MVA: Reinforcement Learning **TP3**

Reinforcement Learning with Function Approximation

Lecturer: *Matteo Pirotta* *matteo.pirotta@inria.fr*

Deadline: January 5, midnight

Note: Your report should be *short* and based on the answers to questions Q1-Q3.
The code is not optional!

The solution (single archive with name *lastname_firstname_TP3.zip*) should be sent by e-mail to matteo.pirotta@inria.fr, with "[MVA 2017] TP3" as object.

We provide support for Matlab and Python but you can solve the homework using any language. You can choose between the following formats for the solution:

- Report and code: a short report (PDF) based on the answers to questions and the source code.
- Code with comments: integrate answers and comments in the code. For example, you can use a Matlab report or a Python notebook.

1 On-Policy Reinforcement Learning with Parametric Policy

We consider the problem of finding a policy that maximizes the expected discounted reward over a class of parameterized policies $\Pi_\theta = \left\{\pi_\theta : \theta \in \mathbb{R}^d\right\}$, where π_θ is a compact representation of $\pi(a|s,\theta)$.

Policies can be ranked by their expected discounted reward starting from the state distribution μ:

$$J_\mu^{\pi_\theta} = \int_{\mathcal{S}} \mu(s) V^{\pi_\theta}(s) \mathrm{d}s = \frac{1}{1-\gamma} \int_{\mathcal{S}} d_\mu^{\pi_\theta}(s) \int_{\mathcal{A}} \pi_\theta(a|s) R(s,a) \mathrm{d}a\mathrm{d}s,$$

where $d_\mu^{\pi_\theta}(s) = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t Pr(s_t = s|\pi,\mu)$ is the γ–discounted future state distribution for a starting state distribution μ [Sutton et al., 2000].

In these settings, we can use gradient ascent to perform a direct search in the policy space in order to maximize the policy performance. The policy update has the following form:

$$\theta_{t+1} = \theta_t + \Delta_t \left(\nabla_\theta J_\mu^{\pi_\theta}|_{\theta=\theta_t}\right)$$
$$\nabla_\theta J_\mu^{\pi_\theta} = \frac{1}{1-\gamma} \int_{\mathcal{S}} d_\mu^{\pi_\theta}(s) \int_{\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s,a) \mathrm{d}a\mathrm{d}s$$

where $\Delta(x) \in \mathbb{R}^d$ is a transformation of the gradient (see Sec. 1.3).

1.1 Trajectory-based formulation and REINFORCE

Another way of viewing the RL problem is to use the trajectory-based formulation. The policy performance can be equivalently written as

$$J_\mu^{\pi_\theta} = E_{\tau\sim\rho_\mu^{\pi_\theta}}\left[R(\tau)\right] = E_{\tau\sim\rho_\mu^{\pi_\theta}}\left[\sum_{t=0}^{\mathrm{len}(\tau)} \gamma^t R(s_t,a_t)\right]$$

where $\rho_\mu^{\pi_\theta}$ is the distribution induced by the policy (and initial state distribution) over the set of trajectories:

$$\rho_\mu^{\pi_\theta}(\tau) = \mu(s_o) \prod_{k=0}^{\text{len}(\tau)-1} P(s_{k+1}|s_k, a_k)\pi_\theta(a_k|s_k).$$

By differentiation of the policy performance, we obtain

$$\nabla_\theta J_\mu^{\pi_\theta} = E_{\tau\sim\rho_\mu^{\pi_\theta}}\left[\left(\sum_{t=0}^{\text{len}(\tau)} \nabla_\theta \log \pi_\theta(a_t|s_t)\right) R(\tau)\right]$$

Since integrating over all the possible histories is practically unfeasible, the previous gradient can only be estimated, e.g., by means of Monte Carlo simulations as in the REINFORCE algorithm [Williams, 1992] (the main exponent of the likelihood-ratio family):

$$\widehat{\nabla}_\theta J_\mu^{\pi_\theta} = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=0}^{\text{len}(\tau)} \nabla_\theta \log \pi_\theta\left(a_t^{(n)}|s_t^{(n)}\right) R\left(\tau^{(n)}\right).$$

where N is the number of trajectories generated from a system by roll-outs (policy simulations). Note that in non-episodic infinite horizon problems the trajectory is truncated after T steps (you can refer to TP1 for rules on how to set T). The main drawback of REINFORCE is its variance, that is strongly affected by the length of the trajectory horizon T.

Note: REINFORCE is an actor-only method since it does not require to explicitly estimate the Q-function (i.e., it does not store a parametric representation of the policy performance).

1.2 Parametric policy models

The most widespread models are the Gaussian and Gibbs (a.k.a. Bolzmann or Softmax) policies.

Gaussian: the action space is continuous

$$\pi(a|s) = \frac{1}{\sigma_\omega\sqrt{2\pi}(s)} e^{-\frac{(a-\mu_\theta(s))^2}{2\sigma_\omega^2(s)}}$$

then

$$\nabla_\theta \log \pi(a|s) = \frac{(a-\mu_\theta(s))}{\sigma_\omega^2(s)}\nabla_\theta\mu_\theta(s)$$
$$\nabla_\omega \log \pi(a|s) = \frac{(a-\mu_\theta(s))^2 - \sigma_\omega^2(s)}{\sigma_\omega^3(s)}\nabla_\omega\sigma_\omega(s)$$

Gibbs: the action space is discrete (and finite)

$$\pi(a|s) = \frac{e^{\kappa Q_\theta(s,a)}}{\sum_{a'\in\mathcal{A}} e^{\kappa Q_\theta(s,a')}}$$

then

$$\nabla_\theta \log \pi(a|s) = \kappa\nabla_\theta Q_\theta(s,a) - \kappa\sum_{a'\in\mathcal{A}} \pi(a'|s)\nabla_\theta Q_\theta(s,a')$$

1.3 Gradient-based Update Rule

We start considering the standard update rule:

$$\Delta_t(\nabla_\theta J_\mu^{\pi_\theta}) = \alpha_t \nabla_\theta J_\mu^{\pi_\theta} \tag{1}$$

In practice α_t is defined through an annealing schema (i.e. $\alpha_t \to 0$ as $t \to \infty$) or just taken constant.

Recently, due to the success of neural networks, optimization literature has focused on the definition of adaptive gradient methods that automatically tunes the step length or directly alter the gradient direction [e.g. Duchi et al., 2011, Kingma and Ba, 2014]. Here we consider the approach proposed in [Kingma and Ba, 2014]. Adaptive Moment Estimation (Adam) is a method that automatically adapt the step length for each parameter. Adam stores an exponentially decaying average of past squared gradients v_t and an exponentially decaying average of past gradients m_t in order to estimate the first and second moment of gradients.

Algorithm 1: Adam

Input: A gradient estimate g_t and parameters β_1, β_2, ϵ and α.
Result: The increment $\Delta(g_t)$ of the parameters

$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$
$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t \circ g_t$ `// ∘ is the Hadamard (component-wise) product`
$\hat{m}_t = \dfrac{m_t}{1-\beta_1^t}$
$\hat{v}_t = \dfrac{v_t}{1-\beta_2^t}$
$\Delta(g_t) = \dfrac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$

The authors propose default values of 0.9 for β_1, 0.999 for β_2, and 10^{-8} for ϵ. Note that $m_0 = 0$ and $v_0 = 0$. Usual values for the step length α are 0.1 or 0.01.

1.4 Experiments

LQG. In this section, we test the proposed methods on the linear-quadratic Gaussian regulation (LQG) problem. The LQG problem is defined by transition model $s_{t+1} \sim \mathcal{N}(As_t + Ba_t, \Sigma_0^2)$, Gaussian policy $a_t \sim \mathcal{N}(\theta \cdot s_t, \Sigma^2)$ and reward $r_t = -0.5(s_t^T Q s_t + a_t^T B a_t)$. In all our simulations we use $\Sigma_0 = 0$, since all the noise can be modelled on the agent's side without loss of generality. The initial state is drawn uniformly in $[-10, 10]$ and we assume state and action space to be bounded in $[-500, 500]$. We use this task as a testing ground because it is simple and the true optimal parameter θ^* is available as a reference.

In particular, we consider a 1-dimensional problem with $A = 1$, $B = 1$, $Q = 1$ and $R = 1$. We use a discount factor $\gamma = 0.9$, which gives an optimal parameter $\theta^* \approx -0.59$ (just to have a rough idea, the policy performance should be of order $-20/-30$). Consider an horizon $T = 100$.

To analyse the performance of the algorithm you can use directly the value of the parameter θ_t or the policy performance $J_\mu^{\pi_{\theta_t}}$.

Q1: Implement REINFORCE with Gaussian policy model.

We start considering the case of fixed standard deviation $\sigma_\omega(s) = 0.5$. The only parameter to learn is the parametrization of the mean $\mu = \theta s$. Consider the classical gradient updated in (1) and try to play with the value of α_t (constant or an annealing schema). Then, try to use Adam. In order to evaluate the performance during the learning process plot $t \mapsto \theta_t$. Try to play with the different parameters (N, T, σ_ω, ...).

Even in this simple domain may be hard to find the correct configuration for all the parameters of the algorithm. Can you explain a little bit the effect of the parameters α_t (in the case of standard update rule (1)) and N?

Note: that to have a significant result you should average multiple experiments (you can just average over 5 runs of fixed number of iterations). In addition, consider that the optimal parameter θ^* has been computed considering zero as standard deviation (deterministic policy).

Cart Pole. Consider now the classical cart pole problem ($\gamma = 0.9999$). In this case the action set is discrete $\{-1, 1\}$ and the state is described by 4 values. Implement a Gibbs policy with a polynomial function approximation (use a full 3-degree polynomial in (s, a)).

Q2: Show the convergence of the algorithm by using the policy performance $t \mapsto \hat{J}(\theta_t)$. You can try the following parameters: $N = 10$, $T = 100$, $n_{it} = 150$.

In python you can use the method `render` to show the environment.

2 Off-Policy Reinforcement Learning with Value Function Approximation

Among Reinforcement Learning (RL) techniques, value-based methods play an important role. Such methods use function approximation techniques to represent the near optimal value function in domains with large (continuous) state spaces. Approximate Value Iteration (AVI) [Puterman, 1994] is the main class of algorithms able to deal with this scenario and, by far, it is the most analysed in literature.

AVI aims to recover the optimal value function as fixed point of the optimal Bellman operator. Under this perspective, the solution to a RL problem is obtained by solving a sequence of supervised learning problems where, at each iteration, the application of the empirical optimal Bellman operator to the current approximation of the value function is projected in a predefined function space ($\mathcal{F}$). This AVI strategy is called *fitted* value iteration in literature. The idea is that, if enough samples are provided and the function space is sufficiently rich, the fitted function will be a good approximation of the one obtained through the optimal Bellman operator, thus mimicking the behaviour of Value Iteration [Puterman, 1994].

AVI methods are usually *off-policy methods* (i.e., batch methods) since they only exploit samples collected in advance. In other words, over the learning process there is no interaction with the environment and it is not possible to collect additional information. The policies used to collect the samples used for the learning process are called behavioural policies. The performance of AVI methods is correlated to the quality of these policies. In particular, these policies should sufficiently explore the environment (*exploration problem*).

Formally, we denote by $\mathcal{D}_n$ the set of trajectories collected using the behavioural policies. We have mentioned that the AVI exploits the empirical Bellman operator Let $\mathcal{D}_n = \{X_i, A_i, R_i, X_i'\}_{i=1}^n$ be a set of transitions such that $(X_i, A_i) \sim \mu$, $R_i \sim \mathcal{R}(\cdot|X_i, A_i)$ and $X_i' \sim P(\cdot|X_i, A_i)$ and define $H_n = \{(X_1, A_1), \ldots, (X_n, A_n)\}$. The empirical Bellman optimal operator[1] $\hat{T}^* : H_n \to \mathcal{R}^n$ is defined as

$$(\hat{T}^* Q)(X_i, A_i) \triangleq R_i + \gamma \max_{a'} Q(X_i', a').$$

The computation of the empirical Bellman operator requires to solve a maximization problem over the action space. This problem cannot be solved in general when the action space is continuous. In practice, continuous action spaces are *discretized*.

[1] The *Bellman optimal operator* T^* is defined as follows: $(T^*Q)(s, a) \triangleq R(s, a) + \gamma \int_{\mathcal{S}} P(s'|s, a) \max_{a'} Q(s', a') \mathrm{d}s'$. Its fixed point is the *optimal value function* Q^* Puterman [1994].

Algorithm 2: Fitted Q-Iteration

Input: A dataset $\mathcal{D}_n$, $Q_0 = 0$.
Result: $\overline{\pi}(s) = \arg\max_a Q_{K+1}(s,a)$.
for $k = 0, \dots, K$ **do**
  $Q_{k+1} = \arg\inf_{f \in \mathcal{F}} \|f - \hat{T}^* Q_k\|^2_{\mathcal{D}_n}$
end

The whole class of Fitted Q-Iteration (FQI) algorithms is based on the fit of the empirical optimal Bellman operator in $\mathcal{F}$ The correctness of this procedure is guaranteed by $E[\hat{T}^* Q_k(X_i, A_i)|X_i, A_i] = T^* Q_k(X_i, A_i)$.

We are now ready to describe the FQI algorithm (Alg. 2). For any $k \geq 0$, given a dataset $\mathcal{D}_n$ and an estimate Q_k, FQI computes the new estimate as follows:

$$Q_{k+1} \in \arg\inf_{f \in \mathcal{F}} \|f - \hat{T}^* Q_k\|^2_{\mathcal{D}_n} = \arg\inf_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \left| f(X_i, A_i) - (\hat{T}^* Q_k)(X_i, A_i) \right|^2.$$

2.1 Experiments

We consider the LQG problem defined in the previous section. Since FQI requires discrete actions in order to evaluate the empirical Bellman operator, we discretize the action space: $\overline{\mathcal{A}}$. In order to generate the dataset $\mathcal{D}_n$ you should use a policy selecting actions in $\overline{\mathcal{A}}$ with uniform probability.

In order to approximate the Q-function you can use a linear approximation with features corresponding to a second-order polynomial in (s, a):

$$\mathcal{F} = \{\phi(s,a)^T \theta | \theta \in \mathbb{R}^3\},$$

where $\phi(s,a) = [a, sa, s^2 + a^2]$.

Note that by using a linear approximator, the fitting problem corresponds to a simple linear least squares problem (with or without regularization).

$$\theta_{k+1} \in \arg\min_{\theta} \sum_{i=1}^{n} \Big(\phi(s_i, a_i)\theta - \hat{T}^* Q_k(s_i, a_i) \Big)^2 + \lambda \|\theta\|_2^2$$

that can be solved in closed form as

$$\theta_{k+1} = (Z_t^\intercal Z_t + \lambda I)^{-1} Z_t^\intercal y_t$$

where $Z_t = [\phi(s_1, a_1)^\intercal; \dots; \phi(s_n, a_n)^\intercal] \in \mathbb{R}^{n \times d}$ and $y_t = [\hat{T}^* Q_k(s_1, a_1); \dots; \hat{T}^* Q_k(s_n, a_n)] \in \mathbb{R}^n$.

Q3: implement FQI with linear function approximation. The provided code contains the skeleton for the generation of the dataset but you have to define the behavioural policy. To evaluate the performance of the algorithm, after each update, you should simulate the policy for 50 episodes of length 50 in order to estimate $J(\pi_k)$. Plot $k \mapsto J(\pi_k)$.

References

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1994. ISBN 0471619779.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.