DQN_v1

August 15, 2020

0.1 Jason's toy model in Q-learning

Thanks for this useful link:https://www.tensorflow.org/agents/tutorials/1_dqn_tutorial So far, Reinforcement learning algorithms are not well GPU optimized since it's hard to do parallization in updating stages + reward. Thus, the DQN part is CPU only :) Thanks to this useful link : https://adventuresinmachinelearning.com/reinforcement-learning-tensorflow/ Here we didn't consider the Volume

```
In [1]: import numpy as np # linear algebra
        import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
        import matplotlib
        import tensorflow as tf
        from matplotlib import colors as mcolors
        import matplotlib.pyplot as plt
        color_array = list(mcolors.CSS4_COLORS.keys())
        root_path = 'Data/Stocks/'
        #root_path = "/kaggle/input/price-volume-data-for-all-us-stocks-etfs/ETFs/"

        ## Here we got 2 sets of data: Google stock price daily and Google stock price in 5 da

        df = pd.read_csv(root_path + "googl.us.txt")
        df_hf = pd.read_csv("Data/High_frequency_data/GOOG_stock_1minute_sample/GOOG_sample.tx
```

```
In [2]: df.head()
```

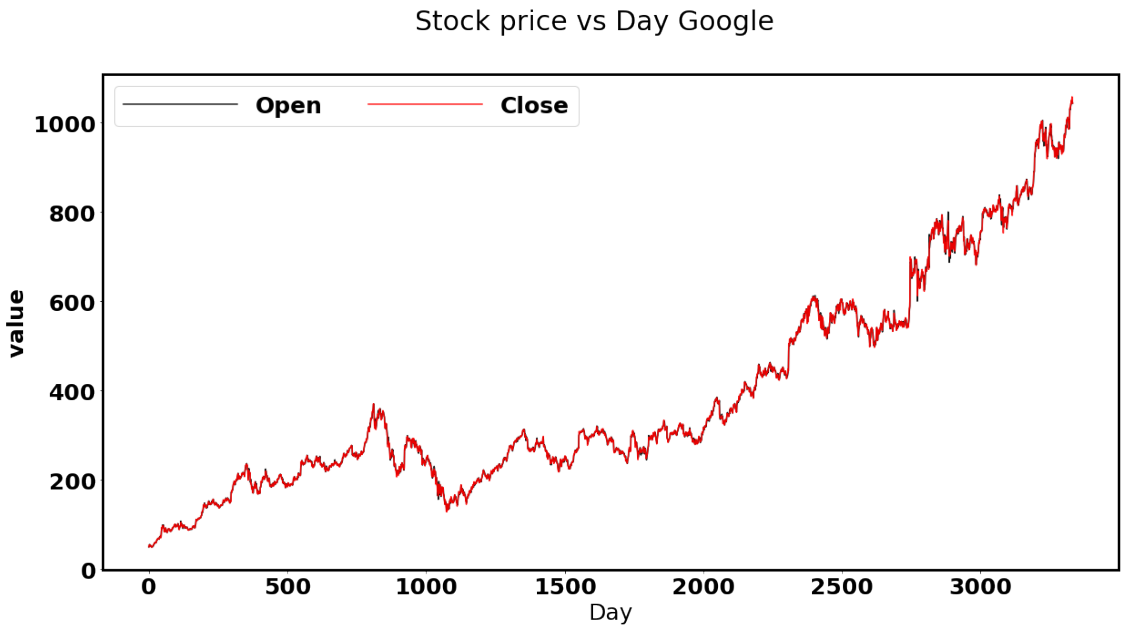
```
Out[2]:          Date    Open   High     Low   Close    Volume  OpenInt
        0  2004-08-19  50.000  52.03  47.980  50.170  44703800        0
        1  2004-08-20  50.505  54.54  50.250  54.155  22857200        0
        2  2004-08-23  55.375  56.74  54.525  54.700  18274400        0
        3  2004-08-24  55.620  55.80  51.785  52.435  15262600        0
        4  2004-08-25  52.480  54.00  51.940  53.000   9197800        0
```

```
In [3]: df_hf.head()
```

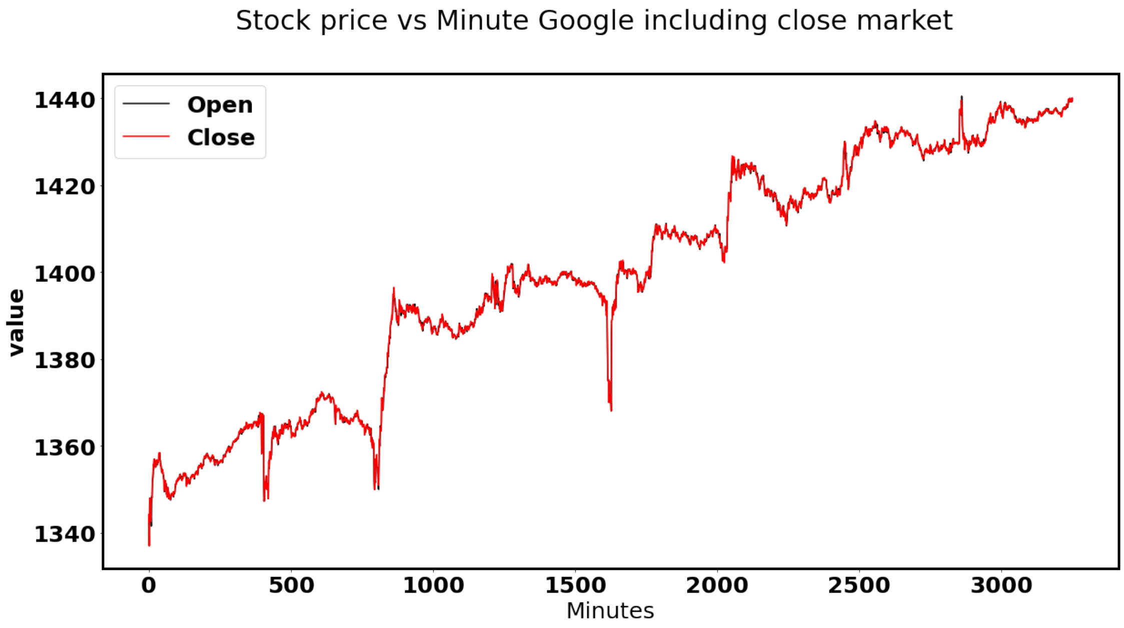
```
Out[3]:               DateTime     Open     High      Low    Close  Volume
        0  2020-01-02 04:00:00  1342.00  1342.20  1342.00  1342.20     424
        1  2020-01-02 04:02:00  1344.20  1344.20  1344.20  1344.20     177
        2  2020-01-02 08:00:00  1337.02  1337.02  1337.02  1337.02     329
        3  2020-01-02 08:09:00  1347.00  1347.00  1347.00  1347.00     155
        4  2020-01-02 08:55:00  1348.00  1348.00  1348.00  1348.00     190
```

```
In [4]: import matplotlib
        from matplotlib.pylab import rc
        font = {'family': 'normal','weight': 'bold',
                'size': 25}

        matplotlib.rc('font', **font)
        rc('axes', linewidth=3)


        plt.subplot(1,1,1)
        plt.plot(df["Open"],"k",label="Open",alpha=0.9)
        plt.plot(df["Close"],"r",label="Close",alpha=0.9)


        plt.xlabel("Day")
        plt.ylabel(r"${\rm value}$")
        plt.suptitle("Stock price vs Day Google")

        fig = matplotlib.pyplot.gcf()

        fig.set_size_inches(20,10)
        plt.legend(fontsize=25,handlelength=5,ncol=3)
        plt.show()
```

```
findfont: Font family ['normal'] not found. Falling back to DejaVu Sans.
findfont: Font family ['normal'] not found. Falling back to DejaVu Sans.
findfont: Font family ['normal'] not found. Falling back to DejaVu Sans.
```



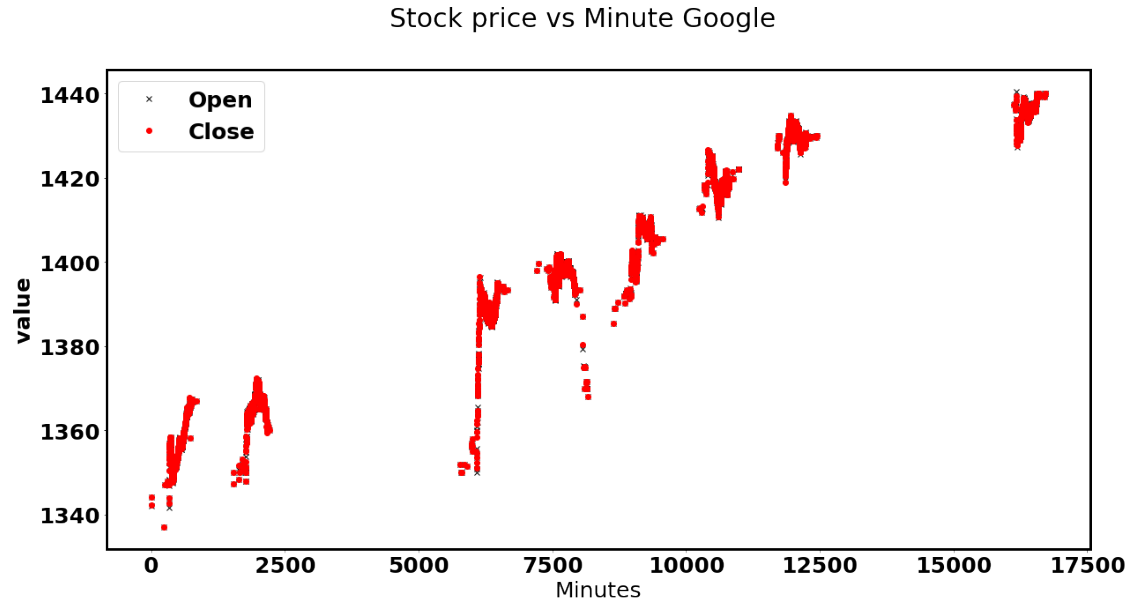
```
In [5]: import matplotlib
        from matplotlib.pylab import rc
        font = {'family': 'normal','weight': 'bold',
                'size': 25}

        matplotlib.rc('font', **font)
        rc('axes', linewidth=3)

        plt.plot(df_hf["Open"],"k",label= "Open")
        plt.plot(df_hf["Close"],"r",label= "Close")

        plt.xlabel("Minutes")
        plt.ylabel(r"${\rm value}$")
        plt.suptitle("Stock price vs Minute Google including close market")
        plt.legend()

        fig = matplotlib.pyplot.gcf()
        fig.set_size_inches(20,10)
```



```
In [6]: df_hf["DateTime"] = pd.to_datetime(df_hf["DateTime"])
        delta_min = (df_hf["DateTime"]-df_hf["DateTime"][0]).dt.total_seconds()//60
```

```
In [7]: import matplotlib
        from matplotlib.pylab import rc
        font = {'family': 'normal','weight': 'bold',
                'size': 25}
```

```
matplotlib.rc('font', **font)
rc('axes', linewidth=3)

plt.plot(delta_min,df_hf["Open"],"kx",label= "Open")
plt.plot(delta_min,df_hf["Close"],"ro",label= "Close")

plt.xlabel("Minutes")
plt.ylabel(r"${\rm value}$")
plt.suptitle("Stock price vs Minute Google")
plt.legend()

fig = matplotlib.pyplot.gcf()
fig.set_size_inches(20,10)
```



```
In [8]: # This time Try DQN on df and df_hf

        class Environment1:

            def __init__(self, data, history_t=90):
                self.data = data
                self.history_t = history_t
                self.reset()

            def reset(self):
                self.t = 0
                self.done = False
                self.profits = 0
                self.positions = []
                self.position_value = 0
                self.history = [0 for _ in range(self.history_t)]
                return [self.position_value] + self.history # obs

            def step(self, act):
                reward = 0

                # act = 0: stay, 1: buy, 2: sell
                if act == 1:
                    self.positions.append(self.data.iloc[self.t, :]['Close'])
                elif act == 2: # sell
                    if len(self.positions) == 0:
                        reward = -1
                    else:
                        profits = 0
                        for p in self.positions:
                            profits += (self.data.iloc[self.t, :]['Close'] - p)
                        reward += profits
                        self.profits += profits
                        self.positions = []

                # set next time
                self.t += 1
                self.position_value = 0
                for p in self.positions:
                    self.position_value += (self.data.iloc[self.t, :]['Close'] - p)
                self.history.pop(0)
                self.history.append(self.data.iloc[self.t, :]['Close'] - self.data.iloc[(self.

                # clipping reward
                if reward > 0:
                    reward = 1
                elif reward < 0:
                    reward = -1

                return [self.position_value] + self.history, reward, self.done
```

```
In [9]: # df

        df['Date'] = pd.to_datetime(df['Date'])
        data = df.set_index('Date')
        env = Environment1(df)


        # df_hf
        data_hf = df_hf.set_index('DateTime')
        env_hf = Environment1(df_hf)
```

```
In [10]: # Train test split :) Here I didn't shuffle dataset since we want to check whether we

         delta_split = int(df.shape[0]*0.7)

         train = df[:delta_split]
         test = df[delta_split:]
         print("Training date length",len(train),"testing date length", len(test))
```

```
Training date length 2333 testing date length 1000
```

```
In [11]: delta_split = int(df_hf.shape[0]*0.7)
         train_hf = df_hf[:delta_split]
         test_hf = df_hf[delta_split:]
         print("Training date length hf",len(train_hf),"testing date length hf", len(test_hf))
```

```
Training date length hf 2275 testing date length hf 976
```

```
In [13]: import chainer
         import chainer.functions as F
         import chainer.links as L
         import copy
         import time



         def train_dqn(env):

             class Q_Network(chainer.Chain):

                 def __init__(self, input_size, hidden_size, output_size):
                     super(Q_Network, self).__init__(
                         fc1 = L.Linear(input_size, hidden_size),
                         fc2 = L.Linear(hidden_size, hidden_size),
                         fc3 = L.Linear(hidden_size, output_size)
                     )

                 def __call__(self, x):
                     h = F.relu(self.fc1(x))
                     h = F.relu(self.fc2(h))
                     y = self.fc3(h)
                     return y

                 def reset(self):
```

```
        self.zerograds()

Q = Q_Network(input_size=env.history_t+1, hidden_size=100, output_size=3)
# GPU Option
#Q.to_gpu()

Q_ast = copy.deepcopy(Q)
optimizer = chainer.optimizers.Adam()
optimizer.setup(Q)

# Hyper-parameters
epoch_num = 25
step_max = len(env.data)-1
memory_size = 200
batch_size = 20
epsilon = 1.0
epsilon_decrease = 1e-3
epsilon_min = 0.1
start_reduce_epsilon = 200
train_freq = 10
update_q_freq = 20
gamma = 0.97
show_log_freq = 5


memory = []
total_step = 0
total_rewards = []
total_losses = []

start = time.time()
for epoch in range(epoch_num):

    pobs = env.reset()
    step = 0
    done = False
    total_reward = 0
    total_loss = 0

    while not done and step < step_max:

        # select act
        pact = np.random.randint(3)
        if np.random.rand() > epsilon:
            pact = Q(np.array(pobs, dtype=np.float32).reshape(1, -1))
            pact = np.argmax(pact.data)

        # act
```

```
            obs, reward, done = env.step(pact)

            # add memory
            memory.append((pobs, pact, reward, obs, done))
            if len(memory) > memory_size:
                memory.pop(0)

            # train or update q
            if len(memory) == memory_size:
                if total_step % train_freq == 0:
                    shuffled_memory = np.random.permutation(memory)
                    memory_idx = range(len(shuffled_memory))
                    for i in memory_idx[::batch_size]:
                        batch = np.array(shuffled_memory[i:i+batch_size])
                        b_pobs = np.array(batch[:, 0].tolist(), dtype=np.float32).resh
                        b_pact = np.array(batch[:, 1].tolist(), dtype=np.int32)
                        b_reward = np.array(batch[:, 2].tolist(), dtype=np.int32)
                        b_obs = np.array(batch[:, 3].tolist(), dtype=np.float32).resha
                        b_done = np.array(batch[:, 4].tolist(), dtype=np.bool)

                        q = Q(b_pobs)
                        maxq = np.max(Q_ast(b_obs).data, axis=1)
                        target = copy.deepcopy(q.data)
                        for j in range(batch_size):
                            target[j, b_pact[j]] = b_reward[j]+gamma*maxq[j]*(not b_d
                        Q.reset()
                        loss = F.mean_squared_error(q, target)
                        total_loss += loss.data
                        loss.backward()
                        optimizer.update()

                if total_step % update_q_freq == 0:
                    Q_ast = copy.deepcopy(Q)

            # epsilon
            if epsilon > epsilon_min and total_step > start_reduce_epsilon:
                epsilon -= epsilon_decrease

            # next step
            total_reward += reward
            pobs = obs
            step += 1
            total_step += 1

        total_rewards.append(total_reward)
        total_losses.append(total_loss)

        if (epoch+1) % show_log_freq == 0:
```

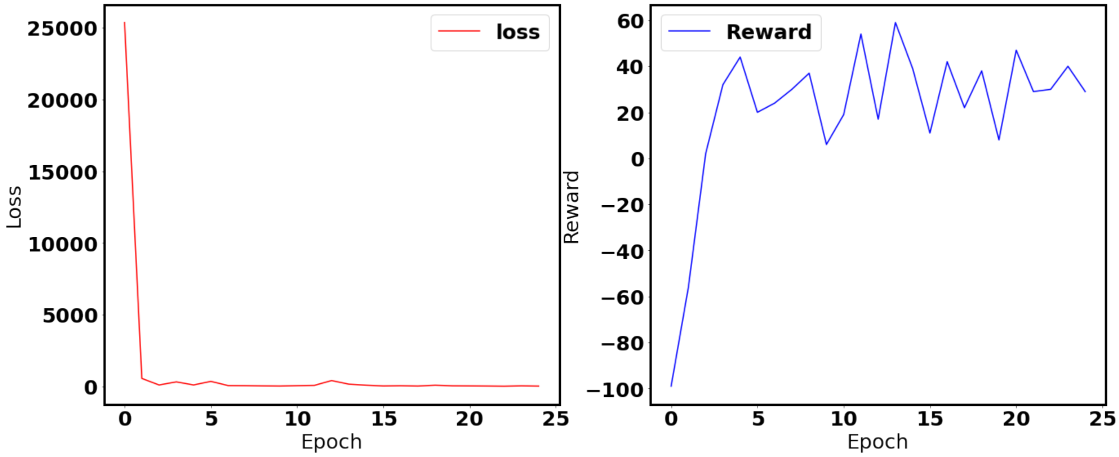
```
                     log_reward = sum(total_rewards[((epoch+1)-show_log_freq):])/show_log_freq
                     log_loss = sum(total_losses[((epoch+1)-show_log_freq):])/show_log_freq
                     elapsed_time = time.time()-start
                     print('\t'.join(map(str, [epoch+1, epsilon, total_step, log_reward, log_lo
                     start = time.time()

             return Q, total_losses, total_rewards
```

```
In [14]: # Train on google_daily:

         # Q, total_losses, total_rewards = train_dqn(Environment1(train))
```

```
In [15]: # Train on google_per_minute:
         # Epoch, epsilon (Randomness in your strategy), steps,log[reward],log[loss], elapsed
         Q, total_losses, total_rewards = train_dqn(Environment1(train_hf))
```

```
5        0.0999999999999992        11370        -15.4        5297.414914591704         113.09484
10        0.0999999999999992        22740        23.4        121.3875683060789         111.33120
15        0.0999999999999992        34110        37.6        169.700724235503         114.359278
20        0.0999999999999992        45480        24.2        62.293562510469926         111.2948
25        0.0999999999999992        56850        35.0        44.13690594714135         113.90225
```

```
In [20]: plt.subplot(1,2,1)
         plt.plot(total_losses,"r",label="loss")
         plt.xlabel("Epoch")
         plt.ylabel("Loss")
         plt.legend()

         plt.subplot(1,2,2)
         plt.plot(total_rewards,"b",label="Reward")
         plt.xlabel("Epoch")
         plt.ylabel("Reward")

         plt.legend()



         fig = matplotlib.pyplot.gcf()

         fig.set_size_inches(22,9)
         plt.legend()
         plt.show()
```

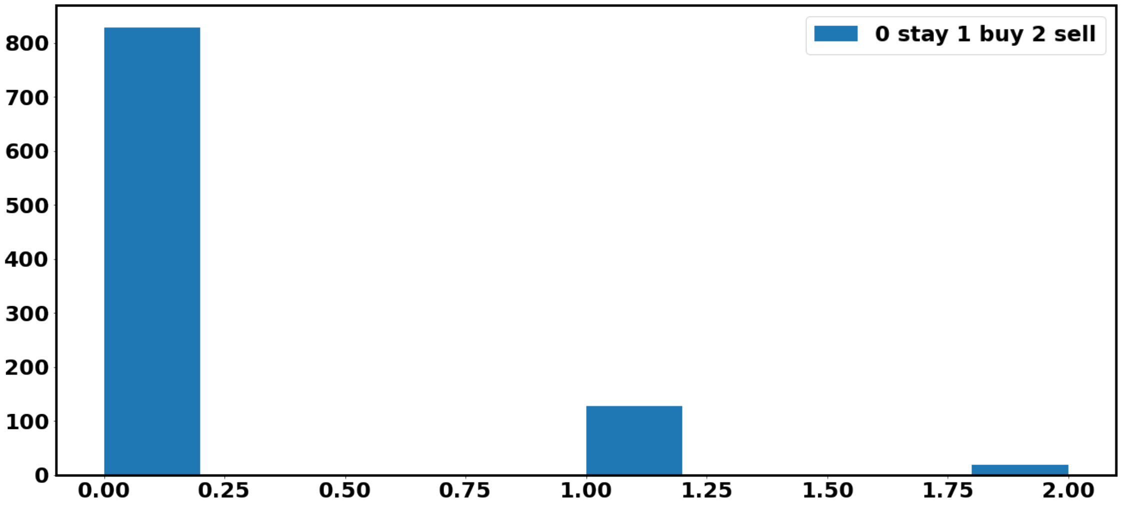


```
In [40]: # Testing :)

         # test
         test_env = Environment1(test_hf)


         # test
         pobs = test_env.reset()
         test_acts = []
         test_rewards = []

         for _ in range(len(test_env.data) - 1):
             pact = Q(np.array(pobs, dtype=np.float32).reshape(1, -1))
             pact = np.argmax(pact.data)
             test_acts.append(pact)

             obs, reward, done = test_env.step(pact)
             test_rewards.append(reward)

             pobs = obs

         test_profits = test_env.profits


         print("Test profit =%.4f in %d steps start price %.4f"%(test_profits,test_env.data.sha
```

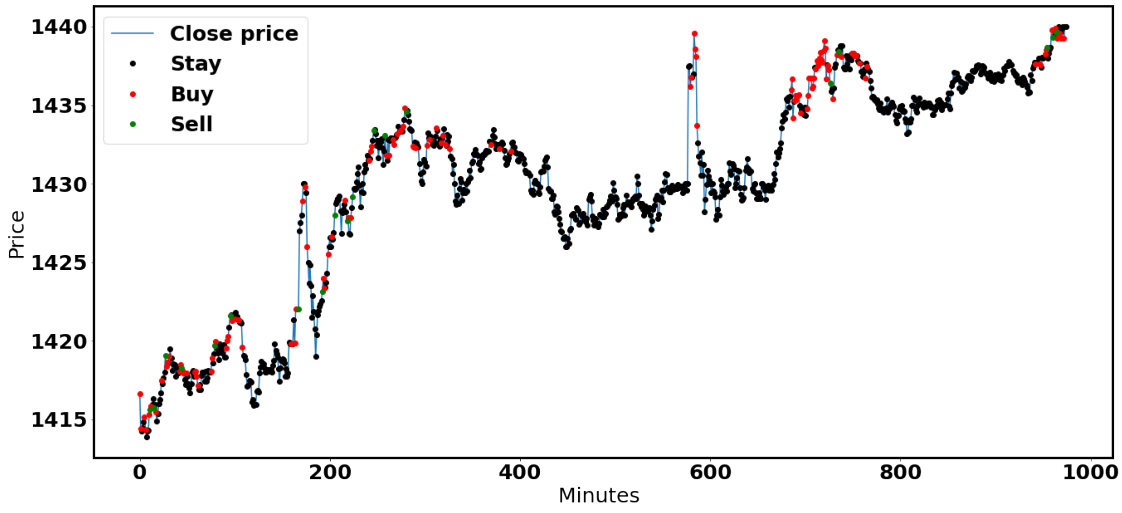
```
Test profit =117.8347 in 976 steps start price 1416.0400
```

```
In [69]: # Our actions in testing set:
         # Here 0 means stay, 1 means buy 2 means sell
         plt.hist(test_acts,label = "0 stay 1 buy 2 sell")
```

```
fig = matplotlib.pyplot.gcf()

fig.set_size_inches(22,10)
plt.legend()
plt.show()
```



```
In [68]: plt.subplot(1,1,1)
         y = test_env.data["Close"].values[1:]

         test_acts = np.array(test_acts)

         m0 = test_acts==0
         m1 = test_acts==1
         m2 = test_acts==2
         plt.plot(np.arange(0,len(y),1),y,label = "Close price")
         plt.plot(np.arange(0,len(y),1)[m0],y[m0],"ko",label = "Stay")
         plt.plot(np.arange(0,len(y),1)[m1],y[m1],"ro",label = "Buy")
         plt.plot(np.arange(0,len(y),1)[m2],y[m2],"go",label = "Sell")

         plt.xlabel("Minutes ")
         plt.ylabel("Price")



         fig = matplotlib.pyplot.gcf()

         fig.set_size_inches(22,10)
         plt.legend()
         plt.show()
```


Close price
Stay
Buy
Sell
Price
1440
1435
1430
1425
1420
1415
0
200
400
600
800
1000
Minutes

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```