05/12/2021

Ben-Gurion University of the Negev

Faculty of Engineering Sciences

Department of Software and Information systems

Deep Reinforcement Learning

Assignment 2

Policy Gradient Methods

General Instructions

1. The assignment should be submitted in pairs.
2. The submission will include a zip file containing:
   - A report in PDF format with answers to the questions appearing in the assignment, the requested outputs of the codes and a short instruction for running the scripts. The answers should be **short**.
   - The scripts of your solutions.
3. The scripts should be written in Python. Use the TensorFlow library for Neural Networks. Use TensorBoard for the visualization and graphs.
4. The report can be written either in English or Hebrew.
5. Write your names and ID's in the report.
6. The final submission is due 25/12/2020

Introduction

In the previous assignment, you trained an agent to find the optimal action-value function $Q^*(s,a)$, and used it to for deriving the policy out of it by $\pi^* = \arg\max_a Q^*(s,a)$. In this assignment you will optimize the policy of the agent directly with policy gradient methods. You will first implement the basic REINFORCE algorithm and then transform it to an Actor-Critic algorithm.

Section 1 – Monte-Carlo Policy Gradient (REINFORCE) (50%)

Policy gradient methods try to optimize the policy $\pi(a|s;\theta)$ directly, working on the policy space itself. They use a probability distribution over actions parameterized by θ, expressing the probability to take an action a in state s and defining the policy itself. An example for such a distribution is the softmax distribution:

$$\pi(a|s;\theta) = \frac{\exp(\theta_i)}{\sum_j \exp(\theta_j)}$$

The parameters are updated with gradient ascent to maximize the return, which is the sum of rewards starting from time-step t until the end of the episode:

$$R_t = \mathrm{E}[\sum_t^T R(s_t)|\pi_\theta]$$

Where T is the number of steps in an episode.

REINFORCE is a basic policy gradient method that is updating θ in each iteration taking a step in the direction of:

$$\mathrm{E_t}[\nabla log \pi_\theta(a_t|s_t) R_t]$$

The REINFORCE algorithm, presented with a slightly different notation:

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^n$
Initialize policy weights $\boldsymbol{\theta}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G_t \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t,\boldsymbol{\theta})$

Figure 1- REINFORCE Alorithm

For reducing the variance of the model, it is common to use a baseline. The baseline is subtracted from the return to form the advantage estimate:

$$\hat{A}_t = R_t - b(s_t)$$

The advantage is used in the computation of the gradient instead of the return. A well accepted choice for the baseline is the value function $V^{\pi}(s_t)$.

- What does the value of the advantage estimate reflect?
  Why is it better to follow the gradient computed with the advantage estimate instead of just the return itself? (5%)
- The reduction of the baseline **does not** introduce bias to the expectation of the gradient since:

  $$\mathrm{E}_{\pi_\theta}[\nabla log \pi_\theta(a_t|s_t) b(s_t)] = 0$$

  What is the prerequisite condition for that equation to be true? Prove the equation (10%)

The value function $V^{\pi}(s_t)$ can be approximated with a function approximation (NN) using another set of weights, v, similar to what you did in the previous assignment with the Q-value.

Change the code in policy_gradients.py from the basic form of the REINFORCE algorithm so it will use an advantage estimate with a value-function approximation baseline instead of the actual return of every episode. The agent is running on the same environment from assignment 1. Feel free to change whatever parameters you want, including the architecture of the policy network and the actions' probability distribution. (35%)

- Compare the results before and after the change (please refer also to convergence time).

Section 2 – Advantage Actor-Critic (50%)

Instead of having to wait until the end of the episode before computing the advantage of each time-step, we can estimate the return R_t with the Q-value $Q_w(s, a)$ using another set of weights, w. So now we have the advantage function as:

$$[1] \quad \hat{A}_t = \hat{Q}_w(s_t, a_t) - \hat{V}_v(s_t)$$

And a total of 3 sets of weights in the model. But, as you can see in the Actor-Critic algorithm in figure 2, only 2 sets of weights are eventually used, and the TD-error of the value function is used in the update of the policy function parameters instead of the advantage estimate from equation 1.

- Why is using the TD-error of the value function for the update of the policy network parameters is practically the same as using the advantage estimate (of equation 1)? Prove. (5%)
- Explain which function is the *actor* and which is the *critic* of the model and what is the role of each one. (3%)

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

Figure 2- Actor-Critic Algorithm

Change the code in policy_gradients.py once again so it will implement the actor-critic algorithm. You can use the same network you built in the previous section for the value function and the same network we provided for the policy function. (42%)

- Compare the results with the previous two models.