ECE433/COS435 Introduction to RL
Assignment 0: Review of Topics
Spring 2024

Due February 1st, 2024

Solutions

Question 1. Probability [0 pts]

In a reinforcement learning setting, we are often interested in settings where an agent interacts with a simple game. We often want to learn the agent's policy π, or its "rule" for making actions. In this problem, you will derive properties of different "rules" over an extremely simple game that may appear in later, more interesting parts of the course.

Suppose we are playing a really simple game with boxes marked $1, ..., 10$. One of these boxes contains a golden coin that awards the player with a reward of 5, one contains a silver coin that awards the player with a reward of 1, and the rest contain nothing. The player gets a choice to open box, and based on the outcome, their final score is just the reward they receive.

Question 1.a [1 pts]

Suppose our **agent's policy** is to uniformly select one of boxes labelled $1, ..., 10$ with equal probability. Conditioned on the fact the the gold coin is in box 5 and the silver coin is in box 1, what is the expected reward?

Solution

By linearity of expectation, $\mathbb{E}[\mathcal{R}] = \frac{1}{10} \cdot 5 + \frac{1}{10} \cdot 1 = \frac{6}{10}$

Question 1.b [1 pts]

Given the same assumptions as (1.a), what is the policy of the agent that maximizes the expected reward?

Solution

The policy of always selecting box 5 yields the highest expected reward, since the game is entirely deterministic.

Question 1.c [1 pts]

What is the entropy of the policy in (1.a) and the entropy of the policy in (1.b)?

Remark: *The optimal distribution having an entropy of 0 is not a coincidence! This is obviously a super simple example, but we often run into scenarios with stochasticity where a deterministic policy is optimal.*

Solution

Entropy of (1.a) is $H(\pi_{1a}) = \sum_{i=1}^{10} -\frac{1}{10} \log \frac{1}{10} = \log 10$
Entropy of (1.b) is $H(\pi_{1b}) = 0$.

Question 1.d Coding

Suppose instead of boxes, we now deal with a *continuous* set of possible decisions. In this scenario, our agent can choose any number $a \in \mathbb{R}$, and the reward given to the player is defined by

$$\mathcal{R}(x) = \max\left\{0, 1 - |x - 1|\right\}$$

In this problem, you will use this reward function to compute the expected return of a policy (to be defined). The key idea is that an expectation can be approximated through sampling (recall Monte Carlo sampling from COS324). In this problem, you will familiarize yourself with the `torch.distributions` library.

See the colab notebook (1.d) for more details. Fill out the indicated code segments below after solving the problem.

Solution

Answer: Any number near 0.58 suffices.

```python
# Solution
MAX_ITERATIONS = 200000
mean = 0.6 * torch.ones((MAX_ITERATIONS))
std = 0.3 * torch.ones((MAX_ITERATIONS))

policy = Normal(mean, std)
r = reward(policy.sample())
print("Mean:", r.mean())
```

Question 1.e Coding

Plot the reward distribution under this policy using `matplotlib.pyplot.hist()` or some other histogram plotting library. Make sure to label your axes for this problem (x-axis is reward, y-axis is probability density).

Hint. *When using `matplotlib.pyplot.hist()`, remember it needs to be a PDF! (check the documentation.)*

Solution

```
MAX_ITERATIONS = 200000
mean = 0.6 * torch.ones((MAX_ITERATIONS))
std = 0.3 * torch.ones((MAX_ITERATIONS))

policy = Normal(mean, std)
r = reward(policy.sample())

plt.hist(r, bins=200, density=True)
plt.xlabel("Reward under policy")
plt.ylabel("Probability Density")
plt.show()
```

Question 2. MLE

The purpose of this question is to review Maximum Likelihood Estimation (MLE), which appears in many machine learning settings. In previous courses, students may have seen or solved a few questions on computing the Maximum Likelihood Estimator for a set of data, in which case this problem will be review.

A random variable X has a Normal distribution with parameters μ, σ^2 if its probability distribution is uniquely defined as

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Question 2.a

Let $\theta = \sigma^2$. For a set of datapoints $X_1, ..., X_n$ sampled from a Normal distribution with unknown parameters μ, θ find the log-likelihood function $L(\mu, \theta)$.

Solution

$$L(\mu,\theta) = \prod_{i=1}^{n} f_X(x_i) = \left(\frac{1}{(2\pi\theta)^{n/2}}\right) \exp\left(-\frac{1}{2\theta}\sum_{i=1}^{n}(x_i-\mu)^2\right)$$

$$\log L(\mu,\theta) = -\frac{n}{2}\log 2\pi - \frac{n}{2}\log\theta - \frac{1}{2\theta}\sum_{i=1}^{n}(x_i-\mu)^2$$

Question 2.b

Using your answer in (3.a), solve for the MLE of μ and θ.

Solution

$$0 = \frac{\partial \log L(\mu,\theta)}{\partial\mu} = \frac{1}{\theta}\sum_{i=1}^{n}(x_i-\mu)$$

$$\implies \mu = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$0 = \frac{\partial \log L(\mu,\theta)}{\partial\theta} = -\frac{n}{2\theta} + \frac{\sum_{i=1}^{n}(x_i-\mu)^2}{2\theta^2}$$

$$\implies \theta = \frac{\sum_{i=1}^{n}(x_i-\mu)}{n}$$

Solutions should also show that the critical points are maximums e.g. show that second derivatives are negative and show that these are the only critical points. A hand-wavy argument suffices, no need to be that rigorous.

Question 3. Coding

The objective of this question is to familiarize you with the basic mechanics of PyTorch and Python. See **Question 3** on the .ipynb notebook, but you will be coding up gradient descent and building a basic neural network model.

Question 3.a

In this problem, you will code a barebones version of gradient descent using `numpy`, as well as an analogous version using `torch`. The goal is to re-familiarize you with the inner-workings of gradient descent, as well as the standard formulations we use in practice.

```
# Your code here...
def gradient_descent(X, y, weights: np.ndarray, eta: float, iterations:
    int) -> None:
  for i in range(iterations):
    norm = 1 / X.shape[0]
    grad = (weights.T @ X.T @ X - y.T @ X) * norm
    weights -= eta * grad.T
```

Question 3.b

Rewrite your solution to (a) using the torch library. Your solution must use `loss.backward()` and `weight.grad`.

```
# Your code here...
def gradient_descent_torch(X, y, weights: torch.tensor,
eta: float, iterations: int) -> None:
  for i in range(iterations):
    y_pred = (X @ weights).T
    loss = (y_pred - y).pow(2).mean(axis=-1) / 2

    # backprop
    loss.backward()

    # optimize
    with torch.no_grad():
      weights -= eta * weights.grad
      weights.grad = None
```