EE473 Deep Reinforcement Learning Homework 4

Author: Allen Liu

Problem 1

Part 1

I will calculate the average return rate for each stock. Since by looking for the return rate on long run, it will make better decision on which stock to invest since some might get a high return for the current day, but not on the next day. In order to guarantee a stable return on a long run, checking the average return value for previous days would help.

Part 2

The mean μ and standard derivation σ is shown as below:

```
mean=240056.21068773943, std=786310.5753302638
```

```python
import numpy as np
import matplotlib.pyplot as plt

if __name__ == "__main__":
    mu_apple = 0.05
    std_apple = np.sqrt(0.1)

    mu_msft = 0.1
    std_msft = np.sqrt(0.3)

    mu_list = np.array([mu_apple, mu_msft])
    std_list = np.array([std_apple, std_msft])

    total_days = 100

    num_samples = 100

    total_apple = mu_apple
    total_msft = mu_msft

    count_apple = 1
    count_msft = 1

    indices = np.array([0, 1])

    results = np.zeros(shape=num_samples)

    for i in range(num_samples):

        val_curr = 1000
        vals = np.zeros(shape=total_days + 1)
        vals[0] = val_curr

        for j in range(total_days):
            avg_apple = total_apple / count_apple
            avg_msft = total_msft / count_msft

            rate = 0
            if avg_apple > avg_msft:
                rate = np.random.normal(mu_apple, std_apple)
                total_apple += rate
                count_apple += 1

            elif avg_msft > avg_apple:
                rate = np.random.normal(mu_msft, std_msft)
                total_msft += rate
                count_msft += 1

            else:
                index = np.random.choice(indices)
                rate = np.random.normal(mu_list[index], std_list[index])

                if index == 0:
                    total_apple += rate
                    count_apple += 1
                else:
                    total_msft += rate
                    count_msft += 1

            val_curr *= 1 + rate
            vals[j + 1] = val_curr

        results[i] = val_curr

    mean = np.mean(results)
    std = np.std(results)

    print(mean, std)

    plt.plot(results)
    plt.show()
```

Part 3

The mean μ and standard derivation σ is shown as below:

```
mean=375372.40922810714 std=2598174.816301151
```

The mean net return is expected to be higher than the method as stated in part 1

```
import numpy as np
from scipy.stats import norm

import matplotlib.pyplot as plt

if __name__ == "__main__":
    num_samples = 100
    num_days = 100
    gamma = 0.5
    num_choice = 2

    result = np.zeros(shape=num_samples)

    mu_list = np.array([0.05, 0.1])
    std_list = np.sqrt(np.array([0.1, 0.3]))

    w_list = np.array([1, 1])
```

```
mean_c = np.sum(mu_list) / 2.0
std_c = np.sum(std_list**2) / 4.0

for r in range(num_samples):

    val_curr = 1000
    for i in range(num_days):
        p_list = np.zeros(shape=num_choice)

        for j in range(num_choice):
            p_list[j] = ((1 - gamma) * w_list[j] / np.sum(w_list)) + (
                gamma / num_choice
            )

        # p_list /= np.sum(p_list)
        index = np.random.choice(a=np.arange(0, num_choice), p=p_list)

        mu = mu_list[index]
        std = std_list[index]

        rate = np.random.normal(loc=mu, scale=std)
        val_curr *= 1 + rate
        reward = norm.cdf(rate, mean_c, std_c)

        # print(reward)

        for j in range(num_choice):
            x_hat = 0

            if index == j:
                x_hat = reward / p_list[j]

            else:
                pass

            w_list[j] = w_list[j] * np.exp(gamma * x_hat / num_choice)

    result[r] = val_curr

mean = np.mean(result)
std = np.std(result)

print(mean, std)

plt.plot(result)
plt.show()
```

Excercise 4.10

$$\begin{aligned} q_{k+1}(s,a) &= \mathbb{E}\left[R_{t+1} + \max_{S_{t+1}} \gamma q_k(S_{t+1}, A_{t+1}) | S_t = s, A_t = a\right] \\ &= \sum_{s',r} p\left(s', r | s, a\right)\left[r + \max_{a'} \gamma q_k\left(s', a'\right)\right] \end{aligned}$$