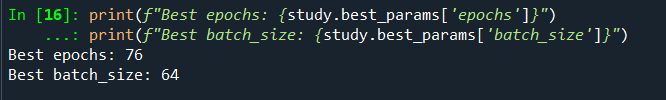
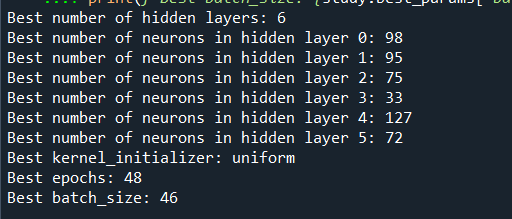
Robert Gorman

Week 11 HW





import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib.dates as dates

import statsmodels.api as sm

plt.rcParams['figure.figsize'] = [20, 15]

sample1 = pd.read\_stata(r"C:\Users\rdg83\OneDrive - Rutgers University\Course Investment Portfolio Management\Week 9 Stuff\finalsample.dta")

sample1.sort\_values(by=['datadate'], inplace=True)

var\_remove = ['PE', 'BM']

sample2 = sample1.drop(var\_remove, axis=1)

sample2['Year']=sample2['datadate'].dt.year

sample2['Month']=sample2['datadate'].dt.month

sample2=sample1[sample1['lagPrice2']>=5]#remove penny stocks

sample2['Year']=sample2['datadate'].dt.year

sample2['Month']=sample2['datadate'].dt.month

#set gvkey and datadate as the index

sample2=sample2.set\_index(['gvkey','datadate'])

#split training and testing samples

Train1=sample2[sample2['Year']<2019]

Test1=sample2[sample2['Year']>=2019]

X\_train=Train1[['lagRet2','loglagVOL2','loglagPrice2', 'loglagMV2','lagShareturnover2','lagRet2\_sic',

'lagRet12','loglagVOL12','lagShareturnover12','lagRet12\_std','lagRet12\_min',

'lagRet12\_max','lagRet12\_sic','epspiq','dvpspq','sale','div\_p', 'cash',

'debt','logatq',

'sp500\_ret\_d','nasdaq\_ret\_d','r2000\_ret\_d','dollar\_ret\_d','VIX',

'yield\_3m','yield\_10y','gdp\_growth','Bull\_ave','Bull\_Bear']]

Y\_train=Train1[['ret']]

X\_test=Test1[['lagRet2','loglagVOL2','loglagPrice2', 'loglagMV2','lagShareturnover2','lagRet2\_sic',

'lagRet12','loglagVOL12','lagShareturnover12','lagRet12\_std','lagRet12\_min',

'lagRet12\_max','lagRet12\_sic','epspiq','dvpspq','sale','div\_p', 'cash',

'debt','logatq',

'sp500\_ret\_d','nasdaq\_ret\_d','r2000\_ret\_d','dollar\_ret\_d','VIX',

'yield\_3m','yield\_10y','gdp\_growth','Bull\_ave','Bull\_Bear']]

Y\_test=Test1[['ret']]

Factor = pd.read\_excel(r"C:\Users\rdg83\OneDrive - Rutgers University\Course Investment Portfolio Management\Week 5 Stuff\Factors-1.xlsx")

rf1 = pd.read\_excel(r"C:\Users\rdg83\OneDrive - Rutgers University\Course Investment Portfolio Management\Week 7 Stuff\Treasury bill.xlsx")

rf1['rf']=rf1['DGS3MO']/1200

rf2=rf1[['Date','rf']].dropna()

rf2['Year']=rf2['Date'].dt.year

rf2['Month']=rf2['Date'].dt.month

rf3=rf2[['Year','Month','rf']].groupby(['Year','Month'], as\_index=False).mean()

indexret1=pd.read\_stata(r"C:\Users\rdg83\OneDrive - Rutgers University\Course Investment Portfolio Management\Week 9 Stuff\Index return-1.dta")

"""PROBLEM #3"""

"""Set the training data from the year 2010 to the year 2015 as the validation dataset and the data

in years <2010 as the new training set for tuning hyperparameters"""

#define a new training and a validation dataset

Train\_new1=sample2[sample2['Year']<2010]

Val\_new1=sample2[(sample2['Year']>=2010)&(sample2['Year']<2016)]

X\_train\_new=Train\_new1[['lagRet2','loglagVOL2','loglagPrice2', 'loglagMV2','lagShareturnover2','lagRet2\_sic',

'lagRet12','loglagVOL12','lagShareturnover12','lagRet12\_std','lagRet12\_min',

'lagRet12\_max','lagRet12\_sic','epspiq','dvpspq','sale','BM','div\_p','PE', 'cash',

'debt','logatq',

'sp500\_ret\_d','nasdaq\_ret\_d','r2000\_ret\_d','dollar\_ret\_d','VIX',

'yield\_3m','yield\_10y','gdp\_growth','Bull\_ave','Bull\_Bear']]

Y\_train\_new=Train\_new1[['ret']]

X\_val=Val\_new1[['lagRet2','loglagVOL2','loglagPrice2', 'loglagMV2','lagShareturnover2','lagRet2\_sic',

'lagRet12','loglagVOL12','lagShareturnover12','lagRet12\_std','lagRet12\_min',

'lagRet12\_max','lagRet12\_sic','epspiq','dvpspq','sale','BM','div\_p','PE', 'cash',

'debt','logatq',

'sp500\_ret\_d','nasdaq\_ret\_d','r2000\_ret\_d','dollar\_ret\_d','VIX',

'yield\_3m','yield\_10y','gdp\_growth','Bull\_ave','Bull\_Bear']]

Y\_val=Val\_new1[['ret']]

"""PROBLEM #4"""

"""Build a neural network with three hidden layers and 50 neurons in each hidden layer. Set

kernel\_initializer=uniform and activation=relu. Use Optuna to tune and search the values of

epochs and batch\_size. Please feel free to choose the number of trials for the search. Report the

values of epochs and batch\_size found by the search"""

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import statsmodels.api as sm

plt.rcParams['figure.figsize'] = [25,20]

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

import optuna

from sklearn.metrics import mean\_squared\_error

def objective(trial):

#neural network model

model = Sequential()

model.add(Dense(50, kernel\_initializer='uniform', activation='relu'))

model.add(Dense(50, kernel\_initializer='uniform', activation='relu'))

model.add(Dense(50, kernel\_initializer='uniform', activation='relu'))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='Adam')

#epochs and batch\_size using Optuna

epochs = trial.suggest\_int("epochs", 10, 100)

batch\_size = trial.suggest\_int("batch\_size", 32, 512)

model.fit(X\_train\_new, Y\_train\_new, epochs=epochs, batch\_size=batch\_size, verbose=0)

score = mean\_squared\_error(Y\_val, model.predict(X\_val))

return score

study = optuna.create\_study(direction='minimize')

study.optimize(objective, n\_trials=10)

print(f"Best epochs: {study.best\_params['epochs']}")

print(f"Best batch\_size: {study.best\_params['batch\_size']}")

"""PROBLEM #5"""

"""Build another neural network and use Optuna to tune and search the values for (1) number of

hidden layers, (2) number of neurons in each hidden layer, (3) kernel\_initializer, (4) epochs, and

(5) batch\_size. Please feel free to choose the number of trials for the search."""

import optuna

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from sklearn.metrics import mean\_squared\_error

def objective(trial):

#tune # of hidden layers

num\_layers = trial.suggest\_int("num\_layers", 2, 6)

#neural network model

model = Sequential()

#hidden layers

for i in range(num\_layers):

#tune # of neurons in each hidden layer

num\_neurons = trial.suggest\_int(f"hidden\_layer\_{i}", 32, 128)

#tune kernel\_initializer

kernel\_initializer = trial.suggest\_categorical("kernel\_initializer", ["uniform", "normal"])

model.add(Dense(num\_neurons, kernel\_initializer=kernel\_initializer, activation="relu"))

model.add(Dense(1))

model.compile(loss="mean\_squared\_error", optimizer="Adam")

#epochs and batch\_size

epochs = trial.suggest\_int("epochs", 10, 100)

batch\_size = trial.suggest\_int("batch\_size", 32, 512)

model.fit(X\_train\_new, Y\_train\_new, epochs=epochs, batch\_size=batch\_size, verbose=0)

score = mean\_squared\_error(Y\_val, model.predict(X\_val))

return score

study = optuna.create\_study(direction="minimize")

study.optimize(objective, n\_trials=15)

print(f"Best number of hidden layers: {study.best\_params['num\_layers']}")

for i in range(study.best\_params['num\_layers']):

print(f"Best number of neurons in hidden layer {i}: {study.best\_params[f'hidden\_layer\_{i}']}")

print(f"Best kernel\_initializer: {study.best\_params['kernel\_initializer']}")

print(f"Best epochs: {study.best\_params['epochs']}")

print(f"Best batch\_size: {study.best\_params['batch\_size']}")

#neural network using best hyperparameters

best\_model = Sequential()

for i in range(study.best\_params['num\_layers']):

best\_model.add(Dense(study.best\_params[f'hidden\_layer\_{i}'], kernel\_initializer=study.best\_params['kernel\_initializer'], activation="relu"))

best\_model.add(Dense(1))

best\_model.compile(loss="mean\_squared\_error", optimizer="Adam")

#train the neural network using best hyperparameters

best\_model.fit(X\_train\_new, Y\_train\_new, epochs=study.best\_params['epochs'], batch\_size=study.best\_params['batch\_size'], verbose=1)

"""Save the trained neural network to your computer.

Load this saved neural network and use it to predict returns based on your testing sample. Report

the average return of the portfolio that consists of the 100 stocks with the highest predicted returns

in each year-month. Also, report the Sharpe ratio of the portfolio."""

import pandas as pd

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.models import load\_model

#save trained neural network

best\_model.save('best\_model.keras')

#load saved neural network

deep\_m\_load = load\_model('best\_model.keras')

#predict returns based on the testing sample

Y\_predict = pd.DataFrame(deep\_m\_load.predict(X\_test), columns=['Y\_predict'])

Y\_test1 = pd.DataFrame(Y\_test).reset\_index()

Comb1 = pd.merge(Y\_test1, Y\_predict, left\_index=True, right\_index=True, how='inner')

Comb1['Year'] = Comb1['datadate'].dt.year

Comb1['Month'] = Comb1['datadate'].dt.month

#rank stocks by predicte returns

rank1 = Comb1[['Y\_predict', 'Year', 'Month']].groupby(['Year', 'Month'], as\_index=False).rank(ascending=False)

rank1.rename(columns={'Y\_predict': 'Y\_predict\_rank'}, inplace=True)

stock\_long1 = pd.merge(Comb1, rank1, left\_index=True, right\_index=True)

stock\_long2 = stock\_long1[stock\_long1['Y\_predict\_rank'] <= 100]

#calculate real returns

stock\_long3 = stock\_long2[['ret', 'Year', 'Month']].groupby(['Year', 'Month']).mean()

#merge with risk-free rate and index return

stock\_long4 = pd.merge(stock\_long3, rf3, left\_on=['Year', 'Month'], right\_on=['Year', 'Month'], how='left')

stock\_long5 = pd.merge(stock\_long4, indexret1, left\_on=['Year', 'Month'], right\_on=['Year', 'Month'], how='left')

#excess returns

stock\_long5['ret\_rf'] = stock\_long5['ret'] - stock\_long5['rf']

stock\_long5['ret\_sp500'] = stock\_long5['ret'] - stock\_long5['sp500\_ret\_m']

#average return of the portfolio

avg\_return = stock\_long5['ret'].mean()

print(f"The average return of the portfolio is: {avg\_return:.4f}")

#Sharpe ratio of the portfolio

Ret\_rf = stock\_long5[['ret\_rf']]

SR = (Ret\_rf.mean()[0] / Ret\_rf.std()[0]) \* np.sqrt(12)

print(f"Sharpe ratio of the portfolio: {SR:.4f}")