S&P Data

Imports

In [1]:

```
import os
import pathlib
from time import strptime
from time import strftime
from datetime import datetime
from tqdm import tqdm

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

import numpy as np
import pandas as pd
import seaborn as sns
sns.despine()
import PIL
import pickle
```

```
<Figure size 432x288 with 0 Axes>
```

Processing

In [3]:

```
pip install GitPython
```

```
Requirement already satisfied: GitPython in c:\users\kanika\anaconda3
\lib\site-packages (3.1.31)
Requirement already satisfied: gitdb<5,>=4.0.1 in c:\users\kanika\anac
onda3\lib\site-packages (from GitPython) (4.0.10)
Requirement already satisfied: smmap<6,>=3.0.1 in c:\users\kanika\anac
onda3\lib\site-packages (from gitdb<5,>=4.0.1->GitPython) (5.0.0)
Note: you may need to restart the kernel to use updated packages.
```

In [4]:

```
data_dir = 'C:\\Deep-Portfolio-Management-Reinforcement-Learning\\indiv
directory = os.path.join(os.getcwd(), data_dir) # path to the files
files_tags = os.listdir(directory) #these are the different files

#this is here because hidden files are also shown in the list.
for file in files_tags:
    if file[0] == '.':
        files_tags.remove(file)
stock_name = [file.split('_')[0] for file in files_tags]
stocks = [file for file in files_tags]
print(len(stock_name) == len(stocks))
print('There are {} different stocks.'.format(len(stock_name)))
kept_stocks = list()
not_kept_stocks = list()

for s in tqdm(stocks):
    df = pd.read_csv(os.path.join(os.getcwd(), data_dir, s))

    if len(df)!=1259:
        not_kept_stocks.append(s)
    else:
        kept_stocks.append(s)
```

```
True
There are 100 different stocks.

100%|██████████| 100/100 [00:01<00:00, 53.60it/s]
```

In [5]:

```
kept_stock_rl = [kept_stocks[3], kept_stocks[7], kept_stocks[12], kept_
```

In [6]:

```
list_open = list()
list_close = list()
list_high = list()
list_low = list()

for s in tqdm(kept_stock_rl):
    data = pd.read_csv(os.path.join(os.getcwd(), data_dir, s)).fillna('
    data = data[['open', 'close', 'high', 'low']]
    list_open.append(data.open.values)
    list_close.append(data.close.values)
    list_high.append(data.high.values)
    list_low.append(data.low.values)

array_open = np.transpose(np.array(list_open))[:-1]
array_open_of_the_day = np.transpose(np.array(list_open))[1:]
array_close = np.transpose(np.array(list_close))[:-1]
array_high = np.transpose(np.array(list_high))[:-1]
array_low = np.transpose(np.array(list_low))[:-1]
```

```
100%|██████████| 5/5 [00:00<00:00, 79.51it/s]
```

In [7]:

```
X = np.transpose(np.array([array_close/array_open,
                           array_high/array_open,
                           array_low/array_open,
                           array_open_of_the_day/array_open]), axes= (0
X.shape
```

Out[7]:

```
(4, 5, 1258)
```

The first dimension represents the four features: closing price divided by opening price, high price divided by opening price, low price divided by opening price, and opening price of the next day divided by opening price of the current day. The second dimension represents the five stocks that were kept after filtering out stocks with incomplete data. The third dimension represents the 1258 trading days of the five stocks.

In [8]:

```
# creating a directory to save X
if not os.path.exists('./np_data'):
    os.makedirs('./np_data')
```

In [9]:

```
np.save('./np_data/input.npy', X)
```

Crypto Data

In [10]:

```
data_dir1 = 'C:\\Deep-Portfolio-Management-Reinforcement-Learning\\polo
directory1 = os.path.join(os.getcwd(), data_dir1) # path to the files
files_tags1 = os.listdir(directory1) #these are the different files

#this is here because hidden files are also shown in the list.
for file in files_tags1:
    if file[0] == '.':
        files_tags1.remove(file)
stock_name1 = [file.split('_')[0] for file in files_tags1]
stocks1 = [file for file in files_tags1]
print(len(stock_name1) == len(stocks1))
print('There are {} different stocks.'.format(len(stock_name1)))
kept_stocks1 = list()
not_kept_stocks1 = list()

for s in tqdm(stocks1):
    df1 = pd.read_csv(os.path.join(os.getcwd(), data_dir1, s))

    if len(df1)!=1259:
        not_kept_stocks1.append(s)
    else:
        kept_stocks1.append(s)
```

```
True
There are 14 different stocks.

100%|██████████| 14/14 [00:01<00:00, 13.84it/s]
```

In [11]:

```
for s in stocks1:
    df1 = pd.read_csv(os.path.join(os.getcwd(), data_dir1, s))
    print(s, len(df1))
```

```
BTCUSDT.csv 42010
DASHBTC.csv 60103
DOGEBTC.csv 60915
ETCBTC.csv 17032
ETCETH.csv 17031
ETHBTC.csv 33878
ETHUSDT.csv 33876
GNTBTC.csv 7001
GNTETH.csv 7001
LTCBTC.csv 61096
REPBTC.csv 13547
REPETH.csv 13547
XMRBTC.csv 55285
XRPBTC.csv 51113
```

In [12]:

```
kept_stocks1 = ['ETCBTC.csv', 'ETHBTC.csv', 'DOGEBTC.csv', 'ETHUSDT.csv
              'XRPBTC.csv', 'DASHBTC.csv', 'XMRBTC.csv', 'LTCBTC.csv',
len_stocks1 = list()

for s in kept_stocks1:
    df1 = pd.read_csv(os.path.join(os.getcwd(), data_dir1, s))
    len_stocks1.append(len(df1))


min_len = np.min(len_stocks1)
min_len
```

Out[12]:

```
17031
```

In [13]:

```
list_open1 = list()
list_close1 = list()
list_high1 = list()
list_low1 = list()

for s in tqdm(kept_stocks1):
    data1 = pd.read_csv(os.path.join(os.getcwd(), data_dir1, s)).fillna
    data1 = data1[['open', 'close', 'high', 'low']]
    data1 = data.tail(min_len)
    list_open1.append(data.open.values)
    list_close1.append(data.close.values)
    list_high1.append(data.high.values)
    list_low1.append(data.low.values)

array_open1 = np.transpose(np.array(list_open1))[:-1]
array_open_of_the_day1 = np.transpose(np.array(list_open1))[1:]
array_close1 = np.transpose(np.array(list_close1))[:-1]
array_high1 = np.transpose(np.array(list_high1))[:-1]
array_low1 = np.transpose(np.array(list_low1))[:-1]
```

```
100%|██████████| 10/10 [00:00<00:00, 12.83it/s]
```

In [14]:

```
np.transpose(np.array(list_low1)).shape
```

Out[14]:

```
(1259, 10)
```

In [15]:

```
X1 = np.transpose(np.array([
    array_high1/array_open1,
    array_low1/array_open1,
    array_open_of_the_day1/array_open1]), axes=(0, 2, 1))
X1.shape
```

Out[15]:

```
(3, 10, 1258)
```

This output means that the shape of the X1 array is (3, 10, 1258), which corresponds to: 3: The number of features (high/low prices and opening prices of the day) used to represent each stock. 10: The number of stocks included in the kept_stocks_rl list. 1258: The number of data points (days) available for each stock.

In [46]:

```
np.save('./np_data/inputCrypto.npy', X1)
```

In [ ]:

```
```

DPM

In [16]:

```
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in c:\users\kanika\anaconda3
\lib\site-packages (2.12.0)
Requirement already satisfied: tensorflow-intel==2.12.0 in c:\users\ka
nika\anaconda3\lib\site-packages (from tensorflow) (2.12.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\users\kanika
\anaconda3\lib\site-packages (from tensorflow-intel==2.12.0->tensorflo
w) (1.54.0)
Requirement already satisfied: packaging in c:\users\kanika\anaconda3
\lib\site-packages (from tensorflow-intel==2.12.0->tensorflow) (21.3)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
c:\users\kanika\anaconda3\lib\site-packages (from tensorflow-intel==2.
12.0->tensorflow) (0.31.0)
Requirement already satisfied: absl-py>=1.0.0 in c:\users\kanika\anaco
nda3\lib\site-packages (from tensorflow-intel==2.12.0->tensorflow) (1.
4.0)
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in
c:\users\kanika\anaconda3\lib\site-packages (from tensorflow-intel==2.
12.0->tensorflow) (2.12.0)
Requirement already satisfied: jax>=0.3.15 in c:\users\kanika\anaconda
3\lib\site-packages (from tensorflow-intel==2.12.0->tensorflow) (0.4.
9)
Requirement already satisfied: six>=1.12.0 in c:\users\kanika\anaconda
3\lib\site-packages (from tensorflow-intel==2.12.0->tensorflow) (1.16.
0)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\kanika\ana
conda3\lib\site-packages (from tensorflow-intel==2.12.0->tensorflow)
(2.3.0)
Requirement already satisfied: h5py>=2.9.0 in c:\users\kanika\anaconda
3\lib\site-packages (from tensorflow-intel==2.12.0->tensorflow) (3.6.
0)
Requirement already satisfied: numpy<1.24,>=1.22 in c:\users\kanika\an
aconda3\lib\site-packages (from tensorflow-intel==2.12.0->tensorflow)
(1.22.4)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\users\ka
nika\anaconda3\lib\site-packages (from tensorflow-intel==2.12.0->tenso
rflow) (4.1.1)
Requirement already satisfied: libclang>=13.0.0 in c:\users\kanika\ana
conda3\lib\site-packages (from tensorflow-intel==2.12.0->tensorflow)
(16.0.0)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\kanika\an
aconda3\lib\site-packages (from tensorflow-intel==2.12.0->tensorflow)
(3.3.0)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\kanika\an
aconda3\lib\site-packages (from tensorflow-intel==2.12.0->tensorflow)
(1.6.3)
Requirement already satisfied: flatbuffers>=2.0 in c:\users\kanika\ana
conda3\lib\site-packages (from tensorflow-intel==2.12.0->tensorflow)
(23.5.9)
Requirement already satisfied: keras<2.13,>=2.12.0 in c:\users\kanika
\anaconda3\lib\site-packages (from tensorflow-intel==2.12.0->tensorflo
w) (2.12.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.
21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in c:\users\kanika\anaconda3
\lib\site-packages (from tensorflow-intel==2.12.0->tensorflow) (4.23.
0)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in c:\users\kanika
\anaconda3\lib\site-packages (from tensorflow-intel==2.12.0->tensorflo
w) (0.4.0)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\kanika
\anaconda3\lib\site-packages (from tensorflow-intel==2.12.0->tensorflo
w) (0.2.0)
```

```
Requirement already satisfied: tensorboard<2.13,>=2.12 in c:\users\kan
ika\anaconda3\lib\site-packages (from tensorflow-intel==2.12.0->tensor
flow) (2.12.3)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in c:\users\kanika
\anaconda3\lib\site-packages (from tensorflow-intel==2.12.0->tensorflo
w) (1.12.1)
Requirement already satisfied: setuptools in c:\users\kanika\anaconda3
\lib\site-packages (from tensorflow-intel==2.12.0->tensorflow) (61.2.
0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\users\kanika\a
naconda3\lib\site-packages (from astunparse>=1.6.0->tensorflow-intel==
2.12.0->tensorflow) (0.37.1)
Requirement already satisfied: scipy>=1.7 in c:\users\kanika\anaconda3
\lib\site-packages (from jax>=0.3.15->tensorflow-intel==2.12.0->tensor
flow) (1.7.3)
Requirement already satisfied: ml-dtypes>=0.1.0 in c:\users\kanika\ana
conda3\lib\site-packages (from jax>=0.3.15->tensorflow-intel==2.12.0->
tensorflow) (0.1.0)
Requirement already satisfied: requests<3,>=2.21.0 in c:\users\kanika
\anaconda3\lib\site-packages (from tensorboard<2.13,>=2.12->tensorflow
-intel==2.12.0->tensorflow) (2.27.1)
Requirement already satisfied: werkzeug>=1.0.1 in c:\users\kanika\anac
onda3\lib\site-packages (from tensorboard<2.13,>=2.12->tensorflow-inte
l==2.12.0->tensorflow) (2.0.3)
Requirement already satisfied: markdown>=2.6.8 in c:\users\kanika\anac
onda3\lib\site-packages (from tensorboard<2.13,>=2.12->tensorflow-inte
l==2.12.0->tensorflow) (3.3.4)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 i
n c:\users\kanika\anaconda3\lib\site-packages (from tensorboard<2.13,>
=2.12->tensorflow-intel==2.12.0->tensorflow) (0.7.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in c:\users\kanik
a\anaconda3\lib\site-packages (from tensorboard<2.13,>=2.12->tensorflo
w-intel==2.12.0->tensorflow) (2.18.0)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in c:\us
ers\kanika\anaconda3\lib\site-packages (from tensorboard<2.13,>=2.12->
tensorflow-intel==2.12.0->tensorflow) (1.0.0)
Requirement already satisfied: urllib3<2.0 in c:\users\kanika\anaconda
3\lib\site-packages (from google-auth<3,>=1.6.3->tensorboard<2.13,>=2.
12->tensorflow-intel==2.12.0->tensorflow) (1.26.9)
Requirement already satisfied: pyasn1-modules>=0.2.1 in c:\users\kanik
a\anaconda3\lib\site-packages (from google-auth<3,>=1.6.3->tensorboard
<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow) (0.2.8)
Requirement already satisfied: rsa<5,>=3.1.4 in c:\users\kanika\anacon
da3\lib\site-packages (from google-auth<3,>=1.6.3->tensorboard<2.13,>=
2.12->tensorflow-intel==2.12.0->tensorflow) (4.7.2)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in c:\users\kani
ka\anaconda3\lib\site-packages (from google-auth<3,>=1.6.3->tensorboar
d<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow) (4.2.2)
Requirement already satisfied: requests-oauthlib>=0.7.0 in c:\users\ka
nika\anaconda3\lib\site-packages (from google-auth-oauthlib<1.1,>=0.5-
>tensorboard<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow) (1.3.
1)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in c:\users\kanika
\anaconda3\lib\site-packages (from pyasn1-modules>=0.2.1->google-auth<
3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow-intel==2.12.0->tensorfl
ow) (0.4.8)
Requirement already satisfied: idna<4,>=2.5 in c:\users\kanika\anacond
a3\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.1
2->tensorflow-intel==2.12.0->tensorflow) (3.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\k
anika\anaconda3\lib\site-packages (from requests<3,>=2.21.0->tensorboa
rd<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\kanika\a
naconda3\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.1
3,>=2.12->tensorflow-intel==2.12.0->tensorflow) (2022.12.7)
Requirement already satisfied: oauthlib>=3.0.0 in c:\users\kanika\anac
onda3\lib\site-packages (from requests-oauthlib>=0.7.0->google-auth-oa
uthlib<1.1,>=0.5->tensorboard<2.13,>=2.12->tensorflow-intel==2.12.0->t
ensorflow) (3.2.2)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\ka
nika\anaconda3\lib\site-packages (from packaging->tensorflow-intel==2.
12.0->tensorflow) (3.0.4)
```

In [2]:

```
pip install ffn
```

```
Requirement already satisfied: ffn in c:\users\kanika\anaconda3\lib\si
te-packages (0.3.6)
Requirement already satisfied: future>=0.15 in c:\users\kanika\anacond
a3\lib\site-packages (from ffn) (0.18.2)
Requirement already satisfied: decorator>=4 in c:\users\kanika\anacond
a3\lib\site-packages (from ffn) (5.1.1)
Requirement already satisfied: scikit-learn>=0.15 in c:\users\kanika\a
naconda3\lib\site-packages (from ffn) (1.0.2)
Requirement already satisfied: pandas>=0.19 in c:\users\kanika\anacond
a3\lib\site-packages (from ffn) (1.4.2)
Requirement already satisfied: numpy>=1.5 in c:\users\kanika\anaconda3
\lib\site-packages (from ffn) (1.22.4)
Requirement already satisfied: matplotlib>=1 in c:\users\kanika\anacon
da3\lib\site-packages (from ffn) (3.5.1)
Requirement already satisfied: scipy>=0.15 in c:\users\kanika\anaconda
3\lib\site-packages (from ffn) (1.7.3)
Requirement already satisfied: pandas-datareader>=0.2 in c:\users\kani
ka\anaconda3\lib\site-packages (from ffn) (0.10.0)
Requirement already satisfied: tabulate>=0.7.5 in c:\users\kanika\anac
onda3\lib\site-packages (from ffn) (0.8.9)
Requirement already satisfied: pillow>=6.2.0 in c:\users\kanika\anacon
da3\lib\site-packages (from matplotlib>=1->ffn) (9.0.1)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\kanika
\anaconda3\lib\site-packages (from matplotlib>=1->ffn) (2.8.2)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\kanika\ana
conda3\lib\site-packages (from matplotlib>=1->ffn) (3.0.4)
Requirement already satisfied: cycler>=0.10 in c:\users\kanika\anacond
a3\lib\site-packages (from matplotlib>=1->ffn) (0.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\kanika\an
aconda3\lib\site-packages (from matplotlib>=1->ffn) (1.3.2)
Requirement already satisfied: packaging>=20.0 in c:\users\kanika\anac
onda3\lib\site-packages (from matplotlib>=1->ffn) (21.3)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\kanika\an
aconda3\lib\site-packages (from matplotlib>=1->ffn) (4.25.0)
Requirement already satisfied: pytz>=2020.1 in c:\users\kanika\anacond
a3\lib\site-packages (from pandas>=0.19->ffn) (2022.7.1)
Requirement already satisfied: lxml in c:\users\kanika\anaconda3\lib\s
ite-packages (from pandas-datareader>=0.2->ffn) (4.9.2)
Requirement already satisfied: requests>=2.19.0 in c:\users\kanika\ana
conda3\lib\site-packages (from pandas-datareader>=0.2->ffn) (2.27.1)
Requirement already satisfied: six>=1.5 in c:\users\kanika\anaconda3\l
ib\site-packages (from python-dateutil>=2.7->matplotlib>=1->ffn) (1.1
6.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\kanika\a
naconda3\lib\site-packages (from requests>=2.19.0->pandas-datareader>=
0.2->ffn) (2022.12.7)
Requirement already satisfied: idna<4,>=2.5 in c:\users\kanika\anacond
a3\lib\site-packages (from requests>=2.19.0->pandas-datareader>=0.2->f
fn) (3.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\k
anika\anaconda3\lib\site-packages (from requests>=2.19.0->pandas-datar
eader>=0.2->ffn) (2.0.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\kanik
a\anaconda3\lib\site-packages (from requests>=2.19.0->pandas-datareade
r>=0.2->ffn) (1.26.9)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\kanika
\anaconda3\lib\site-packages (from scikit-learn>=0.15->ffn) (2.2.0)
Requirement already satisfied: joblib>=0.11 in c:\users\kanika\anacond
a3\lib\site-packages (from scikit-learn>=0.15->ffn) (1.1.0)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:

```
pip install environment
```

```
Requirement already satisfied: environment in c:\users\kanika\anaconda
3\lib\site-packages (1.0.0)
Note: you may need to restart the kernel to use updated packages.
```

In [17]:

```
import tensorflow as tf
import numpy as np
from collections import deque
import random
import pandas as pd
import ffn
from environment import *

%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt

from tqdm import tqdm
```

Parameters

In [20]:

```
print(os.path.exists('./np_data/inputCrypto.npy'))
```

```
True
```

In [19]:

```
path_data = './np_data/inputCrypto.npy'
data_type = path_data.split('/')[2][5:].split('.')[0]
namesBio=['JNJ','PFE','AMGN','MDT','CELG','LLY']
namesUtilities=['XOM','CVX','MRK','SLB','MMM']
namesTech=['FB','AMZN','MSFT','AAPL','T','VZ','CMCSA','IBM','CRM','INTC
namesCrypto = ['ETCBTC', 'ETHBTC', 'DOGEBTC', 'ETHUSDT', 'BTCUSDT', 'XR

if data_type == 'Utilities':
    list_stock = namesUtilities
elif data_type == 'Bio':
    list_stock = namesBio
elif data_type == 'Tech':
    list_stock = namesTech
elif data_type == 'Crypto':
    list_stock = namesCrypto
else:
    list_stock = [i for i in range(m)]
```

In [21]:

```
# determine the length of the data, #features, #stocks
data = np.load(path_data)
trading_period = data.shape[2]
nb_feature_map = data.shape[0]
nb_stocks = data.shape[1]

# fix parameters of the network
m = nb_stocks
```

In [22]:

```
###############################dictionaries of the problem############
dict_hp_net = {'n_filter_1': 2, 'n_filter_2': 20, 'kernel1_size':(1, 3)
dict_hp_pb = {'batch_size': 50, 'ratio_train': 0.6,'ratio_val': 0.2, 'l
              'ratio_greedy':0.8, 'ratio_regul': 0.1}
dict_hp_opt = {'regularization': 1e-8, 'learning': 9e-2}
dict_fin = {'trading_cost': 0.25/100, 'interest_rate': 0.02/250, 'cash_
dict_train = {'pf_init_train': 10000, 'w_init_train': 'd', 'n_episodes'
dict_test = {'pf_init_test': 10000, 'w_init_test': 'd'}


###############################HP of the network #####################
n_filter_1 = dict_hp_net['n_filter_1']
n_filter_2 = dict_hp_net['n_filter_2']
kernel1_size = dict_hp_net['kernel1_size']
```

In [23]:

```
###############################HP of the problem######################

# Size of mini-batch during training
batch_size = dict_hp_pb['batch_size']
# Total number of steps for pre-training in the training set
total_steps_train = int(dict_hp_pb['ratio_train']*trading_period)

# Total number of steps for pre-training in the validation set
total_steps_val = int(dict_hp_pb['ratio_val']*trading_period)

# Total number of steps for the test
total_steps_test = trading_period-total_steps_train-total_steps_val

# Number of the columns (number of the trading periods) in each input p
n = dict_hp_pb['length_tensor']

ratio_greedy = dict_hp_pb['ratio_greedy']

ratio_regul = dict_hp_pb['ratio_regul']
```

In [24]:

```
###############################HP of the optimization#################


# The L2 regularization coefficient applied to network training
regularization = dict_hp_opt['regularization']
# Parameter alpha (i.e. the step size) of the Adam optimization
learning = dict_hp_opt['learning']

optimizer = tf.keras.optimizers.Adam(learning)
```

In [25]:

```
##############################Finance parameters######################

trading_cost= dict_fin['trading_cost']
interest_rate= dict_fin['interest_rate']
cash_bias_init = dict_fin['cash_bias_init']

############################## PVM Parameters #########################
sample_bias = 5e-5  # Beta in the geometric distribution for online tra
```

In [26]:

```
############################## Training Parameters ####################

w_init_train = np.array(np.array([1]+[0]*m))#dict_train['w_init_train']

pf_init_train = dict_train['pf_init_train']

n_episodes = dict_train['n_episodes']
n_batches = dict_train['n_batches']

############################## Test Parameters ########################

w_init_test = np.array(np.array([1]+[0]*m))#dict_test['w_init_test']

pf_init_test = dict_test['pf_init_test']


############################## other environment Parameters ##########

w_eq = np.array(np.array([1/(m+1)]*(m+1)))

w_s = np.array(np.array([1]+[0.0]*m))
```

In [27]:

```
#random action function

def get_random_action(m):
    random_vec = np.random.rand(m+1)
    return random_vec/np.sum(random_vec)
```

Environment

In [21]:

```
pip install gym
```

```
Requirement already satisfied: gym in c:\users\kanika\anaconda3\lib\si
te-packages (0.26.2)
Requirement already satisfied: gym-notices>=0.0.4 in c:\users\kanika\a
naconda3\lib\site-packages (from gym) (0.0.8)
Requirement already satisfied: numpy>=1.18.0 in c:\users\kanika\anacon
da3\lib\site-packages (from gym) (1.22.4)
Requirement already satisfied: cloudpickle>=1.2.0 in c:\users\kanika\a
naconda3\lib\site-packages (from gym) (2.0.0)
Requirement already satisfied: importlib-metadata>=4.8.0 in c:\users\k
anika\anaconda3\lib\site-packages (from gym) (4.11.3)
Requirement already satisfied: zipp>=0.5 in c:\users\kanika\anaconda3
\lib\site-packages (from importlib-metadata>=4.8.0->gym) (3.7.0)
Note: you may need to restart the kernel to use updated packages.
```

In [28]:

```
import math
import gym
from gym import spaces, logger
from gym.utils import seeding
import numpy as np
from gym.envs.registration import register

class TradeEnv():
    def __init__(self, path = './np_data/input.npy', window_length=50,
                 portfolio_value= 10000, trading_cost= 0.25/100,interes
        #path to numpy data
        self.path = path
        #load the whole data
        self.data = np.load(self.path)


        #parameters
        self.portfolio_value = portfolio_value
        self.window_length=window_length
        self.trading_cost = trading_cost
        self.interest_rate = interest_rate

        #number of stocks and features
        self.nb_stocks = self.data.shape[1]
        self.nb_features = self.data.shape[0]
        self.end_train = int((self.data.shape[2]-self.window_length)*tr

        #init state and index
        self.index = None
        self.state = None
        self.done = False

        #init seed
        self.seed()

    def return_pf(self): # returns the value of the portfolio
        return self.portfolio_value

    def readTensor(self,X,t):
        return X[ : , :, t-self.window_length:t ]

    def readUpdate(self, t): #returns the return of each stock for the
        return np.array([1+self.interest_rate]+self.data[-1,:,t].tolist

    def seed(self, seed=None):
        self.np_random, seed = seeding.np_random(seed)
        return [seed]

    def reset(self, w_init, p_init, t=0 ):
        #This function restarts the environment with given initial weig
        self.state= (self.readTensor(self.data, self.window_length) , w
        self.index = self.window_length + t
        self.done = False

        return self.state, self.done

    def step(self, action):
        """
        This function is the main part of the render.
        At each step t, the trading agent gives as input the action he
        The function computes the new value of the portfolio at the ste
        The reward is defined as the evolution of the the value of the 
        """

        index = self.index
        #get Xt from data:
        data = self.readTensor(self.data, index)
        done = self.done

        #beginning of the day
        state = self.state
        w_previous = state[1]
        pf_previous = state[2]

        #the update vector is the vector of the opening price of the da
        update_vector = self.readUpdate(index)

        #allocation choice
        w_alloc = action
        pf_alloc = pf_previous

        #Compute transaction cost
        cost = pf_alloc * np.linalg.norm((w_alloc-w_previous),ord = 1)*

        #convert weight vector into value vector
        v_alloc = pf_alloc*w_alloc

        #pay transaction costs
        pf_trans = pf_alloc - cost
        v_trans = v_alloc - np.array([cost]+ [0]*self.nb_stocks)

        #####market prices evolution
        #we go to the end of the day

        #compute new value vector
        v_evol = v_trans*update_vector


        #compute new portfolio value
        pf_evol = np.sum(v_evol)

        #compute weight vector
        w_evol = v_evol/pf_evol


        #compute instanteanous reward
        reward = (pf_evol-pf_previous)/pf_previous

        #update index
        index = index+1

        #compute state

        state = (self.readTensor(self.data, index), w_evol, pf_evol)

        if index >= self.end_train:
            done = True

        self.state = state
        self.index = index
        self.done = done
        return state, reward, done
```

In [29]:

```
#environment for trading of the agent
# this is the agent trading environment (policy network agent)
env = TradeEnv(path=path_data, window_length=n,
               portfolio_value=pf_init_train, trading_cost=trading_cost
               interest_rate=interest_rate, train_size=dict_hp_pb['rati


#environment for equiweighted
#this environment is set up for an agent who only plays an equiweithed
env_eq = TradeEnv(path=path_data, window_length=n,
               portfolio_value=pf_init_train, trading_cost=trading_cost
               interest_rate=interest_rate, train_size=dict_hp_pb['rati

#environment secured (only money)
#this environment is set up for an agentwho plays secure, keeps its mon
env_s = TradeEnv(path=path_data, window_length=n,
               portfolio_value=pf_init_train, trading_cost=trading_cost
               interest_rate=interest_rate, train_size=dict_hp_pb['rati
```

In [30]:

```
#full on one stock environment
#these environments are set up for agents who play only on one stock

action_fu = list()
env_fu = list()


for i in range(m):
    action = np.array([0]*(i+1) + [1] + [0]*(m-(i+1)))
    action_fu.append(action)

    env_fu_i = TradeEnv(path=path_data, window_length=n,
               portfolio_value=pf_init_train, trading_cost=trading_cost
               interest_rate=interest_rate, train_size=dict_hp_pb['rati

    env_fu.append(env_fu_i)
```

Definition of the Actor

In [68]:

```
# define neural net \pi_\phi(s) as a class
class Policy(object):
    '''
    This class is used to instanciate the policy network agent

    '''


    def __init__(self, m, n, sess, optimizer,
                 trading_cost=trading_cost,
                 interest_rate=interest_rate,
                 n_filter_1=n_filter_1,
                 n_filter_2=n_filter_2):

        # parameters
        self.trading_cost = trading_cost
        self.interest_rate = interest_rate
        self.n_filter_1 = n_filter_1
        self.n_filter_2 = n_filter_2
        self.n = n
        self.m = m

        with tf.compat.v1.variable_scope("Inputs"):

            # Placeholder

            # tensor of the prices
            self.X_t = tf.keras.Input(
                dtype=tf.float32, shape=[nb_feature_map, self.m, self.r
            # weights at the previous time step
            self.W_previous = tf.keras.Input(dtype=tf.float32, shape=[N
            # portfolio value at the previous time step
            self.pf_value_previous = tf.keras.Input(dtype=tf.float32, s
            # vector of Open(t+1)/Open(t)
            self.dailyReturn_t = tf.keras.Input(dtype=tf.float32, shape

            #self.pf_value_previous_eq = tf.placeholder(tf.float32, [No



        with tf.compat.v1.variable_scope("Policy_Model"):

            # variable of the cash bias
            bias = tf.compat.v1.get_variable('cash_bias', shape=[
                                   1, 1, 1, 1], initializer=tf.constant
            # shape of the tensor == batchsize
            shape_X_t = tf.shape(self.X_t)[0]
            # trick to get a "tensor size" for the cash bias
            self.cash_bias = tf.tile(bias, tf.stack([shape_X_t, 1, 1, 1
            # print(self.cash_bias.shape)

            with tf.compat.v1.variable_scope("Conv1"):
                # first layer on the X_t tensor
                # return a tensor of depth 2
                self.conv1 = tf.keras.layers.Conv2D(
                    input_shape=tf.transpose(self.X_t, perm=[0, 3, 2, 1
                    activation=tf.nn.relu,
                    filters=self.n_filter_1,
                    strides=(1, 1),
                    kernel_size=kernel1_size,
                    padding='same')
            with tf.compat.v1.variable_scope("Conv2"):

                #feature maps
                self.conv2 = tf.keras.layers.Conv2D(
                    inputs=self.conv1,
                    activation=tf.nn.relu,
                    filters=self.n_filter_2,
                    strides=(self.n, 1),
                    kernel_size=(1, self.n),
                    padding='same')

            with tf.compat.v1.variable_scope("Tensor3"):
                #w from last periods
                # trick to have good dimensions
                w_wo_c = self.W_previous[:, 1:]
                w_wo_c = tf.expand_dims(w_wo_c, 1)
                w_wo_c = tf.expand_dims(w_wo_c, -1)
                self.tensor3 = tf.concat([self.conv2, w_wo_c], axis=3)

            with tf.compat.v1.variable_scope("Conv3"):
                #last feature map WITHOUT cash bias
                self.conv3 = tf.keras.layers.Conv2D(
                    inputs=self.conv2,
                    activation=tf.nn.relu,
                    filters=1,
                    strides=(self.n_filter_2 + 1, 1),
                    kernel_size=(1, 1),
                    padding='same')

            with tf.compat.v1.variable_scope("Tensor4"):
                #last feature map WITH cash bias
                self.tensor4 = tf.concat([self.cash_bias, self.conv3],
                # we squeeze to reduce and get the good dimension
                self.squeezed_tensor4 = tf.squeeze(self.tensor4, [1, 3]

            with tf.compat.v1.variable_scope("Policy_Output"):
                # softmax layer to obtain weights
                self.action = tf.nn.softmax(self.squeezed_tensor4)

            with tf.compat.v1.variable_scope("Reward"):
                # computation of the reward
                #please look at the chronological map to understand
                constant_return = tf.constant(
                    1+self.interest_rate, shape=[1, 1])
                cash_return = tf.tile(
                    constant_return, tf.stack([shape_X_t, 1]))
                y_t = tf.concat(
                    [cash_return, self.dailyReturn_t], axis=1)
                Vprime_t = self.action * self.pf_value_previous
                Vprevious = self.W_previous*self.pf_value_previous

                # this is just a trick to get the good shape for cost
                constant = tf.constant(1.0, shape=[1])

                cost = self.trading_cost * \
                    tf.norm(Vprime_t-Vprevious, ord=1, axis=1)*constant

                cost = tf.expand_dims(cost, 1)

                zero = tf.constant(
                    np.array([0.0]*m).reshape(1, m), shape=[1, m], dtyp
                vec_zero = tf.tile(zero, tf.stack([shape_X_t, 1]))
                vec_cost = tf.concat([cost, vec_zero], axis=1)

                Vsecond_t = Vprime_t - vec_cost

                V_t = tf.multiply(Vsecond_t, y_t)
                self.portfolioValue = tf.norm(V_t, ord=1)
                self.instantaneous_reward = (
                    self.portfolioValue-self.pf_value_previous)/self.pf


            with tf.compat.v1.variable_scope("Reward_Equiweighted"):
                constant_return = tf.constant(
                    1+self.interest_rate, shape=[1, 1])
                cash_return = tf.tile(
                    constant_return, tf.stack([shape_X_t, 1]))
                y_t = tf.concat(
                    [cash_return, self.dailyReturn_t], axis=1)


                V_eq = w_eq*self.pf_value_previous
                V_eq_second = tf.multiply(V_eq, y_t)

                self.portfolioValue_eq = tf.norm(V_eq_second, ord=1)

                self.instantaneous_reward_eq = (
                    self.portfolioValue_eq-self.pf_value_previous)/self

            with tf.compat.v1.variable_scope("Max_weight"):
                self.max_weight = tf.reduce_max(self.action)
                print(self.max_weight.shape)


            with tf.compat.v1.variable_scope("Reward_adjusted"):

                self.adjested_reward = self.instantaneous_reward - self

        #objective function
        #maximize reward over the batch
        # min(-r) = max(r)
        self.train_op = optimizer.minimize(-self.adjested_reward)

        # some bookkeeping
        self.optimizer = optimizer
        self.sess = sess

    def compute_W(self, X_t_, W_previous_):
        """
        This function returns the action the agent takes
        given the input tensor and the W_previous

        It is a vector of weight

        """

        return self.sess.run(tf.squeeze(self.action), feed_dict={self.X

    def train(self, X_t_, W_previous_, pf_value_previous_, dailyReturn_
        """
        This function trains the neural network
        maximizing the reward
        the input is a batch of the differents values
        """
        self.sess.run(self.train_op, feed_dict={self.X_t: X_t_,
                                                self.W_previous: W_prev
                                                self.pf_value_previous:
                                                self.dailyReturn_t: dai
```

Definition of the PVM Class

In [37]:

```
class PVM(object):
    '''
    This is the memory stack called PVM in the paper
    '''

    def __init__(self, m, sample_bias, total_steps = total_steps_train,
                 batch_size = batch_size, w_init = w_init_train):


        #initialization of the memory
        #we have a total_step_times the initialization portfolio tensor
        self.memory = np.transpose(np.array([w_init]*total_steps))
        self.sample_bias = sample_bias
        self.total_steps = total_steps
        self.batch_size = batch_size

    def get_W(self, t):
        #return the weight from the PVM at time t
        return self.memory[:, t]

    def update(self, t, w):
        #update the weight at time t
        self.memory[:, t] = w


    def draw(self, beta=sample_bias):
        '''
        returns a valid step so you can get a training batch starting a
        '''
        while 1:
            z = np.random.geometric(p=beta)
            tb = self.total_steps - self.batch_size + 1 - z
            if tb >= 0:
                return tb

    def test(self):
        #just to test
        return self.memory
```

In [38]:

```python
def get_max_draw_down(xs):
    xs = np.array(xs)
    i = np.argmax(np.maximum.accumulate(xs) - xs) # end of the period
    j = np.argmax(xs[:i]) # start of period

    return xs[j] - xs[i]
```

In [39]:

```
def eval_perf(e):
    """
    This function evaluates the performance of the different types of a


    """
    list_weight_end_val = list()
    list_pf_end_training = list()
    list_pf_min_training = list()
    list_pf_max_training = list()
    list_pf_mean_training = list()
    list_pf_dd_training = list()

    #######TEST#######
    #environment for trading of the agent
    env_eval = TradeEnv(path=path_data, window_length=n,
                   portfolio_value=pf_init_train, trading_cost=trading_
                   interest_rate=interest_rate, train_size=dict_hp_pb['



    #initialization of the environment
    state_eval, done_eval = env_eval.reset(w_init_test, pf_init_test, t



    #first element of the weight and portfolio value
    p_list_eval = [pf_init_test]
    w_list_eval = [w_init_test]

    for k in range(total_steps_train, total_steps_train +total_steps_va
        X_t = state_eval[0].reshape([-1]+ list(state_eval[0].shape))
        W_previous = state_eval[1].reshape([-1]+ list(state_eval[1].sha
        pf_value_previous = state_eval[2]
        #compute the action
        action = actor.compute_W(X_t, W_previous)
        #step forward environment
        state_eval, reward_eval, done_eval = env_eval.step(action)

        X_next = state_eval[0]
        W_t_eval = state_eval[1]
        pf_value_t_eval = state_eval[2]

        dailyReturn_t = X_next[-1, :, -1]
        #print('current portfolio value', round(pf_value_previous,0))
        #print('weights', W_previous)
        p_list_eval.append(pf_value_t_eval)
        w_list_eval.append(W_t_eval)

    list_weight_end_val.append(w_list_eval[-1])
    list_pf_end_training.append(p_list_eval[-1])
    list_pf_min_training.append(np.min(p_list_eval))
    list_pf_max_training.append(np.max(p_list_eval))
    list_pf_mean_training.append(np.mean(p_list_eval))

    list_pf_dd_training.append(get_max_draw_down(p_list_eval))

    print('End of test PF value:',round(p_list_eval[-1]))
    print('Min of test PF value:',round(np.min(p_list_eval)))
    print('Max of test PF value:',round(np.max(p_list_eval)))
    print('Mean of test PF value:',round(np.mean(p_list_eval)))
    print('Max Draw Down of test PF value:',round(get_max_draw_down(p_l
    print('End of test weights:',w_list_eval[-1])
    plt.title('Portfolio evolution (validation set) episode {}'.format(
    plt.plot(p_list_eval, label = 'Agent Portfolio Value')
    plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
    plt.show()
    plt.title('Portfolio weights (end of validation set) episode {}'.fo
    plt.bar(np.arange(m+1), list_weight_end_val[-1])
    plt.xticks(np.arange(m+1), ['Money'] + list_stock, rotation=45)
    plt.show()


    names = ['Money'] + list_stock
    w_list_eval = np.array(w_list_eval)
    for j in range(m+1):
        plt.plot(w_list_eval[:,j], label = 'Weight Stock {}'.format(nam
        plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.5)
    plt.show()
```

RL Algorithm

In [ ]:

```
############# TRAINING #####################
###########################################
tf.compat.v1.reset_default_graph()

# sess
sess = tf.compat.v1.Session()

# initialize networks
actor = Policy(m, n, sess, optimizer,
                 trading_cost=trading_cost,
                 interest_rate=interest_rate)  # policy initialization

# initialize tensorflow graphs
sess.run(tf.global_variables_initializer())



list_final_pf = list()
list_final_pf_eq = list()
list_final_pf_s = list()

list_final_pf_fu = list()
state_fu = [0]*m
done_fu = [0]*m

pf_value_t_fu = [0]*m

for i in range(m):
    list_final_pf_fu.append(list())


###### Train #####
for e in range(n_episodes):
    print('Start Episode', e)
    if e==0:
        eval_perf('Before Training')
    print('Episode:', e)
    #init the PVM with the training parameters
    memory = PVM(m,sample_bias, total_steps = total_steps_train,
                 batch_size = batch_size, w_init = w_init_train)

    for nb in range(n_batches):
        #draw the starting point of the batch
        i_start = memory.draw()


        #reset the environment with the weight from PVM at the starting
        #reset also with a portfolio value with initial portfolio value
        state, done = env.reset(memory.get_W(i_start), pf_init_train, t
        state_eq, done_eq = env_eq.reset(w_eq, pf_init_train, t=i_start
        state_s, done_s = env_s.reset(w_s, pf_init_train, t=i_start )

        for i in range(m):
            state_fu[i], done_fu[i] = env_fu[i].reset(action_fu[i], pf_



        list_X_t, list_W_previous, list_pf_value_previous, list_dailyRe
        list_pf_value_previous_eq, list_pf_value_previous_s = [],[]
        list_pf_value_previous_fu = list()
        for i in range(m):
            list_pf_value_previous_fu.append(list())




        for bs in range(batch_size):

            #load the different inputs from the previous loaded state
            X_t = state[0].reshape([-1] + list(state[0].shape))
            W_previous = state[1].reshape([-1] + list(state[1].shape))
            pf_value_previous = state[2]


            if np.random.rand()< ratio_greedy:
                #print('go')
                #computation of the action of the agent
                action = actor.compute_W(X_t, W_previous)
            else:
                action = get_random_action(m)

            #given the state and the action, call the environment to go
            state, reward, done = env.step(action)
            state_eq, reward_eq, done_eq = env_eq.step(w_eq)
            state_s, reward_s, done_s = env_s.step(w_s)

            for i in range(m):
                state_fu[i], _ , done_fu[i] = env_fu[i].step(action_fu[



            #get the new state
            X_next = state[0]
            W_t = state[1]
            pf_value_t = state[2]

            pf_value_t_eq = state_eq[2]
            pf_value_t_s = state_s[2]

            for i in range(m):
                pf_value_t_fu[i] = state_fu[i][2]


            #let us compute the returns
            dailyReturn_t = X_next[-1, :, -1]
            #update into the PVM
            memory.update(i_start+bs, W_t)
            #store elements
            list_X_t.append(X_t.reshape(state[0].shape))
            list_W_previous.append(W_previous.reshape(state[1].shape))
            list_pf_value_previous.append([pf_value_previous])
            list_dailyReturn_t.append(dailyReturn_t)

            list_pf_value_previous_eq.append(pf_value_t_eq)
            list_pf_value_previous_s.append(pf_value_t_s)

            for i in range(m):
                list_pf_value_previous_fu[i].append(pf_value_t_fu[i])

            if bs==batch_size-1:
                list_final_pf.append(pf_value_t)
                list_final_pf_eq.append(pf_value_t_eq)
                list_final_pf_s.append(pf_value_t_s)
                for i in range(m):
                    list_final_pf_fu[i].append(pf_value_t_fu[i])



#             #printing
#             if bs==0:
#                 print('start', i_start)
#                 print('PF_start', round(pf_value_previous,0))

#             if bs==batch_size-1:
#                 print('PF_end', round(pf_value_t,0))
#                 print('weight', W_t)

        list_X_t = np.array(list_X_t)
        list_W_previous = np.array(list_W_previous)
        list_pf_value_previous = np.array(list_pf_value_previous)
        list_dailyReturn_t = np.array(list_dailyReturn_t)


        #for each batch, train the network to maximize the reward
        actor.train(list_X_t, list_W_previous,
                    list_pf_value_previous, list_dailyReturn_t)
    eval_perf(e)
```

In [ ]:

In [ ]: