Online and Reinforcement Learning

2024-2025

Home Assignment 3

Christian Igel **Sadegh Talebi**
Department of Computer Science
University of Copenhagen

The deadline for this assignment is **26 February 2025, 20:59**. You must submit your *individual* solution electronically via the Absalon home page.

A solution consists of:

- A PDF file with detailed answers to the questions, which may include graphs and tables if needed. Do *not* include your full source code in the PDF file, only selected lines if you are asked to do so.
- A .zip file with all your solution source code with comments about the major steps involved in each question (see below). Source code must be submitted in the original file format, not as PDF. The programming language of the course is Python.

Important Remarks:

- IMPORTANT: Do NOT zip the PDF file, since zipped files cannot be opened in *SpeedGrader*. Zipped PDF submissions will not be graded.
- Your PDF report should be self-sufficient. I.e., it should be possible to grade it without opening the .zip file. We do not guarantee opening the .zip file when grading.
- Your code should be structured such that there is one main file (or one main file per question) that we can run to reproduce all the results presented in your report. This main file can, if you like, call other files with functions, classes, etc.
- Handwritten solutions will not be accepted.

1 Direct Policy Search `(50 points) [Christian]`

1.1 Multi-variate normal distribution

Multi-variate normal distributions are often used for random exploration, and the CMA-ES is a prime example for an algorithm doing so (Hansen, 2016). At a first glance, the

CMA-ES algorithm looks very complicated. However, the basics are not if you are familiar with properties of the normal distribution. In this assignment, we will therefore look at selected math around the normal distribution. Solving it will most likely require that you consult books or the Internet for information about matrices and Gaussian distributions.

Let $\mathcal{N}(\boldsymbol{m}, \boldsymbol{C})$ denote multi-variate normal distribution with mean $\boldsymbol{m} \in \mathbb{R}^n$ and covariance matrix $\boldsymbol{C} \in \mathbb{R}^{n \times n}$. Accordingly, $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ is the distribution of a standard normally distributed random vector with mean $\boldsymbol{0} \in \mathbb{R}^n$ and covariance matrix equal to the identity matrix $\boldsymbol{I} = \mathrm{diag}(1, \ldots 1) \in \mathbb{R}^{n \times n}$.

1. Consider $a \in \mathbb{R}$ and $\boldsymbol{a} \in \mathbb{R}^n$ if $\boldsymbol{a}$ is not the zero vector.

   (a) Prove that the rank of the matrix $\boldsymbol{C} = \boldsymbol{a}\boldsymbol{a}^{\mathrm{T}}$ equals one (remember from MLA).

   (b) Prove that $\boldsymbol{a}$ is an eigenvector of matrix $\boldsymbol{C} = \boldsymbol{a}\boldsymbol{a}^{\mathrm{T}}$. What is the corresponding eigenvalue?

   (c) Consider the family of one-dimensional normal distribution $\mathcal{N}(0, \sigma^2)$ with zero mean and variance σ^2. Now we ask which of these distributions has the highest likelihood of generating $a \in \mathbb{R}$? The correct answer is the distribution with variance $\sigma^2 = a^2$. Prove this statement by writing down the likelihood function for a single observed data point a and the one-dimensional normal distribution with zero mean. Then optimize the likelihood function by setting the derivative w.r.t. the variance to zero.

   (d) *Bonus, not for submission:* Now let's put things together informally (doing this properly is tricky). Point 1a shows that the matrix $\boldsymbol{C} = \boldsymbol{a}\boldsymbol{a}^{\mathrm{T}}$ projects to a one-dimensional subspace. Point 1b shows that this subspace is in the direction of $\boldsymbol{a}$. Thus, moving the covariance matrix $\boldsymbol{\Sigma}$ of a normal distribution $\mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$ towards $\boldsymbol{C} = \boldsymbol{a}\boldsymbol{a}^{\mathrm{T}}$ makes sampling in the direction of $\boldsymbol{a}$ more likely. Point 1b combined with point 1c show that also the scaling is right for increasing the probability to sample $\boldsymbol{a}$.

2. Consider m random vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, sampled i.i.d.

   (a) How is the random vector
   $$\boldsymbol{z} = \sum_{i=1}^{m} \boldsymbol{x}_i$$
   distributed? What is its mean and covariance?

   (b) Now additionally consider m positive real-valued weights $w_1, \ldots, w_m \in \mathbb{R}^+$. How is the random vector
   $$\boldsymbol{z}_w = \sum_{i=1}^{m} w_i \boldsymbol{x}_i$$
   distributed? What is its mean and covariance?

   (c) What is (with probability one) the rank of
   $$\boldsymbol{C} = \sum_{i=1}^{m} \boldsymbol{x}_i \boldsymbol{x}_i^{\mathrm{T}} \ ?$$
   Consider the cases $m < n$ and $m \geq n$.

1.2 Neuroevolution

We look at solving the simple pole balancing problem with the CMA-ES and a policy encoded by a feed-forward neural network (Heidrich-Meisner and Igel, 2009).

Consider the notebook `CMA Cart Pole Torch Assignment.ipynb`. To run it, you first need to install some Python packages, in particular one implementing the RL environment and one for the CMA-ES. You can install a reference implementation of the CMA-ES via `python -m pip install cma`, see `https://pypi.org/project/cma` for more information. The RL enviroment is implemented in the Gymnasium framework, see `https://gymnasium.farama.org/`.

Read `https://gymnasium.farama.org/environments/classic_control/cart_pole/` to become familiar with the task.

The notebook misses an important part, the definition of the neural network representing the policy.

1. Add the definition of the neural network that represents the policy. Use a network with a single hidden layer with five neurons using tanh activation functions. The output layer should have a single neuron with a linear activation function (i.e., the activation function is the identity). The constructor of the network should allow to select between two different architectures, one where hidden neurons and output neuron have a trainable bias (offset) parameter and one where hidden neurons and output neuron do not use a bias. Show the source code snippet that defines the network in your report.

2. Compare the performance of the two different architectures. Repeat the learning ten times with and ten times without bias parameters. How long does it take on average to find a solution, that is, a network that balances from a starting position for 500 steps, in the two settings? If you try the 20 solutions again from a random starting position, how long do they balance the pole on average depending on whether bias parameters are used? How do you explain your observations?

2 Off-Policy Optimization in RiverSwim `(30 points) [Sadegh]`

In this exercise, we examine off-policy learning in the RiverSwim MDP.

Consider the 4-state RiverSwim MDP with $\gamma = 0.98$. We assume that the reward and transition functions of the MDP are unknown to us, and we are interested in learning an optimal policy using the method designed based on the certainty equivalence principle. Specifically, we assume that we start in the left-most state and interact with the MDP following an ε-greedy policy, and without any reset (i.e., there is no notion of episode, etc.). Set $\varepsilon = 0.15$.

(i) Run the `CE-OPO` algorithm for off-policy learning (with smoothing parameter $\alpha =$

0.1). Choose a sufficiently large horizon (e.g., 10^6). Plot the following performance metrics as a function of t:

1. $\|Q^\star - Q_t\|_\infty$, which captures the error in estimated Q-values at time t
2. $\mathsf{PolicyDiff}(t) := \sum_{s\in\mathcal{S}} \mathbb{I}\{\widehat{\pi}_t(s) \neq \pi^\star(s)\}$, which captures the mismatch at time t between the optimal policy $\pi^\star$ and the learned policy $\widehat{\pi}_t$ (i.e., the greedy policy w.r.t. Q_t)
3. $\mathsf{ReturnLoss(t)} := Q^\star(1, \mathsf{right}) - Q^{\widehat{\pi}_t}(1, \mathsf{right})$, which captures the *return loss* (of not choosing an optimal policy) at time t at the left bank of the river. Here $Q^{\widehat{\pi}_t}$ denotes the true Q-value of the learned policy $\widehat{\pi}_t$ (i.e., the greedy policy w.r.t. Q_t).

(ii) Repeat Part (i) for a variant of 4-state RiverSwim, which has a slightly different reward function: if action 'right' is executed in state L, it yields a random reward uniformly distributed over $[0, 2]$. More formally, $R(L, \mathsf{right}) = \mathrm{Unif}[0, 2]$, where $\mathrm{Unif}[a, b]$ represents the uniform distribution supported on $[a, b]$.

(iii) Briefly compare results in Parts (i) and (ii), indicate whether there is a meaningful difference between the two.

3 Reward Shaping (20 points) [Sadegh]

In modeling a task as an MDP, the transition function is defined by state dynamics of the underlying environment, which in many applications is governed by physical phenomena (e.g., cart-pole balancing in robotics, market dynamics in economics, etc.). In contrast, the reward function is determined by a human expert. Therefore, for a given notion of state, there is some freedom in the way rewards are defined.

In reinforcement learning, it is a common practice to modify the *original* rewards with the aim of guiding the agent to explore the environment better and to speed up learning. This technique is called *reward shaping* (Ng et al., 1999). In simple words, it consists in providing the agent with some additional rewards, beyond those offered by the MDP, in an attempt to speed up learning. It is desired that the additional reward makes learning faster while the optimal policy *remains* the same under the modified reward function.

Concretely, consider a discounted MDP $M = (\mathcal{S}, \mathcal{A}, R, P, \gamma)$. We are interested in learning the optimal policy of M, denoted by $\pi^\star_M$. Consider a *shaping reward function* $F : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ and a *reward-shaped* MDP $M' = (\mathcal{S}, \mathcal{A}, R', P, \gamma)$ where:

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

In *potential-based reward shaping*, the function F has the following form

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s), \quad \forall(s, a, s'), \tag{1}$$

for some function $\Phi : \mathcal{S} \to \mathbb{R}$ that is called the *potential function*. Note that potential-based reward shaping may increase or decrease the speed of learning, but it shapes the rewards independently of the action.

(i) Prove that for any F in (1), the set of optimal policies in M and M' coincides, namely $\pi^\star_M = \pi^\star_{M'}$.

Hint: First, recall the relation between the two reward models $R(s,a)$ and $R(s,a,s')$: $R(s,a) = \sum_{s'} P(s'|s,a)R(s,a,s')$. To prove the result, write down the Bellman optimality equation for M and M' in terms of the respective optimal Q-values $Q^\star_M$ and $Q^\star_{M'}$. Then, relate the maximizers of the two Q-functions. You may as well consult (Ng et al., 1999).

(ii) Show that the optimal value functions in the two MDPs satisfy

$$V^\star_{M'}(s) = V^\star_M(s) - \Phi(s), \quad \forall s \in \mathcal{S}.$$

Further, argue that a good choice for the potential function (in hindsight) is $\Phi(s) = V^\star_M(s)$.

(iii) Consider the 6-state RiverSwim MDP with $\gamma = 0.98$. Derive its reward-shaped MDP with $\Phi(s) = s/2$. It suffices to report the modified reward function.

References

Nikolaus Hansen. The CMA evolution strategy: A tutorial. *CoRR*, abs/1604.00772, 2016. URL `http://arxiv.org/abs/1604.00772`.

Verena Heidrich-Meisner and Christian Igel. Neuroevolution strategies for episodic reinforcement learning. *Journal of Algorithms*, 64(4):152–168, 2009.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287, 1999.