Forex Trading with RNN

To simplify the problem, assume we always buy in at the start of the day and sell at the end of day. The key problem now is to predict days that will generate positive return. Assume we have $1000 principal at the start.

Read data into appropriate format

```
# read the data while discard the last column
df <- read.csv('DAT_ASCII_GBPUSD_M1_2017.csv', sep = ";",
         col.names = c('timestamp', 'open', 'high', 'low', 'close','-'),
         stringsAsFactors=FALSE)[-6] %>%
  mutate(timestamp = as.POSIXct(timestamp, format="%Y%m%d %H%M%S")) %>%
  select(-c(high, low)) %>%
  mutate(date = date(timestamp))
```

Data preprocessing

Extract daily open and close price

```
daily.open <- df %>%
  group_by(date) %>%
  filter(timestamp == min(timestamp)) %>%
  ungroup() %>%
  select(open, date)

daily.close <- df %>%
  group_by(date) %>%
  filter(timestamp == max(timestamp)) %>%
  ungroup() %>%
  select(close, date)

daily.df <- daily.open %>%
  merge(daily.close, by='date',
        all.x=T, all.y=T) %>%
  mutate(month = month(date)) %>%
  mutate(return = close/open-1) %>%
  mutate(day_of_mon = mday(date))

summary(daily.df)
```

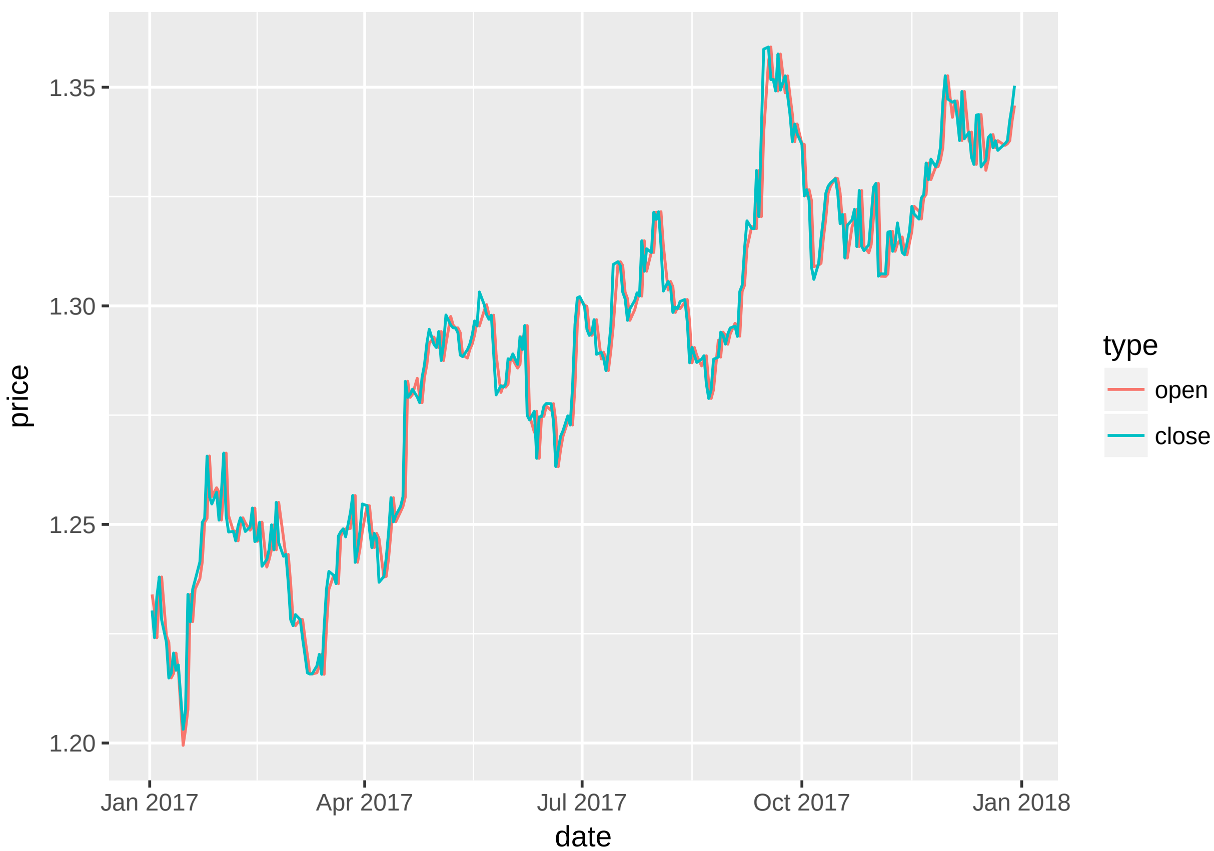
```
##       date                 open           close           month
##  Min.   :2017-01-02   Min.   :1.199   Min.   :1.203   Min.   : 1.000
##  1st Qu.:2017-04-02   1st Qu.:1.253   1st Qu.:1.254   1st Qu.: 4.000
##  Median :2017-07-01   Median :1.293   Median :1.293   Median : 6.500
##  Mean   :2017-07-01   Mean   :1.288   Mean   :1.289   Mean   : 6.494
##  3rd Qu.:2017-09-28   3rd Qu.:1.317   3rd Qu.:1.318   3rd Qu.: 9.000
##  Max.   :2017-12-29   Max.   :1.359   Max.   :1.359   Max.   :12.000
```

```
##       return             day_of_mon
##  Min.   :-0.0160010   Min.   : 1.00
##  1st Qu.:-0.0018463   1st Qu.: 8.00
##  Median : 0.0005080   Median :16.00
##  Mean   : 0.0003912   Mean   :15.73
##  3rd Qu.: 0.0028817   3rd Qu.:23.00
##  Max.   : 0.0218108   Max.   :31.00
```

Exploratory analysis

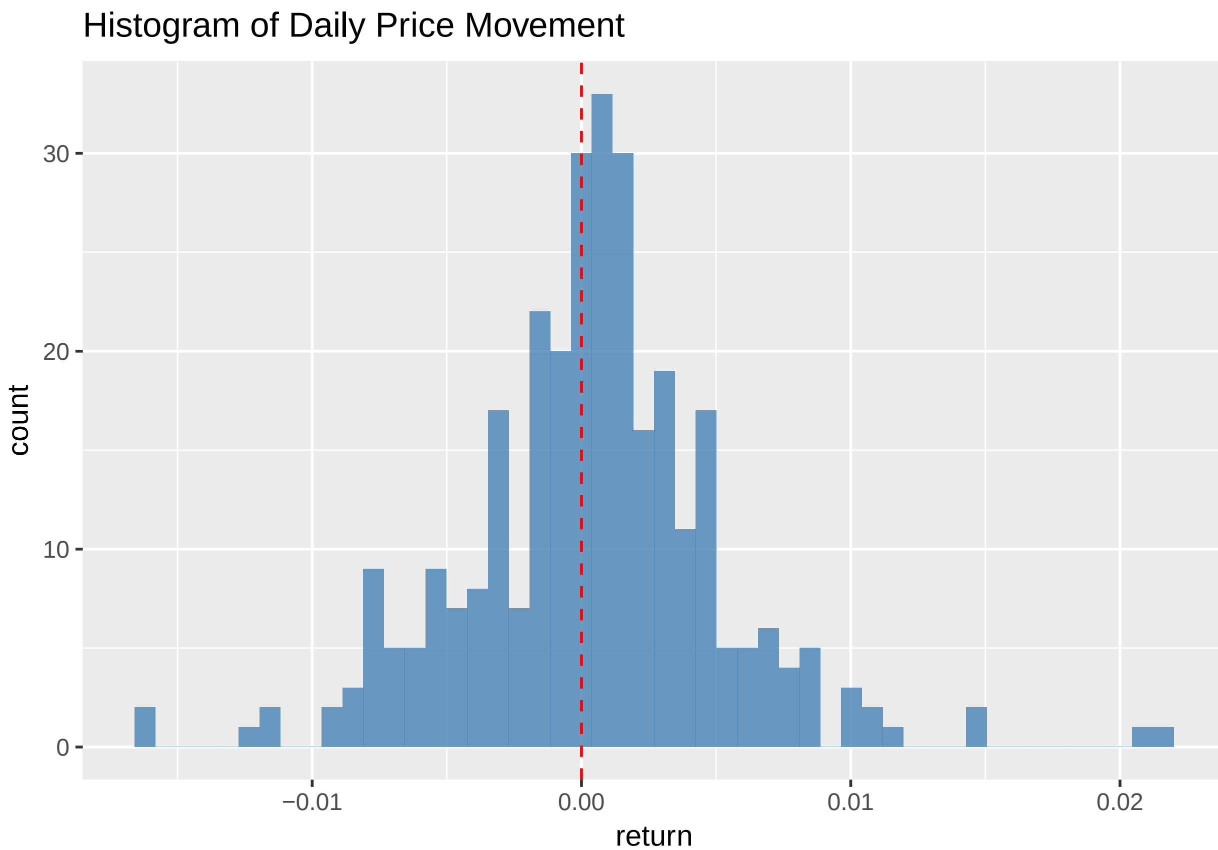
how does daily open and close price differ to each other?

```
melted <- daily.df %>%
  select(c(date, month, open, close)) %>%
  melt(id = c("date", "month")) %>%
  rename(type = variable, price = value)

ggplot(data = melted, aes(x=date, y=price)) +
  geom_line(aes(color=type))
```



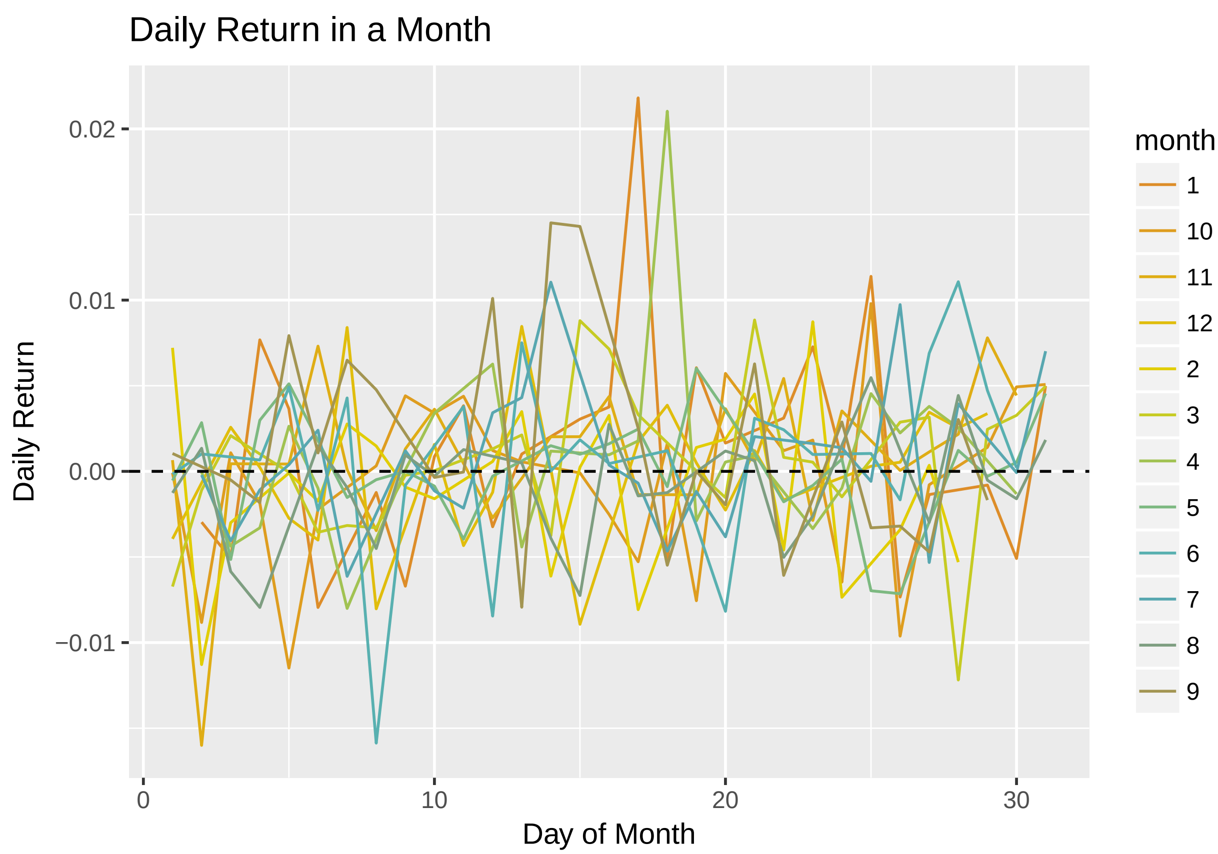
What kind of daily return to expect?

```r
daily.df %>%
  ggplot(aes(x=return))+
  geom_histogram(alpha=0.8, fill='#4682b4',bins=50)+
  geom_vline(xintercept = 0, color="red", linetype="dashed")+
  ggtitle("Histogram of Daily Price Movement")
```



Is there seasonality in return?

```r
# seasonality of return
daily.df %>%
  mutate(month = as.character(month)) %>%
  ggplot(aes(x=day_of_mon, y=return) ) +
  geom_line(aes(color=month)) +
  geom_hline(yintercept = 0, color='black', linetype='dashed') +
  scale_color_manual(values=wes_palette(type = 'continuous', 18,
                                        name = 'FantasticFox1')) +
  ggtitle('Daily Return in a Month')+
  xlab('Day of Month')+
  ylab('Daily Return')
```



Trading Algorithm

Preparation: train-val split & function for backtest

```
# train - val split
train = daily.df[daily.df$month<=10,]
val = daily.df[daily.df$month>10,]

# helper function for backtest
BackTest <- function(decision_vec, principal=1000) {
  # args:
  #   1. decision_vec: vector of 1 or 0s indicating the days we are buying in
  #   2. principal: be default to be 1000
  # output:
  #   the profit
  returns = val[decision_vec, 'return']
  profit = prod(returns+1)*principal - principal
  return(profit)
}
```

A simple baseline

Baseline strategy: \ buy if price went down on previou day; sell if price went up

```r
# baseline strategy:
# buy if price went down on previou day; sell if price went up
decisions = c(train[nrow(train),'return'], val[1:nrow(val)-1, 'return'])
decisions = decisions >0

BackTest(decisions)
```

```
## [1] -3.943526
```

```r
# -3.943526
```

We will lose $3 if follow the baseline strategy.

Predict profitable days using recurrent NN

Prepare data

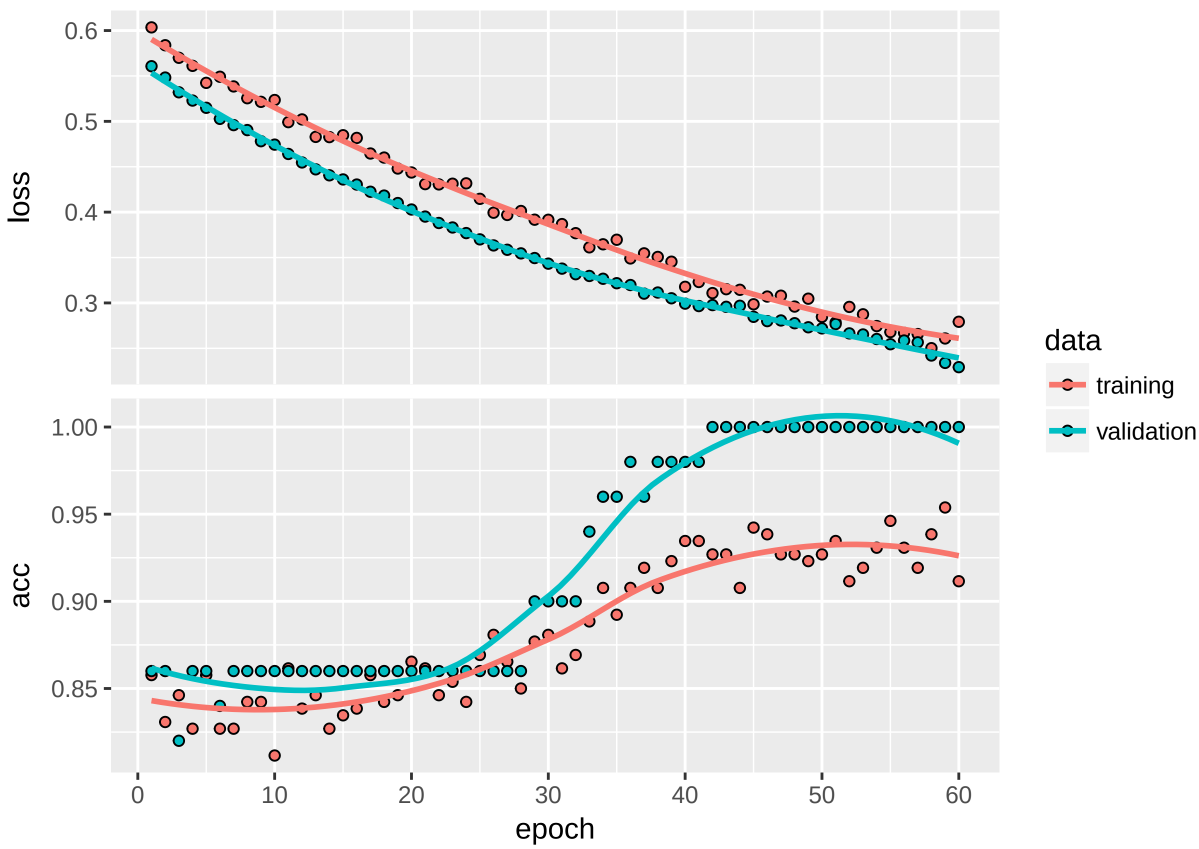
```r
y = daily.df$return>0
#one-hot-encoding
y_one_hot = to_categorical(y)

# normalise x
X = daily.df %>%
  select(-c(date, month, day_of_mon)) %>%
  scale()
X_array_expanded = array(0, dim = c(nrow(X), ncol(X), 1))

# create a test/validation set
X_array_expanded[,,1] = X

ndata = nrow(X)
n_train = as.integer(nrow(daily.df[daily.df$month<=10,]))
X_train = X_array_expanded[1:n_train,,]
dim(X_train) = c(n_train, ncol(X_train), 1)
X_valid = X_array_expanded[(n_train+1):ndata,,]
dim(X_valid) = c(ndata - n_train, ncol(X_train), 1)
```

```r
# RNN model
input_X = layer_input(shape = c(ncol(X),1))
output_GRU_basic = input_X %>%
  layer_gru(units=16, return_sequences = F,
            dropout=0.1) %>%
  layer_dense(units = 6, activation = "elu")  %>%
  layer_dense(units = 2, activation = "softmax")
model_basic = keras_model(inputs = input_X,
                          outputs = output_GRU_basic)

model_basic %>% summary()
```

```
## ___________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## ===========================================================================
## input_1 (InputLayer)             (None, 3, 1)                  0
```

```
## ___________________________________________________________________________
## gru_1 (GRU)                        (None, 16)                    864
## ___________________________________________________________________________
## dense_1 (Dense)                    (None, 6)                     102
## ___________________________________________________________________________
## dense_2 (Dense)                    (None, 2)                     14
## ===========================================================================
## Total params: 980
## Trainable params: 980
## Non-trainable params: 0
## ___________________________________________________________________________
```

```r
model_basic %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)

hist <- model_basic %>% fit(x=X_train,
                    y=y_one_hot[1:n_train,],
                    epochs = 60,
                    batch_size = 128,
                    validation_data = list(X_valid,
                                           y_one_hot[(n_train+1):ndata,]))
plot(hist)
```




```r
#make predictions
library(ramify)  #for argmax
```

```r
pred_proba = model_basic %>% predict(X_valid)
pred_class = argmax(pred_proba)

decisions <- pred_class - 1
BackTest(decisions)
```

```
## [1] 22.53114
```

We are making profit now. However, in the context of currency trading, false negatives are more detrimental to us, we shall adjust the threshold to make decision.

```r
decisions.adjusted <- pred_proba[,2]-pred_proba[,1] >0.3
BackTest(decisions.adjusted)
```

```
## [1] 92.2777
```

We are now making 9.2277705 % return.