n-Armed Bandit Problem

February 15, 2023

You are faced repeatedly with a choice among n different options. After each choice you receive a numerical reward chosen from a stationary probability distribution that depends on the option you selected. Your objective is to maximize the expected total reward over some time period, for example, over 1000 time steps. $q(a)$ - true value of action a $Q_t(a)$ - estimated value on the tth time step of action a If by the tth time step action a has been chosen $N_t(a)$ times prior to t, yielding rewards $R_1, R_2, ..., R_{N_t(a)}$, then its value is estimated to be

$$Q_t(a) = \frac{R_1 + R_2 + ... + R_{N_t(a)}}{N_t(a)} \tag{1}$$

If $N_t(a) = 0$, then we define $Q_t(a)$ as some default value, such as $Q_t(a) = 0$. We call this the **sample-average** method for estimating action values. **-greedy methods**: each time with a small probability , select randomly from all the acctions with equal probability independently of the action value estimates instead of behaving greedily.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt

class Bandit:
    def __init__(self, n_arm, epsilon):
        self.epsilon = epsilon
        self.n_arm = n_arm
        self.true_value = [np.random.normal(0, 1) for _ in range(n_arm)]
        self.optimal_action = np.argmax(self.true_value)
        self.estimated_value = [0 for _ in range(n_arm)]
        self.reward_history = [list() for _ in range(n_arm)]
        self.reward = 0
        self.action_history = [0 for _ in range(n_arm)]

    def step(self):
        if np.random.rand() < self.epsilon:
            action = np.random.choice(self.n_arm)
        else:
            action = np.argmax(self.estimated_value)
        self.action_history[action] += 1
        reward = self.true_value[action] + np.random.normal(0, 1)
        self.reward += reward
        self.reward_history[action].append(reward)
        self.estimated_value[action] = np.mean(self.reward_history[action])
```

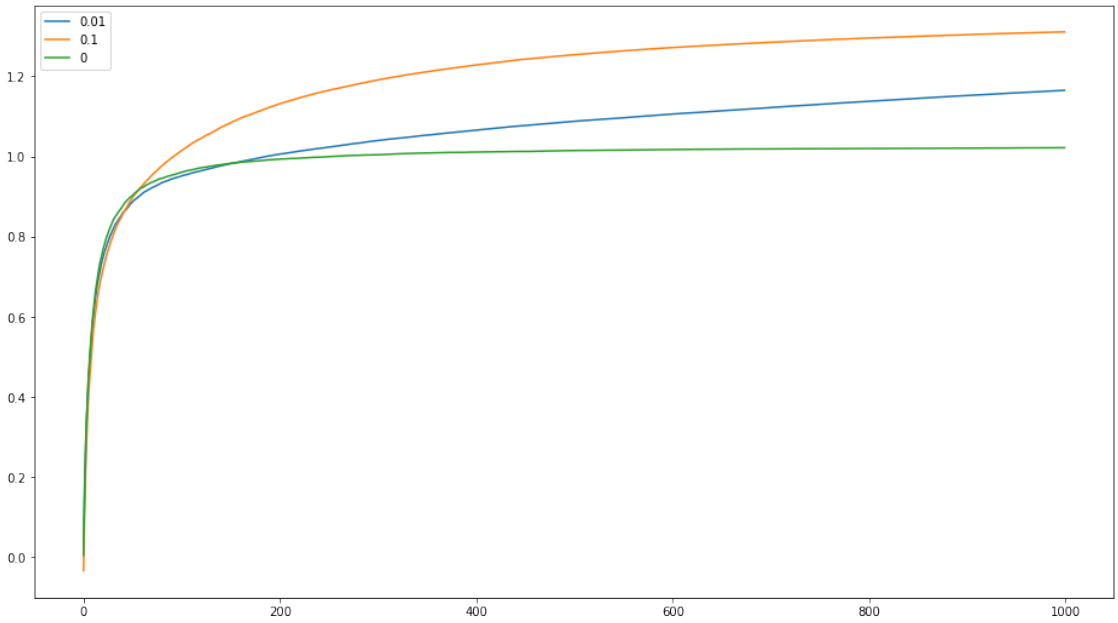
[2]:
```
bandits001 = [Bandit(10, 0.01) for _ in range(2000)]
rewards001 = list()
actions001 = list()
for i in range(1000):
    reward = 0
    action = 0
    for bandit in bandits001:
        bandit.step()
        reward += bandit.reward / (i + 1)
        action += bandit.action_history[bandit.optimal_action] / (i + 1)
    rewards001.append(reward / len(bandits001))
    actions001.append(action / len(bandits001))
```

[3]:
```
bandits01 = [Bandit(10, 0.1) for _ in range(2000)]
rewards01 = list()
actions01 = list()
for i in range(1000):
    reward = 0
    action = 0
    for bandit in bandits01:
        bandit.step()
        reward += bandit.reward / (i + 1)
        action += bandit.action_history[bandit.optimal_action] / (i + 1)
    rewards01.append(reward / len(bandits01))
    actions01.append(action / len(bandits01))
```

[4]:
```
bandits0 = [Bandit(10, 0) for _ in range(2000)]
rewards0 = list()
actions0 = list()
for i in range(1000):
    reward = 0
    action = 0
    for bandit in bandits0:
        bandit.step()
        reward += bandit.reward / (i + 1)
        action += bandit.action_history[bandit.optimal_action] / (i + 1)
    rewards0.append(reward / len(bandits0))
    actions0.append(action / len(bandits0))
```

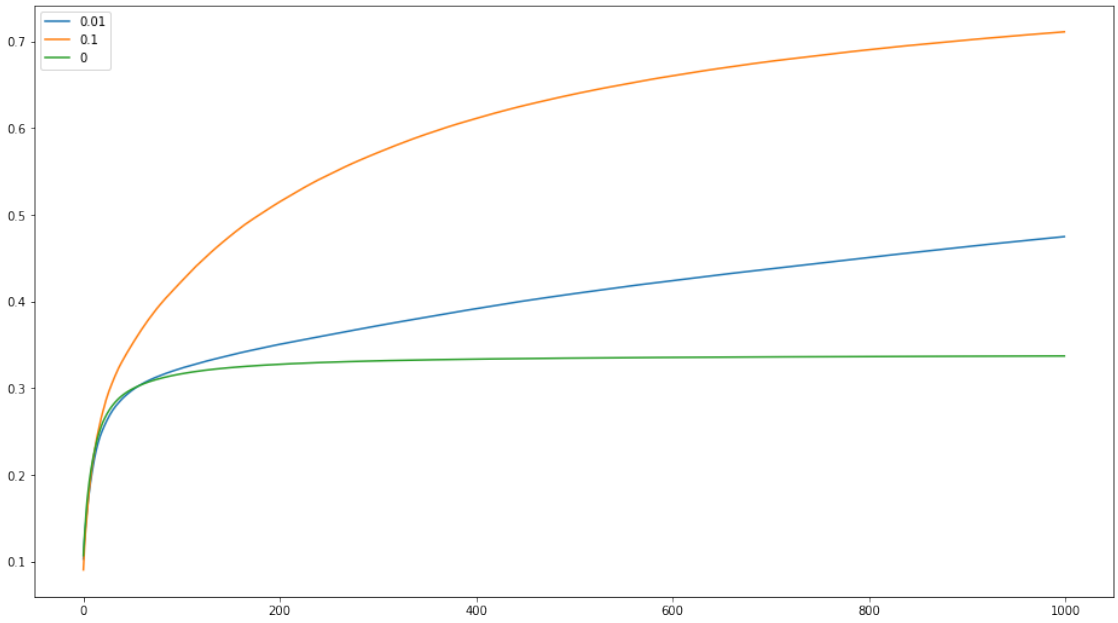
[8]:
```
fig, ax = plt.subplots(figsize=(16, 9))
ax.plot(rewards001, label='0.01')
ax.plot(rewards01, label='0.1')
ax.plot(rewards0, label='0')
ax.legend()
```

[8]:
```
<matplotlib.legend.Legend at 0x219d6b50f70>
```



[9]:
```
fig, ax = plt.subplots(figsize=(16, 9))
ax.plot(actions001, label='0.01')
ax.plot(actions01, label='0.1')
ax.plot(actions0, label='0')
ax.legend()
```

[9]: <matplotlib.legend.Legend at 0x219d6dc4340>



[ ]: