Problem set 3

Lars, Runar and Årne

February 2024

1 Monte Carlo methods

Exercise1 Monte Carlo Methods

Consider the application of a Monte Carlo method to the computation of the value of π.

8. (*) How do the two estimates change and why?

In the second situation we are sampling from a slice of the possible area. Said slice has more area outside of the circle than within relative to the entire area of the square. As such our estimate of the circle-square ratio will be smaller when sampling from the second distribution.

Exercise2 MC and MABs

You previously showed that MABs can be modelled as MDPs. Now, we want to show that the e-greedy algorithm for MABs can be seen as an instance of an MC algorithm. Specifically:

1. (*) What sort of problem are we solving in a MAB: prediction or control?

In a MAB we are attempting to find an optimal policy for selecting arms of the bandit so its a control problem.

2. (*) In a MAB you compute expected rewards of arms. Would this correspond to a state-value function or an action-value function?

The arms of a bandit are actions available, so the expected rewards of arms would correspond to an action-value function.

3. (*) What would correspond to a trajectory in a MAB? What would be a collection of trajectories?

A trajectory in an MAB would simply be a set of levers selected in order. We can a collection of multiple trajectories by varying the levers selected at each timestep. We could also model it as a set of alternating states and levers with the states being the basis state repeating.

4. (*) To what MC family of algorithms would you relate the MAB algorithms we studied: MC prediction or MC control?

The algorithms we have studied starts of with a base policy and a simulator for the problem. Then they compute an optimal policy. This would correspond to the control familiy of MC methods.

5. (*) What would correspond to MC prediction in a MAB?

If we are given a set of probabilities for each lever in a MAB and wish to learn estimate our expected reward under that policy, that would correspond to MC prediction.

6. (*) What is the purpose of e in MABs and in MC control?

e is a tuning parameter which aids with keeping a balance between exploration and exploitation. It makes sure that as we learn more about which actions are optimal, we have a higher probability of selecting optimal actions.

7. (*) Could we reduce a standard RL problem solved by MC to a MAB problem? If so, what would we lose?

We can reduce a standard problem to a MAB probelm however MABs only consider 1 state. So when reducing the problem we lose information on the different states and their rewards.

Exercise3 Importance Sampling

Consider the old slot machine you played with in the first problem assignment. Specifically: the first slot machine R takes values on the domain [0$, 5$, 10$, 20$] with probabilities [.35, .3, .25, .1]; the second slot machine S takes values on the same domain [0$, 5$, 10$, 20$] but with probabilities [.3, .35, .3, .05].

1. As ground truth to use for reference, compute the exact expected values E[R] and E[S].

Expected value for slot machine R:

$$E[R] = (0 * 0.35) + (5 * 0.3) + (10 * 0.25) + (20 * 0.1)$$

$$E[R] = 0 + 1.5 + 2.5 + 2 = 6$$

Expected value for slot machine S:

$$E[S] = (0 * 0.3) + (5 * 0.35) + (10 * 0.3) + (20 * 0.05)$$

$$E[S] = 0 + 1.75 + 3 + 1 = 5.75$$

2. Implement the two slot machines as functions from which you can get samples.

```python
import random

def slot_machine_R():
    value = [0, 5, 10, 20]
    probability = [0.35, 0.3, 0.25, 0.1]
    return random.choices(value, probability)[0]

def slot_machine_S():
    value = [0, 5, 10, 20]
    probability = [0.3, 0.35, 0.3, 0.05]
    return random.choices(value, probability)[0]
```

3. Play with slot machine S. Sample 10000 rewards and compute the empirical expectation $\hat{E}[S]$.

```python
def empirical_expectation_S(sample):
    rewards = [slot_machine_S() for _ in range(sample)]
    return sum(rewards)/sample, rewards

sample = 10000
empirical_expectation, rewards = empirical_expectation_S(sample)
print(f'Empirical expectation of slot machine S with {sample}
    samples: {empirical_expectation}')
```

Running this code yielded an empirical expectation between of 5.7185, this is of course expected as the expected value of slot machine S is 5.75. (Just a disclaimer, running this code multiple times would yield different empirical expectations, but they would all be close to 5.75 which is the ground truth)

4. (*) Compute the ratio p between the distribution probability of R and S.

$$p = [\frac{0.35}{0.3}, \frac{0.3}{0.35}, \frac{0.25}{0.3}, \frac{0.1}{0.05}]$$

5. (*) Suppose now that you only have the samples from S that you have already collected. Suppose, also, that someone let you know the value of p. Use importance sampling to compute the empirical expectation of $\hat{E}[R]$ from the samples of S and p.

```python
p = {
    0: 0.35/0.3,
    5: 0.3/0.35,
    10: 0.25/0.3,
    20: 0.1/0.05
}

adjusted_samples_R = [sample * p.get(sample) for sample in rewards]
empirical_expectation_R = sum(adjusted_samples_R) / len(
    adjusted_samples_R)

print(f'Empirical expectation of R using importance sampling: {
    empirical_expectation_R}')
```

Using the samples from S that we have already collected and the importance ratio p, we computed the empirical expectation of R using importance sampling, yielding us a result of 5.985. This again was expected, considering that the true expected value for slot machine R is 6.

6. Suppose now that you play with a machine S' with a reward distribution of [.25, .25, .25, .25]. Sample again 10000 rewards, evaluate $\hat{E}[S']$, compute the ratio p between the distribution probability of R and S', and estimate $\hat{E}[R]$ via importance sampling from the samples of S' and p.

```python
def slot_machine_S1():
    value = [0, 5, 10, 20]
    probability = [0.25, 0.25, 0.25, 0.25]
    return random.choices(value, probability)[0]

def empirical_expectation_S1(sample):
    rewards = [slot_machine_S1() for _ in range(sample)]
    return sum(rewards)/sample, rewards

sample = 10000
empirical_expectation, rewards = empirical_expectation_S1(sample)
print(f"Empirical expectation of slot machine S' with {sample}
    samples: {empirical_expectation}")

p = {
    0: 0.35/0.25,
    5: 0.3/0.25,
    10: 0.25/0.25,
    20: 0.1/0.25
}

adjusted_samples_R = [sample * p.get(sample) for sample in rewards]
empirical_expectation_R = sum(adjusted_samples_R) / len(
    adjusted_samples_R)

print(f"Empirical expectation of R using importance sampling from
    the samples of S' and p: {empirical_expectation_R}")
```

Results:
Empirical expectation of slot machine S' with 10000 samples: 8.683
Empirical expectation of R using importance sampling from the samples of S' and p: 5.9864

We get some expected results seeing as the expected value of S' is 8.75 (0*.25 + 5*.25 + 10*.25 + 20*.25 = 8.75). The empirical expectation of R using importance sampling from the samples of S' and p is still close to its ground truth of 6, which it should be given the vast amount of samples.

7. Suppose now that you play with a machine S" with a reward distribution of [.001, .499, .499, .001]. Sample again 10000 rewards, evaluate $\hat{E}[S'']$, compute the ratio p between the distribution probability of R and S", and estimate $\hat{E}[R]$ via importance sampling from the samples of S" and p.

```python
def slot_machine_S2():
    value = [0, 5, 10, 20]
    probability = [0.001, 0.499, 0.499, 0.001]
    return random.choices(value, probability)[0]

def empirical_expectation_S2(sample):
    rewards = [slot_machine_S2() for _ in range(sample)]
    return sum(rewards)/sample, rewards

sample = 10000
empirical_expectation, rewards = empirical_expectation_S2(sample)
print(f"Empirical expectation of slot machine S'' with {sample}
    samples: {empirical_expectation}")

p = {
    0: 0.35/0.001,
    5: 0.3/0.499,
    10: 0.25/0.499,
    20: 0.1/0.001
}

adjusted_samples_R = [sample * p.get(sample) for sample in rewards]
empirical_expectation_R = sum(adjusted_samples_R) / len(
    adjusted_samples_R)

print(f"Empirical expectation of R using importance sampling from
    the samples of S'' and p: {empirical_expectation_R}")
```

Results:
Empirical expectation of slot machine S" with 10000 samples: 7.519
Empirical expectation of R using importance sampling from the samples of S" and p: 5.40811623246493

Again, the empirical expectation S" is close to its true expected value of 7.505 (0*.001 + 5*.499 + 10*.499 + 20*.499 = 7.505). But the empirical expectation of R using importance sampling from the samples of S" and p is suddenly not close to its ground truth of 6 anymore. Running the code multiple times we can see that the empirical expectation of R using importance sampling actually jumps around quite a lot, mostly between 5.5 to 7.

8. (*) How do the results you computed differ? What explains the differences?

The results of the empirical expectation of R using importance sampling from the samples of S' and S" differed. While using samples from S', which had a uniform distribution for its reward, we observed that the outcome of the empirical expectation was really close to the true expected value of R for each time we ran the code. On the contrary, using samples from S" resulted in a lot more variance on the outcome of the empirical expectation of R.
We believe this can be explained by the difference in distribution of rewards in the two slot machines. S" has a "bimodal" distribution, heavily weighted towards 5$ and 10$, while S' has a uniform distribution. The difference in distribution shape affects how importance sampling adjusts the samples to es-

timate the empirical expectation of R.

Exercise4 Q-learning

In class we introduced Q-learning as an off-policy algorithm.

1. (*) Given that we do off-policy control, why do we not need to rely on importance ratio in Q-learning?

Q-learning learns from the maximum expected future return regardless of how the action was selected, excluding the need for importance ratio. Therefore, Q-learning manages to learn the optimal policy $\pi*$, without explicitly requiring importance sampling corrections. This also makes it very flexible and used a lot, compared to other off-policy methods.