mlp2_call

MLP2 Call Options

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

```
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_op2 <- options_both %>% filter(cp_flag == "C")
call_op2[, 1] <- NULL

# separating input and output dataframes
# input matrix includes all columns except best_bid and best_offer
call_op2_ver1 <- select(call_op2, -c("best_bid", "best_offer"))

# output matrix includes best_bid and best_offer columns since these are the values we are tryin
g to predict
call_op2_ver2 <- select(call_op2, c("best_bid", "best_offer"))

# Use test_inds from MLP1_call
set.seed(42) # equivalent to python's random_state parameter
test_inds <- sample(1:nrow(call_op2_ver2), ceiling(nrow(call_op2_ver2) * 0.99))

call_x_train2 <- call_op2_ver1[test_inds, ]
call_x_test2 <- call_op2_ver1[-test_inds, ]
call_x_train2 <- as.matrix(call_x_train2)
call_x_test2 <- as.matrix(call_x_test2)

call_y_train2 <- call_op2_ver2[test_inds, ]
call_y_test2 <- call_op2_ver2[-test_inds, ]
call_y_train2 <- as.matrix(call_y_train2)
call_y_test2 <- as.matrix(call_y_test2)
```

Creating Keras Model

```
model2_call <- keras_model_sequential()
model2_call %>% layer_dense(units = n_units, input_shape = c(dim(call_x_train2)[2])) %>% layer_a
ctivation_leaky_relu()

for (i in 1:(layers-1)){
  model2_call <- model2_call %>% layer_dense(units = n_units)
  model2_call <- model2_call %>% layer_batch_normalization()
  model2_call <- model2_call %>% layer_activation_leaky_relu()
}

model2_call %>% layer_dense(units = 2, activation = 'relu')

compile(object = model2_call, optimizer = optimizer_adam(lr = 0.001), loss = 'mse')

summary(model2_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, 2)                       802
## =================================================================================
## Total params: 489,202
## Trainable params: 486,802
## Non-trainable params: 2,400
## _________________________________________________________________________________
```

```
# learning rate: 0.001
#history <- fit(object=model2_call, call_x_train2, call_y_train2, batch_size = n_batch, epochs=3
0, validation_split = 0.01, callbacks = c(callback_tensorboard()), verbose=1)

# learning rate: 0.0001
#compile(object=model2_call, optimizer = optimizer_adam(lr=0.0001), loss='mse')

#history <- fit(object=model2_call, call_x_train2, call_y_train2, batch_size = n_batch, epochs=n
_epochs, validation_split = 0.01, callbacks = c(callback_tensorboard()), verbose=1)

# learning rate: 0.00001
#compile(object=model2_call, optimizer = optimizer_adam(lr=0.00001), loss='mse')

#history <- fit(object=model2_call, call_x_train2, call_y_train2, batch_size = n_batch, epochs=n
_epochs, validation_split = 0.01, callbacks = c(callback_tensorboard()), verbose=1)

# learning rate: 0.000001
#compile(object=model2_call, optimizer = optimizer_adam(lr=0.000001), loss='mse')

#history <- fit(object=model2_call, call_x_train2, call_y_train2, batch_size = n_batch, epochs=n
_epochs, validation_split = 0.01, callbacks = c(callback_tensorboard()), verbose=1)

#save_model_hdf5(object = model2_call, "C:/Users/robin/Desktop/RStudio/mlp2-call60.h5")

model2_call <- load_model_hdf5("mlp2-call60.h5")

call_y2_predpred <- predict(object=model2_call, call_x_test2)

call_y2_predpred2 <- as.data.frame(call_y2_predpred)

mean_call_y2_predpred <- rowMeans(call_y2_predpred2)

call_y2_test2 <- as.data.frame(call_y_test2)

mean_call_y2_test <- rowMeans(call_y2_test2)

eq_mse_call <- mean((mean_call_y2_test - mean_call_y2_predpred) ** 2)
eq_mse_call
```

```
## [1] 0.08773857
```