

# mlp1\_put

## MLP1 Put 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_price", "sigma_20")]

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

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

# separating input and output dataframes
# input matrix includes all columns except best_bid and best_offer
put_op_ver1 <- select(put_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
put_op_ver2 <- as.numeric((put_op$best_bid + put_op$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_put <- sample(1:length(put_op_ver2), ceiling(length(put_op_ver2) * 0.99))

put_x_train <- put_op_ver1[test_inds_put, ]
put_x_test <- put_op_ver1[-test_inds_put, ]
put_x_train <- as.matrix(put_x_train)
put_x_test <- as.matrix(put_x_test)

put_y_train <- put_op_ver2[test_inds_put]
put_y_test <- put_op_ver2[-test_inds_put]

```

## Creating the Keras Model

```
model_put <- keras_model_sequential()
model_put %>% layer_dense(units = n_units, input_shape = c(dim(put_x_train)[2])) %>% layer_activation_leaky_relu()

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

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

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

summary(model_put)
```

```
## 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_put, put_x_train, put_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_put, optimizer=optimizer_adam(lr=0.0001), loss='mse')

#history <- fit(object=model_put, put_x_train, put_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_put, optimizer=optimizer_adam(lr=0.00001), loss='mse')

#history <- fit(object=model_put, put_x_train, put_y_train, batch_size = n_batch, epochs = n_epochs, validation_split = 0.01, callbacks = c(callback_tensorboard()), verbose=1)

#save_model_hdf5(object=model_put, "C:/Users/robin/Desktop/RStudio/mlp1-put30.h5")

model_put <- load_model_hdf5("mlp1-put30.h5")

put_y_predpred <- predict(object=model_put, put_x_test)

put_y_predpred2 <- as.numeric(put_y_predpred)

put_mse_final <- mean((put_y_test - put_y_predpred2) ** 2)
put_mse_final

```

```
## [1] 0.08994421
```