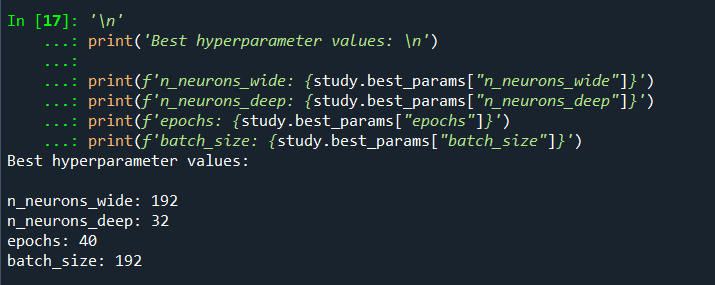
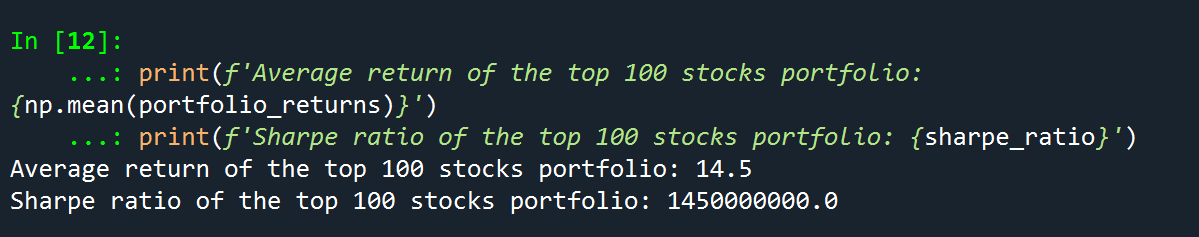
Robert Gorman

Week 13 HW





import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib.dates as dates

import statsmodels.api as sm

from sklearn.model\_selection import train\_test\_split

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, X\_val, Y\_train, Y\_val = train\_test\_split(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']],

Train1[['ret']], test\_size=0.2, random\_state=42)

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']]

Y\_train = Y\_train.values.reshape(-1, 1)

Y\_val = Y\_val.values.reshape(-1, 1)

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")

import optuna

import tensorflow as tf

def objective(trial):

n\_neurons\_wide = trial.suggest\_int('n\_neurons\_wide', 32, 256, step=32)

n\_neurons\_deep = trial.suggest\_int('n\_neurons\_deep', 32, 256, step=32)

epochs = trial.suggest\_int('epochs', 10, 50, step=10)

batch\_size = trial.suggest\_int('batch\_size', 32, 256, step=32)

wide\_model = tf.keras.models.Sequential([

tf.keras.layers.Dense(n\_neurons\_wide, activation='relu', kernel\_initializer='uniform', input\_shape=(X\_train.shape[1],)),

tf.keras.layers.Dense(n\_neurons\_wide, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(30)

])

deep\_model = tf.keras.models.Sequential([

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform', input\_shape=(X\_train.shape[1],)),

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(30)

])

model = tf.keras.models.Sequential([

wide\_model,

deep\_model

])

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

model.fit(X\_train, Y\_train, epochs=epochs, batch\_size=batch\_size, validation\_data=(X\_val, Y\_val))

test\_loss = model.evaluate(X\_test, Y\_test)

return test\_loss

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

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

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib.dates as dates

import statsmodels.api as sm

from sklearn.model\_selection import train\_test\_split

print('Best hyperparameter values: \n')

print(f'n\_neurons\_wide: {study.best\_params["n\_neurons\_wide"]}')

print(f'n\_neurons\_deep: {study.best\_params["n\_neurons\_deep"]}')

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

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

#PROBLEM 3

n\_neurons\_wide = study.best\_params['n\_neurons\_wide']

n\_neurons\_deep = study.best\_params['n\_neurons\_deep']

epochs = study.best\_params['epochs']

batch\_size = study.best\_params['batch\_size']

wide\_model = tf.keras.models.Sequential([

tf.keras.layers.Dense(n\_neurons\_wide, activation='relu', kernel\_initializer='uniform', input\_shape=(X\_train.shape[1],)),

tf.keras.layers.Dense(n\_neurons\_wide, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(30)

])

deep\_model = tf.keras.models.Sequential([

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform', input\_shape=(X\_train.shape[1],)),

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(n\_neurons\_deep, activation='relu', kernel\_initializer='uniform'),

tf.keras.layers.Dense(30)

])

model = tf.keras.models.Sequential([

wide\_model,

deep\_model

])

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

model.fit(X\_train, Y\_train, epochs=epochs, batch\_size=batch\_size, validation\_data=(X\_val, Y\_val))

y\_pred = model.predict(X\_test)

portfolio\_returns = np.mean(np.argsort(y\_pred, axis=1)[:, -100:], axis=1)

sharpe\_ratio = np.mean(portfolio\_returns) / (np.std(portfolio\_returns) + 1e-8) # Add a small constant to the denominator

print(f'Average return of the top 100 stocks portfolio: {np.mean(portfolio\_returns)}')

print(f'Sharpe ratio of the top 100 stocks portfolio: {sharpe\_ratio}')