Online and Reinforcement Learning (2025)
Home Assignment 3

Davide Marchi 777881

Contents

1 Direct Policy Search

1.1 Multi-variate normal distribution

In this exercise we use the notation

$$N(m, C)$$

to denote the multivariate normal distribution with mean $m \in \mathbb{R}^n$ and covariance matrix $C \in \mathbb{R}^{n \times n}$. In particular, $N(0, I)$ denotes the standard normal distribution in $\mathbb{R}^n$.

1.

Let $a \in \mathbb{R}^n$ be a nonzero vector and consider the matrix

$$C = aa^T.$$

(a) Rank of $C = aa^T$

For any $x \in \mathbb{R}^n$ we have

$$Cx = aa^T x = a\,(a^T x).$$

Since $a^T x$ is a scalar, it follows that Cx is always a scalar multiple of a. In other words, the image (or column space) of C is contained in $\operatorname{span}\{a\}$. Since $a \neq 0$, this is a one-dimensional subspace. Hence,

$$\operatorname{rank}(C) = 1.$$

(b) Eigenvector and Eigenvalue of $C = aa^T$

We next show that a is an eigenvector of C. Indeed,

$$C\,a = aa^T a = a\,(a^T a) = \|a\|^2\,a.$$

Thus, a is an eigenvector corresponding to the eigenvalue

$$\lambda = \|a\|^2.$$

(c) Maximum Likelihood for a One-Dimensional Normal Distribution

Consider the family of one-dimensional normal distributions with zero mean and variance σ^2, that is,

$$N(0, \sigma^2).$$

The probability density function (pdf) is given by

$$p(a \mid \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{a^2}{2\sigma^2}\right).$$

For a single observation $a \in \mathbb{R}$, the likelihood function is

$$L(\sigma^2) = p(a \mid \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{a^2}{2\sigma^2}\right).$$

It is more convenient to maximize the logarithm of the likelihood:

$$\ell(\sigma^2) = \log L(\sigma^2) = -\frac{1}{2}\log(2\pi\sigma^2) - \frac{a^2}{2\sigma^2}.$$

Differentiate $\ell(\sigma^2)$ with respect to σ^2:

$$\frac{d\ell}{d\sigma^2} = -\frac{1}{2\sigma^2} + \frac{a^2}{2(\sigma^2)^2}.$$

Setting the derivative equal to zero, we obtain

$$-\frac{1}{2\sigma^2} + \frac{a^2}{2(\sigma^2)^2} = 0 \quad \Longrightarrow \quad \frac{a^2 - \sigma^2}{2(\sigma^2)^2} = 0.$$

Thus,

$$a^2 - \sigma^2 = 0 \quad \Longrightarrow \quad \sigma^2 = a^2.$$

This shows that the likelihood of generating $a \in \mathbb{R}$ is maximized when $\sigma^2 = a^2$.

2.

Let $x_1, x_2, \ldots, x_m \sim N(0, I)$ be independent random vectors in $\mathbb{R}^n$. In this part, we analyze the distribution of their (unweighted and weighted) sums and determine the rank of the matrix

$$C = \sum_{i=1}^{m} x_i x_i^T.$$

(a) Distribution of $z = \sum_{i=1}^m x_i$

Since the sum of independent Gaussian random vectors is Gaussian, we have

$$z \sim N\left(\sum_{i=1}^{m} \mathbb{E}[x_i],\, \sum_{i=1}^{m} \mathrm{Cov}(x_i)\right) = N\Big(0,\, mI\Big).$$

Thus,

$$\mathbb{E}[z] = 0 \quad \text{and} \quad \mathrm{Cov}(z) = mI.$$

(b) Distribution of the Weighted Sum $z_w = \sum_{i=1}^m w_i x_i$

Let $w_1, w_2, \ldots, w_m \in \mathbb{R}_+$ be positive weights. Note that each scaled vector $w_i x_i$ is distributed as

$$w_i x_i \sim N\Big(0,\, w_i^2 I\Big).$$

Since the x_i are independent, the weighted sum z_w is Gaussian with mean

$$\mathbb{E}[z_w] = \sum_{i=1}^m w_i \mathbb{E}[x_i] = 0,$$

and covariance

$$\mathrm{Cov}(z_w) = \sum_{i=1}^m w_i^2 \, \mathrm{Cov}(x_i) = \Big(\sum_{i=1}^m w_i^2\Big) I.$$

Thus, we obtain

$$z_w \sim N\Big(0, \Big(\sum_{i=1}^m w_i^2\Big) I\Big).$$

(c) Rank of $C = \sum_{i=1}^m x_i x_i^T$

For each i, the outer product $x_i x_i^T$ is an $n \times n$ matrix of rank 1 (as shown in part (1a)). Hence, C is the sum of m rank-1 matrices. Since the x_i are sampled from the continuous distribution $N(0, I)$, they are almost surely in *general position* (i.e., any set of up to n such vectors is linearly independent). Therefore:

- If $m < n$, then almost surely the m vectors $\{x_1, \ldots, x_m\}$ are linearly independent, so

$$\mathrm{rank}(C) = m.$$

- If $m \geq n$, then the x_i will almost surely span $\mathbb{R}^n$, and hence

$$\mathrm{rank}(C) = n.$$

1.2 Neuroevolution

In this exercise we consider solving the pole-balancing task using a direct policy search method with the CMA-ES. Our policy is encoded by a feed-forward neural network. In the notebook, two versions of the network were implemented: one that includes trainable bias parameters in the hidden and output layers, and one without bias. The following sections describe the network architecture (part 1) and summarize the experimental performance (part 2).

(a) Neural Network Architecture

We used a network with a single hidden layer consisting of five neurons and a `tanh` activation. The output layer is a single neuron with a linear activation. The constructor of the network allows the user to select whether or not to include bias parameters. A code snippet implementing this in PyTorch is shown below.

```python
import torch.nn as nn

class PolicyNetwork(nn.Module):
    def __init__(self, input_dim, hidden_dim=5, use_bias=True):
        super().__init__()
        # Hidden layer: a linear layer followed by tanh activation
        self.hidden = nn.Linear(input_dim, hidden_dim, bias=
   use_bias)
        # Output layer: a single neuron with linear activation
        self.output = nn.Linear(hidden_dim, 1, bias=use_bias)

    def forward(self, x):
        x = torch.tanh(self.hidden(x))
        x = self.output(x)  # Linear output
        return x

# Instantiate the policy network.
# Set 'use_bias' to True for an architecture with trainable biases
   ,
# or False to have no bias parameters.
policy_net = PolicyNetwork(state_space_dimension, hidden_dim=5,
   use_bias=True)
```

Listing 1: Definition of the policy network.

In the code above, `hidden_dim` can be changed, but is kept to 5 as a default value. The boolean parameter `use_bias` allows for switching between the two architectures.

(b) Performance Comparison

We compared the performance of the two architectures by running the learning procedure 10 times for each variant (with bias and without bias). Two metrics were recorded:

- **Evaluations:** The number of evaluations (i.e., CMA-ES iterations) required to find a policy that balances the pole for 500 time steps.
- **Balancing Steps:** When testing the learned policies from a random starting position, the number of steps the pole remained balanced.

The summary of the experimental results is given in Table 1.

Architecture	Average Evaluations	Average Balancing Steps
With bias	1422.0	330.9
Without bias	15.8	428.7

Table 1: Performance comparison of policy networks with and without bias parameters.

The results clearly indicate that the network without bias parameters converges much faster (only about 16 evaluations on average, compared to over 1400 when using bias) and yields policies that keep the pole balanced for a longer duration (average of approximately 429 steps versus 331 steps).

A possible explanation is that adding bias parameters increases the number of free parameters in the model, enlarging the search space. This makes the optimization using CMA-ES more challenging and slows down convergence. Additionally, the simpler architecture without bias not only accelerates the search but also generalizes better to unseen starting conditions. This suggests that the increased complexity of the model is unnecessary and even counterproductive.

2 Off-Policy Optimization in RiverSwim

(i) Original RiverSwim MDP

I implemented a certainty-equivalence off-policy optimization (CE-OPO) approach in the RiverSwim environment. I used an ϵ-greedy behavior policy (with $\epsilon = 0.15$) to gather samples and periodically updated my estimates of the transition probabilities and rewards. I then ran Value Iteration (VI) on the estimated MDP to obtain an approximate Q-function.

Because running VI at every single step (up to 10^6) was computationally expensive on my machine, I chose to call VI every 1000 steps instead. This allowed me to keep a large horizon (on the order of 10^6 steps) without overheating my computer. The figure below shows the evolution of three performance metrics for the *original* RiverSwim environment:

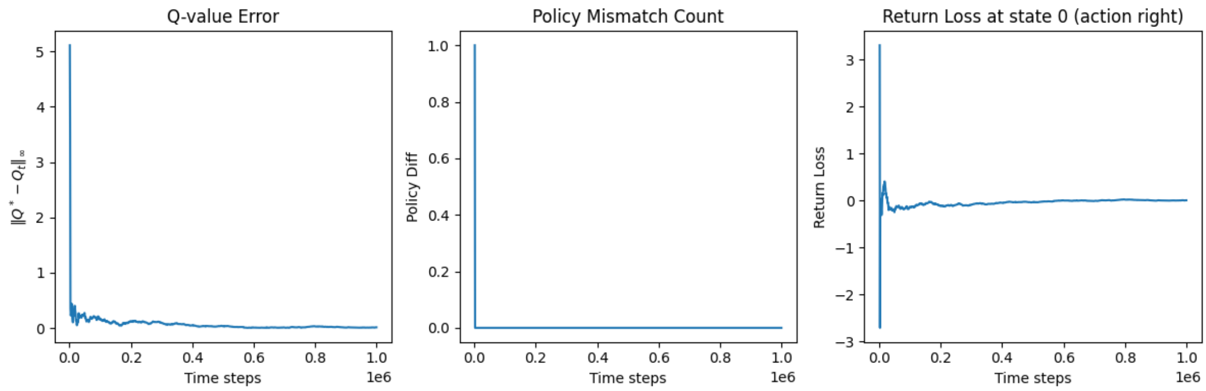

Figure 1: Performance metrics in the original RiverSwim MDP: (left) Q-value error $\|Q^* - Q_t\|_\infty$, (middle) policy mismatch count, and (right) return loss at state 0 (action = right).

(ii) Variant RiverSwim MDP

I also tested a variant of RiverSwim in which taking the action "right" in the rightmost state (index `nS-1`) yields a reward drawn uniformly at random from $[0, 2]$. Below is the snippet of my modified `step` function for this variant:

```
def variant_step(self, action):
    # If in the rightmost state (nS-1) and taking action '
right' (1):
    if self.s == self.nS - 1 and action == 1:
        reward = np.random.uniform(0, 2)
    else:
        reward = self.R[self.s, action]
    new_s = np.random.choice(np.arange(self.nS), p=self.P[self
.s, action])
    self.s = new_s
    return new_s, reward
```

Listing 2: Definition of the policy network.

In order to evaluate the true Q-function in this variant, I set the expected reward of that random transition to 1 in the "true" reward matrix. I then ran the same CE-OPO procedure (again calling VI every 1000 steps). Figure 2 shows the corresponding performance metrics:

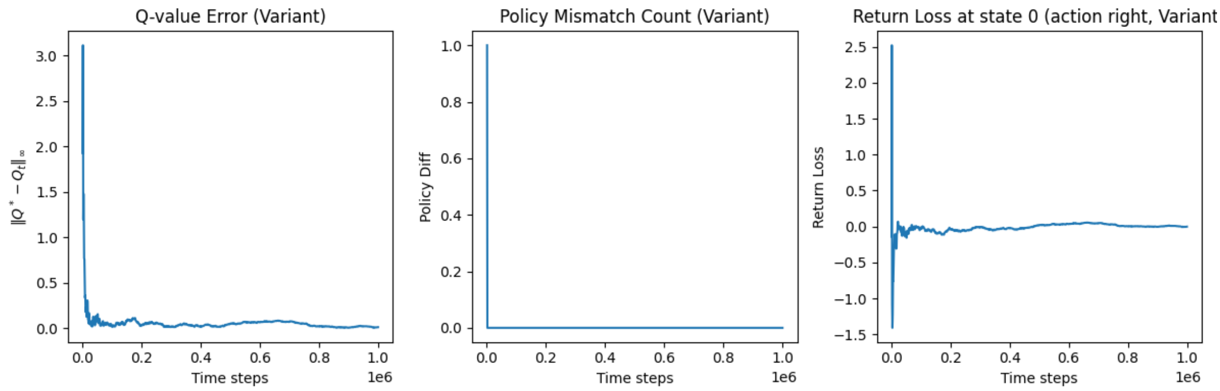

Figure 2: Performance metrics in the variant RiverSwim MDP, where the reward for action = right in the rightmost state is drawn from Unif$[0, 2]$.

(iii) Comments and Possible Explanations

From the two figures, I notice that in both the original and the variant setting, the Q-value error $\|Q^* - Q_t\|_\infty$ drops quickly within the first portion of the training, then continues to decrease more slowly as time progresses. Similarly, the policy mismatch count (i.e., the number of states in which the greedy policy w.r.t. Q_t differs from the true optimal policy) quickly goes to zero and remains there, indicating that the estimated policy converges to the true one.

In the variant setting, the randomness in the reward causes a slightly shakier behavior in the Q-value error. However, the overall performance remains robust, demonstrating that the certainty-equivalence principle effectively supports off-policy learning in RiverSwim, even under mild stochastic reward modifications.

3 Reward Shaping

In this exercise, I analyze a discounted MDP

$$M = (S, A, R, P, \gamma)$$

and its corresponding reward-shaped MDP

$$M' = (S, A, R', P, \gamma),$$

where

$$R'(s, a) = R(s, a) + \sum_{s' \in S} P(s'|s, a) F(s, a, s')$$

with the shaping function chosen in a potential-based form,

$$F(s, a, s') = \gamma\, \Phi(s') - \Phi(s),$$

for some potential function $\Phi : S \to \mathbb{R}$. I show that this transformation leaves the set of optimal policies unchanged, derive the relation between the optimal value functions, and finally derive the modified reward for a particular choice of Φ in the 6-state RiverSwim MDP.

(i) Invariance of Optimal Policies

I begin by recalling that in the original MDP, the Bellman optimality equation for the optimal Q–function is:

$$Q^*_M(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a'} Q^*_M(s',a').$$

In the reward-shaped MDP, the reward is modified as

$$R'(s,a) = R(s,a) + \sum_{s' \in S} P(s'|s,a)[\gamma\, \Phi(s') - \Phi(s)].$$

Thus, its Bellman equation becomes

$$Q^*_{M'}(s,a) = R'(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a'} Q^*_{M'}(s',a').$$

Now, I define

$$\tilde{Q}(s,a) = Q^*_M(s,a) - \Phi(s).$$

Then,

$$\tilde{Q}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a'} Q^*_M(s',a') - \Phi(s).$$

Since for any s' I can write

$$Q^*_M(s',a') = \tilde{Q}(s',a') + \Phi(s'),$$

it follows that

$$\tilde{Q}(s,a) = R(s,a) - \Phi(s) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a'} \Big[\tilde{Q}(s',a') + \Phi(s')\Big].$$

But note that

$$R(s,a) - \Phi(s) + \gamma \sum_{s' \in S} P(s'|s,a)\Phi(s') = R'(s,a)$$

by the definition of R'. Hence,

$$\tilde{Q}(s,a) = R'(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a'} \tilde{Q}(s',a').$$

This is exactly the Bellman optimality equation for M'. By the uniqueness of the solution of the Bellman optimality equation, I conclude that

$$Q^*_{M'}(s,a) = \tilde{Q}(s,a) = Q^*_M(s,a) - \Phi(s).$$

Since the greedy policy is given by

$$\pi^*(s) \in \arg\max_a Q(s,a),$$

the term $-\Phi(s)$ is independent of a so that

$$\arg\max_a Q^*_M(s,a) = \arg\max_a \Big(Q^*_M(s,a) - \Phi(s) \Big) = \arg\max_a Q^*_{M'}(s,a).$$

Thus, the set of optimal policies is invariant:

$$\pi^*_{M'} = \pi^*_M.$$

(ii) Relation Between Optimal Value Functions

Since the state-value functions are defined by

$$V^*_M(s) = \max_a Q^*_M(s,a) \quad \text{and} \quad V^*_{M'}(s) = \max_a Q^*_{M'}(s,a),$$

and since

$$Q^*_{M'}(s,a) = Q^*_M(s,a) - \Phi(s),$$

it immediately follows that

$$V^*_{M'}(s) = \max_a \Big(Q^*_M(s,a) - \Phi(s) \Big) = \Big(\max_a Q^*_M(s,a) \Big) - \Phi(s) = V^*_M(s) - \Phi(s), \quad \forall s \in S.$$

A good choice in hindsight for the potential function is $\Phi(s) = V^*_M(s)$. With this choice,

$$V^*_{M'}(s) = V^*_M(s) - V^*_M(s) = 0,$$

so that every state has a zero optimal value under the shaped rewards. Although this "trivializes" the value function, it means that the shaping does not alter the optimal policy. In practice, one cannot choose $\Phi(s)$ in this ideal way because V^*_M is unknown. However, the observation shows that a potential function resembling the optimal value function can be beneficial for guiding the learning process.

(iii) Reward Shaping for the 6-State RiverSwim MDP

I now consider the 6-state RiverSwim MDP with $\gamma = 0.98$ and choose the potential function $\Phi(s) = \frac{s}{2}$ (with the state space $S = \{0, 1, 2, 3, 4, 5\}$). The shaped reward function is then given by:

$$R'(s,a) = R(s,a) + \sum_{s' \in S} P(s'|s,a)\Big[\gamma\,\Phi(s') - \Phi(s)\Big].$$

Since I have chosen the potential function

$$\Phi(s) = \frac{s}{2},$$

the shaping term in the modified reward is given by

$$\sum_{s' \in S} P(s'|s,a)\Big[\gamma\,\Phi(s') - \Phi(s)\Big].$$

Substituting $\Phi(s') = \frac{s'}{2}$ and $\Phi(s) = \frac{s}{2}$, I obtain

$$\sum_{s' \in S} P(s'|s,a)\left[\gamma\,\frac{s'}{2} - \frac{s}{2}\right] = \frac{1}{2}\sum_{s' \in S} P(s'|s,a)\left[\gamma\,s' - s\right].$$

Note that s is constant with respect to the summation (i.e., it does not depend on s'); hence, I can factor it out:

$$\frac{1}{2}\left[\gamma \sum_{s' \in S} s' P(s'|s,a) - s \sum_{s' \in S} P(s'|s,a)\right].$$

Since $\sum_{s' \in S} P(s'|s,a) = 1$ and $\sum_{s' \in S} s' P(s'|s,a) = \mathbb{E}[s'|s,a]$, it follows that

$$\frac{1}{2}\left[\gamma\,\mathbb{E}[s'|s,a] - s\right].$$

Thus, the reward-shaped function becomes

$$R'(s,a) = R(s,a) + \frac{1}{2}\Big(\gamma\,\mathbb{E}[s'|s,a] - s\Big), \quad \forall s \in \{0,1,2,3,4,5\},\ a \in A.$$