts each decision could have on the external economy. For many countries which relate heavily on the external market (e.g. Singapore), the movement of the exchange rate is used as a key monetary policy tool to affect the economy, in lieu of more conventional interest rate policy mechanisms. I have chosen USDJPY as the benchmark rate as it is one of the most widely referenced exchange rate.

II. Dataset and Features

This paper has identified 21 variables which economic literature has shown to be predictive–at least in theory–of fluctuations in USDJPY foreign exchange rates. They are:

- Spot daily rates of other major currencies: EURUSD, GBPUSD, USDCNY, NZDUSD and USDCHF
- Consumer price inflation rates of US and Japan
- Close price of stock indices in US and Japan: S&P 500 Index and Nikkei 225
- Yields of government bonds in US and Japan: US 13 Week Treasury Bill, US Treasury 10 Year Bond, US Treasury 30 Year Bond, Japanese Government 2 Year Bond, Japanese Government 10 Year Bond
- Export and import price indices in US and Japan
- Cross-border trading volumes between US and Japan
- CBOE volatility index (VIX)

These variables dated from 1995-03-31 to 2017-09-11 were extracted manually from FRED, Quandl, Yahoo Finance, Bloomberg and Bureau of Labor Statistics and transformed into daily frequencies. Observations on the first 5800 days, next 1200 days and last 1000 days of the above-mentioned period were placed into the training, dev and test sets (58:15:10) respectively.

III. Method: Baseline Regression

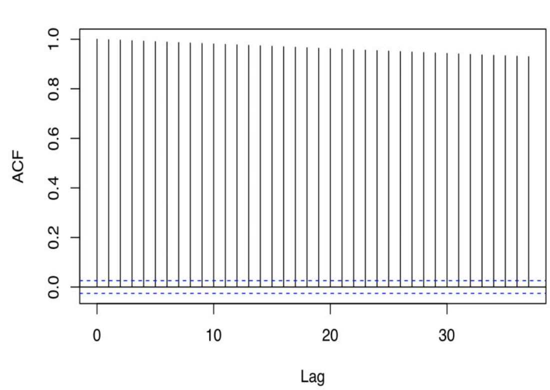
Multiple linear regression analysis was used to test if these 21 covariates X_t significantly predicted the next-day USDJPY rate, denoted as y_{t+1}. The linear regression model, when applied on the dev-set, indicated that these 21 covariates combined explained only 24.8% of the variance (RMSE=17.10).

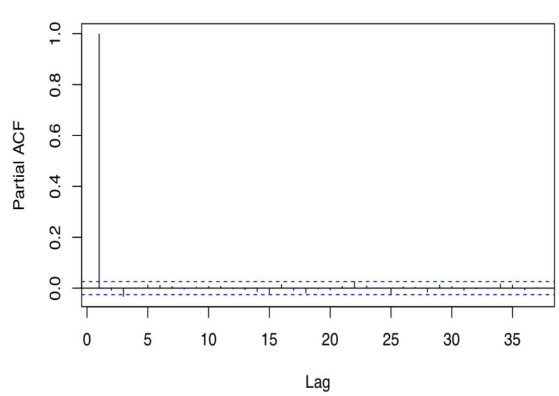
(1) $$y_{t+1} \approx \beta\, \mathrm{X}_t$$

A one-day lag of covariate was clearly insufficient in explaining y_{t+1}. Based on the weak form of Efficient Market Hypothesis, which claims that prices on traded assets (e.g., stocks, bonds, or property) already reflect all past publicly available information, we hypothesize that y_t captures much of the variations of the 21 variables prior to day t which are relevant in predicting y_{t+1}. If this is true, a robust model for y_{t+1} could omitted covariates prior to day t and be described as:

$$y_{t+1} \approx f(y_t\,, \mathrm{X}_t)$$

The partial autocorrelation function (PACF) of y_t corroborated this hypothesis as it sufficiently cuts off after lag = 1.





Furthermore, an ARIMA(1,0,0) model based only on y_t

(2) $$y_{t+1} \approx \beta\, y_t$$

had a dev-set RMSE of 0.674 which is below that of model (1) .

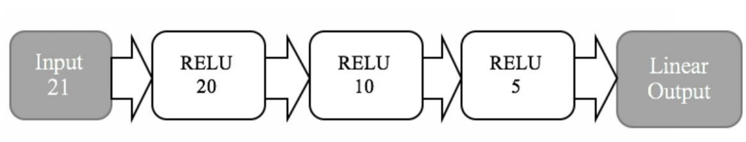
In combination, a 1-lag autoregressive model with macroeconomic regressors was found to be powerful in delivering a low dev-set RMSE of 0.520.

(3) $$y_{t+1} \approx \beta_1\, y_t + \beta_2\, X_t$$

IV. Deep Learning Model Specification

This paper considered two deep learning architectures–(i) deep neural network with the inclusion of y_t as an input for prediction of y_{t+1} and (ii) a Long Short Term Memory (LSTM) network which did not require an explicit inclusion of y_t in predicting y_{t+1} due to the architecture's ability to store past information in a time sequence. We inferred from the excellent performance of (3) that deep neural network should be sufficient (if not more robust due to reduction of noise) in tackling this problem.

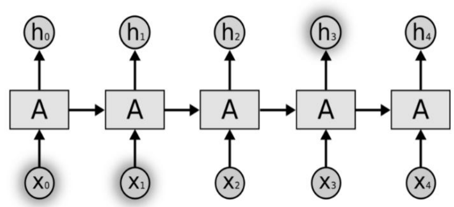
(4)



Loss Function: Mean Squared Error (MSE);
Adam Optimizer; Batch Size: 128; Epoch: 100

Prior to any hyper-parameter tuning, (4) generated a RMSE of 0.688 which was comparable to (2). The model shows promise to yield better results.

(5)



LSTM(50) > ReLu(10) > Linear(1); Loss Function:
Mean Squared Error (MSE); Adam Optimizer; Batch Size: 128; Epoch: 100

As expected, the LSTM architecture performed poorly across many varying hyper-parameters. For the specific set of hyperparameters outlined in (5), it had a RMSE of 24.35.

Model (4) is therefore chosen for the project. In its hidden layers, we employ ReLu activation function to accelerate learning and avoid vanishing gradient. Since we are solving a regression problem, a ReLu function is also used for the outcome layer to generate a real value. ReLu is particularly suitable for the output layer in this case because USDJPY rates are all positive real numbers. Mathematically, the deep network is described as follows:

$$h_t^{[1]} = ReLu(W_h^{[1]} X_t + b_h^{[1]})$$

$$h_t^{[2]} = ReLu(W_h^{[2]} h_t^{[1]} + b_h^{[2]})$$

$$\tilde{y}_t = ReLu(W_y h_t^{[2]} + b_y)$$

where $h_t^{[i]}$ represents the i-th hidden layer and *W, U* and *b* are parameter matrices. Loss function used is MSE.

V. Hyperparameter Tuning

First, preliminary optimization of the number of training iterations, N, is performed on the chosen model (4) using the dev set RMSE as the minimizing metrics.

Hyperparams	Tuned Value(s)
Epoch	100,500,**1000**,2500,5000

Tune 1: Preliminary Number of Epochs

Epoch=1000 is used for subsequent tuning as it gave the lowest RMSE (0.345) on the validation set. Then, the number of nodes n_x in each of the 3 hidden layer and dropout regularization in the first 2 hidden layers are tuned.

Hyperparams	Tuned Value(s)
n_1	**20**,40,80
n_2	10,20,40
n_3	**5**,10
dropout1	**0,**0.3,0.7
dropout2	**0**,0.3,0.7

Tune 2: Size of NN & Dropout

Based on 162 combinations of the above hyperparams, the 3 configurations with the lowest RMSE on the dev set are:

n_1	n_2	n_3	dropout1	dropout2	devRMSE
20	20	5	0	0	0.217
20	40	5	0	0	0.244
20	10	5	0	0	0.254

The fourth best configuration 40>10>5 with no regularization has significantly higher devRMSE of 0.328. Also, from the previous results on the dev set, it seems that more search could be conducted for n_2 to find more optimal configurations. In this final set of tuning, n_2 is tuned together with the number of training iterations, N around epoch=1000, while setting n_1=20, n_3=5, dropout1=0, dropout2=0.

Hyperparams	Tuned Value(s)
Epoch	750,1000,1500, **2000**
n_2	10,20,30,40,**50**

Tune 3: Fine-tune hidden layer 2 & #epoch

Altogether, the chosen hyperparameter configuration is as follows:

- ReLu[20]>ReLu[50]>ReLu[5]
- No dropout regularization
- Epoch=2000
- Loss: MSE
- Adam Optimizer

VI. Results

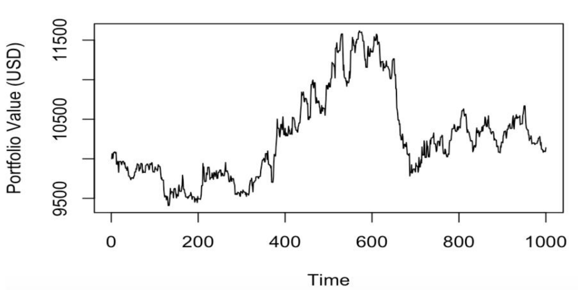
The results of the various models developed in this paper are summarized as follows:

Model	Train RMSE	Dev RMSE	Test RMSE
Simple linreg on covariates only	5.511	17.10	
Linreg with autoregression	0.671	0.520	
LSTM	0.084	24.35	
NN pre tuning	0.469	0.688	
NN post tuning	**0.237**	**0.163**	**0.235**

The tuned model achieved a test RMSE of 0.235 and test MAE of 0.222.

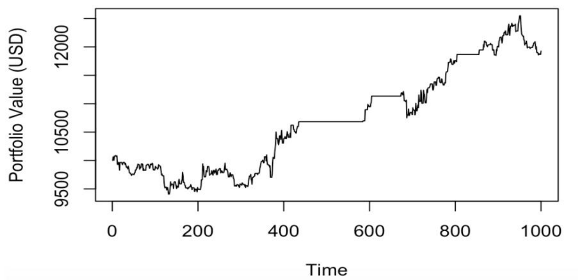
VII. Portfolio Simulation

A hypothetical portfolio of $10,000 on the test set is simulated. Investor A invests entire portfolio in USD if the model prediction of next-day USDJPY is higher than the current rate. Conversely, he shorts USDJPY if the predicted next-day rate is lower.



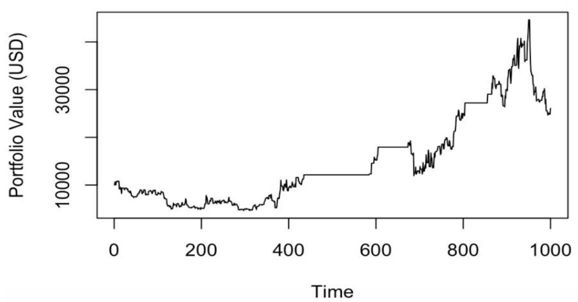
Investor A would have scored an annualized investment return of 0.3% assuming no leverage and no transaction costs.

It is however unrealistic to assume an investor will invest if the prediction differs marginally from the current rate. Instead, assume Investor B is like A except that he invests only when the forecasted deviation exceeds the mean absolute error of the dev set (~0.25).



The portfolio performance improves significantly to 6.8% annualized return; as shown, investor B is not compelled to invest during a period when the model was not producing a strong signal (around t= 500).

Lastly, we model a third investor C who engages in leveraged positions (which is a staple in foreign currency trading). We assume a fixed 10x leverage.



At the expense of increase volatility, Investor is able to generate an impressive return of 54.3% each year.

The portfolio metrics of each investment strategy on the test set is outlined as follows:

Investment Strategy	Annualized Returns	Sharpe Ratio
A: no leverage; no forecast margin	0.3%	0.10
B: no leverage; margin=dev_MAE	6.8%	0.834
C: 10x leverage; margin=dev_MAE	54.3%	0.834

VIII. DISCUSSION

This paper has demonstrated that a deep neural network with an autoregression lag of one is a robust architecture in forecasting the next-day USDJPY exchange rate, when used in conjunction with relevant macroeconomic and financial indicators commonly monitored by central banks and financial market participants.

Initial investigations of this paper also empirically validated claims of the Weak Efficient Market hypothesis by showing that current price is a sufficient, if not more superior, indicator of past macroeconomic information in predicting future prices. In other words, the foreign exchange market seems to "price in" past information efficiently. This is to be expected given the huge liquidity and transaction volumes of the USDJPY market. Further work could possibly investigate if similar claims could be made in more exotic currency pairs which are traded less frequently.

The deep neural network trained on MSE not only gave low test error rates, but also translated well into hypothetical portfolio performance on the test set. There are three possible areas of extension to improve the profitability of this model. Firstly, instead of using MSE as the loss function, one could train either the annualized return rate or Sharpe ratio as the objective. Secondly, the amount of leverage and the forecast margin before which investments are executed are hyperparameters which could be tuned on the dev set before use on the test set. Currently, these parameters are set *a priori*. Lastly, a three class classification-based neural network could also be built to predict if the next-day rate would increase (or decrease) past a certain threshold. An ensemble could be designed such that the model only "participate" in a prediction of there is concurrence between the classification model and regression model.

References

[1] E. Hadavandi, H. Shavandi, A. Ghanbari, "Integration of genetic fuzzy systems and artificial neural networks for stock price forecasting", Knowledge-Based System, Vol. 23, No. 8, 2010, pp. 800-808.

[2] A. Fan, and M. Palaniswami, "Stock Selection Using Support Vector Machines", Proceedings of the International Joint Conference on Neural Networks, Vol. 3, 2001, pp. 1793-1798.

[3] K.J. Kim, and W. Lee, "Stock Market Prediction Using Artificial Neural Networks", Neural Computing & Applications, Vol. 13, No. 3, 2004, pp. 255-250.

[4] K.Y., Shen, "Implementing Value Investing Strategy by Artificial Neural Network", International Journal of Business and Information Technology, Vol. 1, No. 1, 2010, pp. 12-22.

Appendix: Code and Methodology

Data Extraction

Data is extracted manually as .csv files from FRED, Quandl, Yahoo Finance and Bureau of Labour Statistics. All dates between 1990-01-01 to 2017-12-31 are generated, and the 21 downloaded tuples are joined in Excel using vlookup(). Then for all covariates except USDJPY (the response), lag by t=1 since only the t-1 variables should be available to predict t.

Data Import

````
```{r import}
macromodel <- read.csv(file="USDJPY.csv", header=TRUE, sep=",")
macromodel$Date <- as.Date(macromodel$Date)
```
````

Data Cleaning

Apply LOCF so that monthly data is synchronized into daily frequencies. Remove missing observations (which are all before a certain date due to LOCF). Split into 58:15:10 train-dev-test sets.

```
#Last Observation Carried Forward
library(zoo)
macromodel<-na.locf(macromodel, na.rm = TRUE)
sum(is.na(macromodel))

#Turn chr into num types
for (i in 2:22){
  macromodel[, i]  <- as.numeric(macromodel[, i])
}

#remove observations with missing value (i.e. before 1995-03-31)
macromodel.nafix <- na.omit(macromodel)
sum(is.na(macromodel.nafix))

library("caret")

#split into train-dev-test roughly 58:12:10
train.set <- macromodel.nafix[1:5800,]
dev.set <- macromodel.nafix[5801:7000,]
test.set <- macromodel.nafix[7001:8000,]
```

Data Exploration
####Simple linear regression on covariate

```
```{r simple1}
train.set1 <- train.set
train.set1$Date <- NULL

#Linear Regression
lmFit<-train(USDJPY~.,data = train.set1, method="lm")
summary(lmFit)

dev.set1 <- dev.set
dev.set1$Date <- NULL

#Evaluation of Linear Regression on train Set
predicted.lmFit<-predict(lmFit,train.set1)
modelvalues.lmFit<-data.frame(obs = train.set1$USDJPY, pred=predicted.lmFit)
defaultSummary(modelvalues.lmFit)

#Evaluation of Linear Regression on Dev Set
predicted.lmFit<-predict(lmFit,dev.set1)
modelvalues.lmFit<-data.frame(obs = dev.set1$USDJPY, pred=predicted.lmFit)
defaultSummary(modelvalues.lmFit)

```
```

####Time series analysis (plot ACF, PACF)

```
```{r lag}
##transform USDJPY into time series
train.set3 <- train.set
USDJPY2 <- train.set3$USDJPY
train.set3$Date[1]
train.set3$Date[length(USDJPY2)]

USDJPY.ts <- ts(USDJPY2)
plot(USDJPY.ts)

##plot ACF and PACF for the time series
acf(USDJPY.ts)
pacf(USDJPY.ts)
```
```

```
Call:
arima(x = USDJPY.ts, order = c(1, 0, 0))

Coefficients:
         ar1  intercept
      0.9993   109.5580
s.e.  0.0007    13.1669

sigma^2 estimated as 0.4542:  log likelihood = -5944.4,  aic = 11894.81

Training set error measures:
                       ME     RMSE       MAE          MPE   MAPE     MASE
Training set -0.001835565 0.673947 0.4019397 -0.005401689 0.36689 1.006833
                   ACF1
Training set 0.01667421
```

```
```{r arima}
##fit arima on USDJPY alone
arima.model<- arima(USDJPY.ts, order=c(1,0,0))
summary(arima.model)
```

As shown, the ARIMA(1,0,0) model alone provides a low training RMSE of 0.674, close to that of the autoregressive model with t-1 regressors used. This make sense as the previous day's USDJPY would provide a strong anchor for the next day's price.
```

####Linear regression on covariate and lag =1

```r
##add previous day's USDJPY as predictor of today's USDJPY
train.set2 <- train.set
USDJPY1 <- train.set2$USDJPY
USDJPY1<-append(USDJPY1, 0, after=0)
USDJPY1<-USDJPY1[-length(USDJPY1)]

train.set2$Date <- NULL
train.set2$USDJPY1 <- USDJPY1
train.set2 <- train.set2[2:dim(train.set2),]

#Linear Regression
lmFit2<-train(USDJPY~.,data = train.set2, method="lm")
summary(lmFit2)
```

LSTM

```r
```{r LSTM}
##try a simple LSTM with 50 neurons in 1st hidden layer, 1 "relu" layer and 1 linear layer for output

X_train.lstm<-X_train
X_dev.lstm<-X_dev

#center & scale
preProc <- preProcess(X_train.lstm, method = c("center", "scale"))
X_train.lstm<-predict(preProc, X_train.lstm)
preProc2 <- preProcess(X_dev.lstm, method = c("center", "scale"))
X_dev.lstm<-predict(preProc2, X_dev.lstm)

#reshape 2D into 3D
X_train.lstm<-array_reshape(x=as.matrix(X_train.lstm), dim = list(nrow(X_train.lstm), 1,
ncol(X_train.lstm)))
X_dev.lstm<-array_reshape(x=as.matrix(X_dev.lstm), dim = list(nrow(X_dev.lstm), 1, ncol(X_dev.lstm)))


##set up model
model.lstm <- keras_model_sequential()
model.lstm %>%
  layer_lstm(units = 50, input_shape = c(1,21),kernel_initializer='normal') %>%
  layer_dense(units = 10, activation = 'relu',kernel_initializer='normal') %>%
  layer_dense(units = 1, activation = 'linear',kernel_initializer='normal')

summary(model.lstm)
```

```r
##compile with loss function MSE
##default is adam's paper values
model.lstm %>% compile(
  optimizer = optimizer_adam(),
  loss = 'mse')

##train model with 1000 epochs and 128 batch size
history<-model.lstm %>% fit(
  X_train.lstm,
  as.matrix(Y_train),
  epochs=1000,
  batch_size=128,
  verbose=1)

plot(history)

# calculate pseudo R-squared
predict.lstm <- model.lstm %>% predict(X_dev.lstm, batch_size=128)
predict.lstm<-as.vector(predict.lstm)

plot(predict.lstm,Y_dev)

rmse.lstm<-sqrt(mean((Y_dev-predict.lstm)^2))
rmse.lstm
```
```

Deep NN

```r
##set up flags
FLAGS <- flags(
  flag_numeric("layer1size", 20),
  flag_numeric("layer2size", 10),
  flag_numeric("layer3size",5),
  flag_numeric("epoch",2000),
  flag_numeric("dropout1",0),
  flag_numeric("dropout2",0)
)

##set up model
model <- keras_model_sequential()
model %>%
  layer_dense(units = FLAGS$layer1size, input_shape = k, activation = 'relu',kernel_initializer='normal') %>%
  layer_dropout(rate = FLAGS$dropout1) %>%
  layer_dense(units = FLAGS$layer2size, activation = 'relu',kernel_initializer='normal') %>%
  layer_dropout(rate = FLAGS$dropout2) %>%
  layer_dense(units = FLAGS$layer3size, activation = 'relu',kernel_initializer='normal') %>%
  layer_dense(units = 1, activation = 'linear',kernel_initializer='normal')
summary(model)

##compile with loss function MSE
##default is adam's paper values
model %>% compile(
  optimizer = optimizer_adam(),
  loss = 'mse',
  metrics = 'mae')

##train model with 1000 epochs and 128 batch size
history<-model %>% fit(
  as.matrix(X_train),
  as.matrix(Y_train),
  validation_data=list(as.matrix(X_test), as.matrix(Y_test)),
  epochs=FLAGS$epoch,
  batch_size=128,
  verbose=1)

plot(history)

# calculate RMSE
predict <- model %>% predict(as.matrix(X_test), batch_size=128)
predict<-as.vector(predict)

plot(predict,Y_test)
```

Hyperparameter tuning

```r
```{r tuning}
library(tfruns)

runs <- tuning_run("model.R", flags = list(
  layer1size = c(20,40,80),
  layer2size = c(10,20,30,40,50),
  layer3size = c(5,15),
  epoch = c(500,1500,2000,2500,5000),
  dropout1= c(0,0.3,0.7),
  dropout2=c(0,0.3,0.7)
))

View(ls_runs())
```
```

Portfolio Simulation

```r
X_test$nnpredict <- predict
X_test$actual <- Y_test

portfolio <- rep(0,999)
portfolio[1] = 10000

for (i in 1:999) {
  today_price <- X_test[i, "USDJPY1"]
  next_day_guess <- X_test[i, "nnpredict"]
  next_day_realize  <- X_test[i, "actual"]
  if(next_day_guess>today_price){
    #long USDJPY
    portfolio[i+1] = portfolio[i]*(1+(next_day_realize/today_price-1))
  }
  if(next_day_guess<today_price){
    #short USDJPY
    portfolio[i+1] = (portfolio[i])*(1-(next_day_realize/today_price-1))
  }
  else{
    portfolio[i+1]=portfolio[i]
  }
}

portfolio[999]
plot(portfolio, type='l', xlab="Time", ylab="Portfolio Value (USD)")

portfolio_returns <- rep(0,999)
for (i in 1:999){
  portfolio_returns[i+1] <- portfolio[i+1]/portfolio[i]-1
}

sqrt(365)*(mean(portfolio_returns)/sqrt(var(portfolio_returns)))
```