Final Project

1. For the final project we are interested in applying the one-step actor critic algorithms to a selection of environments with continuous state- and action spaces. You can choose from the following gymnasium environments (see Figure 2):

   - Pendulum.
   - Inverted Pendulum.
   - Inverted Double Pendulum.
   - Swimmer.

   You should choose only one of the listed environments for your final project.

   Implement and train the One-Step Actor-Critic algorithm as shown in the book by Sutton and Barto (and the lecture) and your chosen environment. For convenience, the pseudocode can be found below. Choose a suitable evaluation method for evaluating your agent, including relevant metrics and figures. Also provide some numerical evidence that your agent is learning during training (i.e., a comparison with the random policy).

   Hand in a small report containing the following:

   - Abstract: Summarize the project's goal, methodology, main findings, and the significance of your results in a concise manner.
   - Introduction: Introduce reinforcement learning and the scenarios that motivate the use of the One-Step Actor-Critic algorithm.
   - Methods: Here you describe and explain the actor-critic algorithm, neural network architectures, and hyperparameter tuning.
   - Results: Containing the evaluation of your agents on the selected environments. The plots must have mean and standard deviations and a descriptive caption.
   - Discussion: Interpret the results, comparing them to what was expected and what your simulation produced, highlighting the efficacy of the algorithm and any observed trends or anomalies. If the results are not as expected, justify here what might be the problem.

   To assist you with the coding, a skeleton program is provided to you via GitHub. The included `README` file provides additional information.

   One thing regarding the environments. If the environment has a discrete action space, then our function approximator for the policy can simply output a discrete probability distribution (which gives a probability for each action). In the context of neural networks, this would mean n output neurons for n actions with a softmax activation function.

   However, the available environments all have continuous action spaces, which we need to take into account when parameterizing the policy. If the action space is (a subset of) $\mathbb{R}$, then the policy can be defined as a Gaussian probability distribution $\mathcal{N}$, with mean μ and standard deviation σ given by parametric function approximators that depend on the state:

   $$\pi(a|s,\theta) = \mathcal{N}(a|\mu(s,\theta), \sigma^2(s,\theta)).$$

   Here $\mu : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$ and $\sigma : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}^+$ are two parameterized function approximators and d is the feature vector dimensionality.

   Instead of using two function approximators, we can use a neural network with two output heads, one for the mean and the other for the standard deviation (see Figure 1). We can then get a continuous scalar action for an action by passing the state S to the neural network, computing the two output heads, constructing the Gaussian distribution, and sampling an action from the Gaussian distribution $A \sim \mathcal{N}(a|\mu(s,\theta), \sigma^2(s,\theta))$.

Lastly, if we have n-dimensional continuous actions $\mathbf{a} \in \mathcal{A} \subseteq \mathbb{R}^n$, then the mean head will output an n-dimensional mean vector $\boldsymbol{\mu}$ and the other head an $n \times n$ covariance matrix $\boldsymbol{\Sigma}$ (more precisely, the unique elements of the covariance matrix will be outputted as a vector). The process is otherwise the same with $A \sim \mathcal{N}(\mathbf{a}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

In the `models` module of the skeleton code, there is an implementation of a two-headed multilayer perceptron.

Bonuses

There is a possibility to implement some bonuses to obtain up to 1 bonus point. This includes:

- Extensions to the standard one-step actor critic algorithms. This can include a different advantage function, learning on batches of trajectories or support for continuing/non-episodic tasks.
- Successful training on more difficult environments.

Mention (and discuss) the bonuses you have implemented in your report.

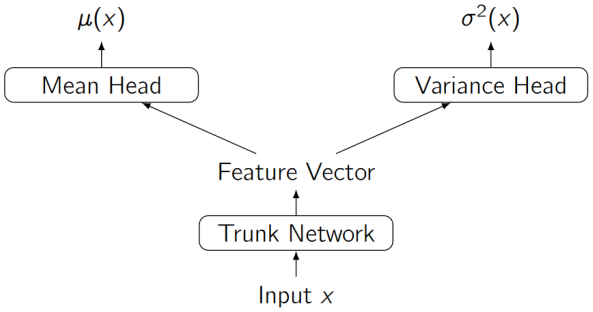

Figure 1: A neural network with two output heads for regression (figure taken from the Uncertainty in Machine Learning course). Note that in implementations it is common to use the log standard deviation instead of the variance for numerical stability.

Algorithm 1 One-Step Actor-Critic (episodic), for estimating $\pi_\theta \approx \pi_*$. δ is also referred to as the advantage here.

Input: a differentiable policy parameterization $\pi(a|s, \theta)$
Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$
Parameters: step size $\alpha^\theta > 0, \alpha^{\mathbf{w}} > 0$
Initialize policy parameters $\theta \in \mathbb{R}^{D'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^D$
for each episode **do**
    Initialize environment, get initial state S
    $I = 1$
    for $t = 1, ..., T$ (until terminal state) **do**
        $A \sim \pi(\cdot|S, \theta)$
        Take action A and observe R and next state S' from environment
        $\delta \leftarrow \begin{cases} R & \text{if } S' \text{ is terminal state} \\ R + \gamma\hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w}) & \text{otherwise} \end{cases}$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I\delta\nabla\hat{v}(S, \mathbf{w})$
        $\theta \leftarrow \theta + \alpha^\theta I\delta\nabla \ln \pi(A|S, \theta)$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$

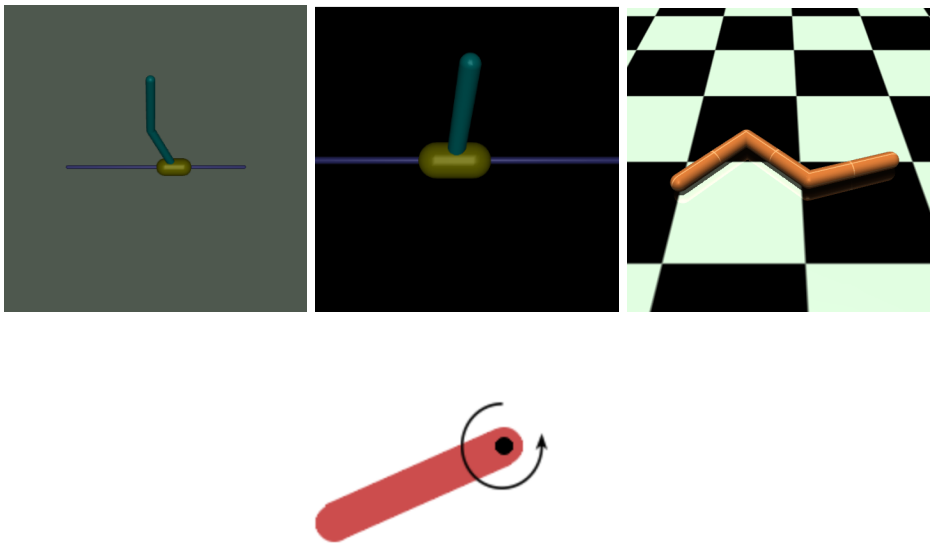

Figure 2: Gymnasium environments you can choose from. From left to right: Inverted Double Pendulum, Inverted Pendulum, Swimmer, Pendulum.