Online and Reinforcement Learning

2023-2024

Home Assignment 4

Christian Igel
Department of Computer Science
University of Copenhagen

The deadline for this assignment is **6 March 2024, 21:00**. 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)

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.

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

(c) The one-dimensional normal distribution $\mathcal{N}(0, \sigma^2)$ with zero mean having the highest likelihood to generate a has variance $\sigma^2 = a^2$. Proof 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 w.r.t. 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 $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$.

   (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) *Bonus, not for submission (→ MLB):* 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 get 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 representing 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 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 defining the network in your report.

2. Compare the performance of the two different architectures. Repeat the learning ten times with and 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? If you try the 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 Policy Gradient Methods (50 points)

The policy gradient theorem can be generalized to include a baseline, e.g.,

$$\nabla_{\boldsymbol{\theta}} J(\pi) = \sum_s \mu^\pi(s) \sum_a \nabla_{\boldsymbol{\theta}} \pi(s,a) \left(Q^\pi(s,a) - b(s)\right) \quad ,$$

where $b(s) : S \to \mathbb{R}$ is an arbitrary baseline function. Here S denotes the state space, J the performance measure, Q^π and μ^π are the state-value function and the stationary state distribution when following policy π, respectively, and $\boldsymbol{\theta}$ are the parameters of the policy π.

$\sum_a \nabla_{\boldsymbol{\theta}} \pi(s,a) b(s)$ acts as a control variate. Prove that $\mathbb{E}\left[\sum_a \nabla_{\boldsymbol{\theta}} \pi(s,a) b(s)\right] = 0$. Please have a look at the notebook `CartPolePGMAssignment.ipynb`, which implements the basic REINFORCE algorithm (Williams, 1992). It uses a softmax policy (see lecture slide "Example: Softmax policy"). The algorithm requires the score function, that, is the gradient $\nabla \ln \pi(s,a)$ w.r.t. the policy parameters.

Your task is to compute the score function and to implement a membership function

```
def gradient_log_pi(self, s, a):
    """
        Computes the gradient of the logarithm of the policy
        :param s: state feature vector
        :param a: action
        :return: gradient of the logarithm of the policy
    """
```

in the class `Softmax_policy` that computes the score function analytically. This can be done in less than additional 10 lines.

You can check if your are on the right track using the membership function `gradient_log_pi_test`, which approximates the gradient numerically.

Deliverables:

1. Derive the analytical expression for the score function of the softmax policy in the report.
2. Show the code for the implementation of the analytical score function `gradient_log_pi(self, s, a)` in the report.
3. Briefly describe how you verified that your implementation is correct.

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.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.