mlp1_call

MLP1 Call Options

```r
library(keras)
library(tensorflow)
library(tidyverse)
library(dplyr)
```

```r
n_units <- 400
layers <- 4
n_batch <- 4096
n_epochs <- 10
```

Data Preparation

```
# preliminary data cleaning
options_both <- read.csv("msft_final_df2.csv")
options_both$treasury_rate <- as.numeric(options_both$treasury_rate)
options_both <- na.omit(options_both)
options_both$strike_price <- options_both$strike_price / 1000
options_both <- options_both[, c("date", "exdate", "cp_flag", "strike_price", "best_bid", "best_
offer", "volume", "open_interest", "impl_volatility", "date_ndiff", "treasury_rate", "closing_pr
ice", "sigma_20")]

options_both <- options_both[, c("cp_flag", "strike_price", "best_bid", "best_offer", "date_ndif
f", "treasury_rate", "closing_price", "sigma_20")]

call_op <- options_both %>% filter(cp_flag == "C")
call_op[, 1] <- NULL

# separating input and output dataframes
# input matrix includes all columns except best_bid and best_offer
call_op_ver1 <- select(call_op, -c("best_bid", "best_offer"))

# output vector is the average of bid and ask, which is taken to be the equilibrium price of an
 option
call_op_ver2 <- as.numeric((call_op$best_bid + call_op$best_offer) / 2)

# creating data partitions for training and testing (training: 99%, testing: 1%)
set.seed(42) # equivalent to python's random_state parameter
test_inds <- sample(1:length(call_op_ver2), ceiling(length(call_op_ver2) * 0.99))

call_x_train <- call_op_ver1[test_inds, ]
call_x_test <- call_op_ver1[-test_inds, ]
call_x_train <- as.matrix(call_x_train)
call_x_test <- as.matrix(call_x_test)

call_y_train <- call_op_ver2[test_inds]
call_y_test <- call_op_ver2[-test_inds]
```

Creating the Keras Model

```r
model_call <- keras_model_sequential()
model_call %>% layer_dense(units = n_units, input_shape = c(dim(call_x_train)[2])) %>% layer_act
ivation_leaky_relu()

for (i in 1:(layers-1)){
  model_call <- model_call %>% layer_dense(units = n_units)
  model_call <- model_call %>% layer_batch_normalization()
  model_call <- model_call %>% layer_activation_leaky_relu()
}

model_call %>% layer_dense(units = 1, activation = 'relu')

compile(object = model_call, optimizer = optimizer_adam(), loss = 'mse')

summary(model_call)
```

```
## Model: "sequential"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## dense (Dense)                       (None, 400)                     2400
## ________________________________________________________________________________
## leaky_re_lu (LeakyReLU)             (None, 400)                     0
## ________________________________________________________________________________
## dense_1 (Dense)                     (None, 400)                     160400
## ________________________________________________________________________________
## batch_normalization (BatchNormaliza (None, 400)                     1600
## ________________________________________________________________________________
## leaky_re_lu_1 (LeakyReLU)           (None, 400)                     0
## ________________________________________________________________________________
## dense_2 (Dense)                     (None, 400)                     160400
## ________________________________________________________________________________
## batch_normalization_1 (BatchNormali (None, 400)                     1600
## ________________________________________________________________________________
## leaky_re_lu_2 (LeakyReLU)           (None, 400)                     0
## ________________________________________________________________________________
## dense_3 (Dense)                     (None, 400)                     160400
## ________________________________________________________________________________
## batch_normalization_2 (BatchNormali (None, 400)                     1600
## ________________________________________________________________________________
## leaky_re_lu_3 (LeakyReLU)           (None, 400)                     0
## ________________________________________________________________________________
## dense_4 (Dense)                     (None, 1)                       401
## ================================================================================
## Total params: 488,801
## Trainable params: 486,401
## Non-trainable params: 2,400
## ________________________________________________________________________________
```

```
# learning rate: 0.001
#history <- fit(object=model_call, call_x_train, call_y_train, batch_size = n_batch, epochs = n_
epochs, validation_split = 0.01, callbacks = c(callback_tensorboard()), verbose=1)

# learning rate: 0.0001
#compile(object=model_call, optimizer=optimizer_adam(lr=0.0001), loss='mse')

#history <- fit(object=model_call, call_x_train, call_y_train, batch_size = n_batch, epochs = n_
epochs, validation_split = 0.01, callbacks = c(callback_tensorboard()), verbose=1)

# learning rate: 0.00001
#compile(object=model_call, optimizer = optimizer_adam(lr=0.00001), loss = 'mse')

#history <- fit(object=model_call, call_x_train, call_y_train, batch_size = n_batch, epochs = n_
epochs, validation_split = 0.01, callbacks = c(callback_tensorboard()), verbose=1)

#save_model_hdf5(object=model_call, "C:/Users/robin/Desktop/RStudio/mlp1-call30.h5")

model_call <- load_model_hdf5("mlp1-call30.h5")

call_y_predpred <- predict(object=model_call, call_x_test)

call_y_predpredv2 <- as.numeric(call_y_predpred)

diff <- call_y_predpredv2 - call_y_test

call_mse_final <- mean(diff ** 2)
call_mse_final
```

```
## [1] 0.2456122
```