Reinforcement Learning Lecture: Homework 07

Ngo Anh Vien

MLR, University of Stuttgart

June 8, 2017

[Programming] Write a policy gradient algorithm (using Finite Difference Method) on Cart-Pole as in Fig. 1 (Sutton's RL book, 1998). The task is to apply forces to a cart moving a long a track in order to keep the pole balanced. If the pole falls apart a given angle (12 degree = 0.21 rad), the episode terminates. The termination also happens when the cart runs off the track. The state space of this task is defined as $s = \{x, \dot{x}, \theta, \dot{\theta}\}$, where $x, \dot{x}$ are the position and velocity of the cart, $x \in [-2.4, 2.4]$; $\theta, \dot{\theta}$ are the the angle and angular velocity (w.r.t the vetical) of the pole. The episode always starts at $\{0, 0, 0, 0\}$. Actions are continuous $a \in [-10, +10]$. The reward function is simple, if the pole is still balanced $r = 1.$, otherwise if fails $r = -1$. For the dynamics of this system, you can refer to the appendix of this note, in which to make the task more interesting, I added a small Gaussian noise to the transition.

Implmentation Note:

The policy is a stochastic Gaussian controller (which is parameterized by parameters w) as

$$\pi(a|s) = \frac{1}{Z} \exp\Big(-\frac{(w^\top s - a)^2}{2\sigma^2}\Big)$$

where Z is a normalization constant of the Gaussian policy distribution, and note that we are using a linear policy function purturbed with a small Gaussian noise (which is similar to a PID controller).

- See the FD algorithm in slide 23 (an updated version).
- Let's fix $\sigma^2 = 0.001$.
- Each $\delta w_i^{(j)}$ (the dimension j of a sample i) is sampled from a uniform distribution $[-1, 1]$. Then run one episode to compute $J(w + \delta w_i)$. Assume that $J(w)$ is evaluted using 50 episodes.

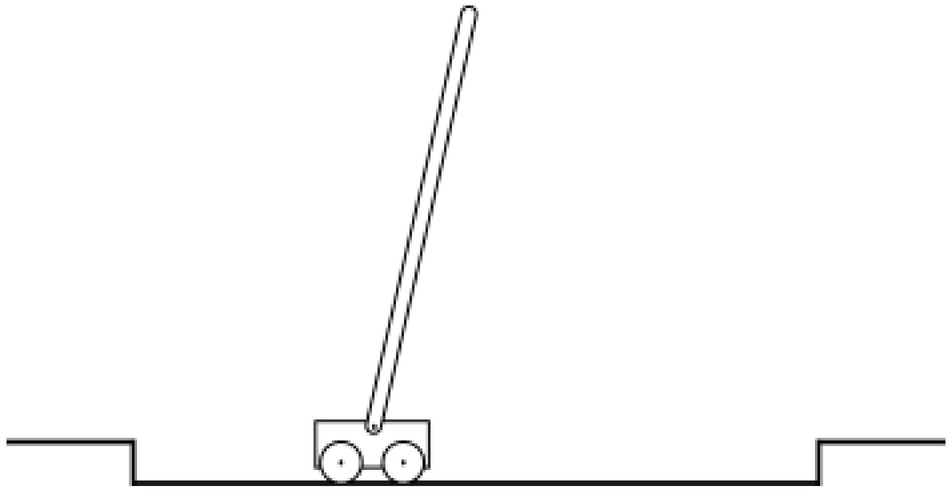

Figure 1: A pole-balancing task

- $\gamma = 1.0$
- The number of samples of δw: $M = 50$.
- Each episode terminates either after 1000 steps or observing the failure of the pole.

1 Normalized gradient

In this part, you are asked to use a normalized gradient to update w with a **fixed** step-size $\alpha = 0.5$

$$w = w + \alpha \frac{g_{FD}}{\|g_{FD}\|}$$

Step by step:

- Run the above algorithm for 200 iterations. Record $\{k, J(w_k)\}_{k=1}^{200}$ for each iteration.
- Plot the data $\{k, J(w_k)\}_{k=1}^{200}$

2 Heuristic step-size

In this part, you are asked to repeat the above exercise with a better choice of α. Let's use your own experience in optimization to heuristically choose the best scheduling strategy better than the above fixed choice? (for an example of deacreasing α by time: $\alpha_t = 10./t$, where t is the iteration number).

3 Adaptive step-size

In more complex domain, we might need to adaptively tune α. In this question, you are asked to program the Rprop method (Resilient Back Propagation)(see the algorithm below) to tune α at each iteration. Assuming that $w \in R^n$, the following algorithm tune the step-size for each dimension,

---

for $i = 1 : n$ **do**
  if $g_{FD}^{(i)} g_{prev}^{(i)} > 0$ **then**
    $\alpha_i = 1.2\alpha_i$
    $w_i = w_i + \alpha_i \ \mathrm{sign}(g_{FD}^{(i)})$
    $g_{prev}^{(i)} = g_{FD}^{(i)}$
  else if $g_{FD}^{(i)} g_{prev}^{(i)} < 0$ **then**
    $\alpha_i = 0.5\alpha_i$
    $w_i = w_i + \alpha_i \ \mathrm{sign}(g_{FD}^{(i)})$
    $g_{prev}^{(i)} = 0$
  else
    $w_i = w_i + \alpha_i \ \mathrm{sign}(g_{FD}^{(i)})$
    $g_{prev}^{(i)} = g_{FD}^{(i)}$
  end if
  optionally: cap $\alpha_i \in [\alpha_{\min}, \alpha_{\max}]$
end for

---

where g_{prev} is the value of g_{FD} in the previous iteration. Implementation note:

- Initialize $\alpha_i = 0.5$, $g_{prev}^{(i)} = 0$
- $\alpha_{\min} = 0.01; \alpha_{\max} = 5.0;$

Appendix

The following text describes the dynamics of Cart-Pole (see Sutton's book). Alternatively, you can use OpenAI Gym (CartPole-v1, where the dynamics might be a bit different: discrete actions (you should still use the above algorithm, but only send a signal 0 ($a < 0$) or 1 ($a > 0$) to Gym)).

```
#define GRAVITY 9.8
#define MASSCART 1.0
#define m1 0.1
#define m2 (m1 + MASSCART)
#define LENGTH 0.5   /* actually half the pole's length */
#define POLE (MASSPOLE * LENGTH)
#define FORCE_MAG 10.0
#define TAU 0.02   /* seconds between state updates */
```

Given the current state $(x, \dot{x}, \theta, \dot{\theta})$, a taken action a, a next state is computed (with a small noisy effect) as:

$$
\begin{aligned}
temp &= (a + POLE \times \dot{\theta}^2 \times \sin(\theta))/m2 \\
thetaacc &= (GRAVITY \times \sin(\theta) - \cos(\theta) \times temp)/(LENGTH \times (4/3 - m1 \times \cos(\theta)^2/m2)) \\
xacc &= temp - POLE \times thetaacc \times \cos(\theta)/m2 \\
x_{next} &= x + TAU \times \dot{x} + \epsilon_1 \\
\dot{x}_{next} &= \dot{x} + TAU \times xacc + \epsilon_1 \\
\theta_{next} &= \theta + TAU \times \dot{\theta} + \epsilon_2 \\
\dot{\theta}_{next} &= \dot{\theta} + TAU \times thetaacc + \epsilon_2
\end{aligned}
$$

where ϵ_1 is a Gaussian noise with mean zero and standard deviation 0.01; and ϵ_2 is a Gaussian noise with mean zero and standard deviation 0.0001.