

Advanced Methods in ML 2017 - Exercise 4

1. In this exercise you will implement and train a reinforcement learning agent for solving [Open AI's Lunar Lander](#) environment. The agent will learn how to behave in the environment by updating its parameters via stochastic *policy gradient* ascent.

Task Explanation:

- OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. The gym open-source library is a collection of test problems called “environments” that you can use to explore reinforcement learning algorithms. These environments have a shared interface, allowing you to write general algorithms.
- In the Lunar Lander environment, the agent is a small spacecraft which flies in a two dimensional world. The spacecraft should use its three engines (left, main, right) to land gracefully at the landing pad, which is located at coordinates (0,0). Four discrete actions are available: do nothing, fire left engine, fire main engine, fire right engine. State vector is 8-dimensional, where the first two numbers are the spacecraft's coordinates, and the rest of the vector is various other physical properties. See link above to learn more details about this environment (dynamics, scoring scheme, etc).

Algorithm Specification:

The agent to be implemented in this exercise will be a neural network, consisting of two hidden layers with *tanh* activations. At each step, the input to the network will be the agent's observation of the environment, and the output of the network will be the probabilities from which actions will be sampled.

More formally, let $\mathbf{x} \in \mathbb{R}^8$ be the observation at a given step. The network computes the following:

$$\mathbf{h}_1 = \tanh(W_1 \mathbf{x} + \mathbf{b}_1) \quad (1)$$

$$\mathbf{h}_2 = \tanh(W_2 \mathbf{h}_1 + \mathbf{b}_2) \quad (2)$$

$$\mathbf{y} = \text{softmax}(W_3 \mathbf{h}_2 + \mathbf{b}_3) \quad (3)$$

Where $\{W_i\}_{i=1}^3$ and $\{\mathbf{b}_i\}_{i=1}^3$ are trainable parameters. The dimensions are $W_i \in \mathbb{R}^{n_i, n_{i-1}}$, where n_i is number of neurons in layer i and $n_0 = 8$ (namely the dimension of the input x) and $n_3 = 4$ (since there are four actions and this is the output layer). And $\mathbf{b}_i \in \mathbb{R}^{n_i}$. The output \mathbf{y} are the probabilities of each action given the state \mathbf{x} , namely the policy π (note that \mathbf{y} is a normalized distribution by construction). For training the network, empirically estimate the gradient of its cumulative rewards then use gradient ascent using any [SGD-based optimizer](#).

For approximating the gradient, use the following identity we showed in class:

$$\nabla_{\theta} L(\theta) = \mathbb{E} \left[\sum_{t=0}^T \left(\nabla_{\theta} \log \pi(A_t | S_t) \sum_{k=t}^T \gamma^k r(A_k, S_k) \right) \right] \quad (4)$$

Where γ is a discount factor, and should be around 0.99.

Estimate the gradient by sampling several sequences of states and actions, calculate the above expression for these, and average to get an estimate of the gradient.

For better convergence and variance reduction, use the following "trick":

Instead of multiplying the log-probability gradients by the scalars $[\sum_{k=t}^T \gamma^k r(A_k, S_k)]_{t=0}^T$, normalize these scalars per episode by subtracting their mean and then dividing by their standard deviation. Once these scalars are normalized, plug them back to Eq. 4.

Implementation Guidelines:

- Go through the Open AI gym [tutorial](#).
- Your agent should be implemented with [TensorFlow](#). Both TensorFlow and Gym are installed on CS Linux machines.
- You can use any optimizer you want, but [ADAM](#) should work ok.
- You can use any number of hidden units you want, but using roughly 15 per layer should work ok.
- You can use any batch size and stopping criteria you want, but a batch size of 10 and a stopping criteria of 30K episodes should work ok.
- Your code should finish running at < 12 hours at the worst case.

Submission Guidelines:

- Add your IDs and the path to your codes folder in the following Google Form: <https://tinyurl.com/ybskbrq4>
- Your code should be run by the command 'python lunar_lander.py', and save a [cPickle](#) file named 'ws.p', containing your agent's parameters.
- Keep in your directory a pre-trained 'ws.p' file which was trained by your algorithm.
- Your code should output parameters under which the spacecraft reasonably lands on the landing pad. The average reward of your spacecraft should be at least 180.
- Make sure your code's folder has the proper read permissions.
- Your code should be *readable* and *well-documented*.

Hints and Tips:

- Training can take a long time, so for basic testings of your implementation you can use the [CartPole](#) task, which can be solved quickly.
 - Agent performance can vary and oscillate through training, so if $\theta_0, \dots, \theta_t$ is the series of parameters the learning algorithm went through, you might want at test time to use θ_i for the i which had best performance at train time, instead of using θ_t .
 - If training performance oscillates too much, consider adding additional step size decreases to your optimizer. TensorFlow has built-in methods for this.
 - You are free to use the skeleton code at the course website.
2. Prove that for MDPs, stochastic policies are not better than deterministic ones. Namely, show that if a stochastic policy $\pi^*(a|s)$ obtains an optimal value function $V^*(s)$, then there is a deterministic policy $\hat{\pi}(s)$ that achieves the same value.
 3. Prove that the Value Iteration update operator is a contraction with respect to the ℓ_∞ norm, and constant γ .