Robot Learning
Weekly Exercise 5

Marc Toussaint & Wolfgang Hönig
Learning & Intelligent Systems Lab, Intelligent Multi-Robot Coordination Lab, TU Berlin
Marchstr. 23, 10587 Berlin, Germany

Summer 2024

1 Literature: SAC

The following paper introduces *Soft Actor-Critic*, a state-of-the art RL method that integrates many good ideas that have been discovered over the last decade into a rather clean algorithm:

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870, 2018. URL: https://proceedings.mlr.press/v80/haarnoja18b

a) First some bug hunting:
   - In the Supplementary Material, Appendix A., Equation (14), there is a notational bug. Can you find it?
   - In the main paper, going from Eq. (5) to (6), I think there is another bug. Can you find it?
   - The line below (6) states "where the actions are sampled" – can you explain where actions are sampled?
   - **Idea for another exercise:** In the paper the authors state that the gradient of the policy parameters could be estimated using the REINFORCE / likelihood ratio gradient estimator. The students could derive this one, or show that the reparametrization one has lower variance.
     This would link ex 1 and 2 nicely.

in (14): There is an expectation $\mathbb{E}_{s_l,a_l}\{\cdot\}$ but s_l, a_l nowhere used. That must be a bug, just syntactically. Solution: It should be $[r(s_l, a_l) + \alpha\mathcal{H}(\pi(\cdot|s_l)) \,|\, s_t, a_t]$.

Eq (5) has an $\mathbb{E}_{a_t}\{\cdot\}$, which is lost in (6). It should be
$\cdots (V_\psi(s_t) - \mathbb{E}_{a_t\sim\pi_\phi}\{Q_\theta(s_t, a_t) - \log \pi_\phi(a_t|s_t)\})$

That also explains what "actions are sampled" means. (Maybe they dropped the $\mathbb{E}_{a_t\sim\pi_\phi}\{\}$ because they wanted to save space and then put in the sentence below the "where actions are sampled"..? But that's crazy.)

b) Now the core question: In Alg. 1 lower part you find three lines to train the parameters ψ, θ_i, ϕ, as well as a low-pass filter for $\bar\psi$.
   - Find out which functions these parameters parameterize.
   - Find out where these parameters are used during training, i.e., the inter-dependencies of training: For instance, when ϕ is trained, does that depend on ψ? Answer this for all parameters ψ, θ_i, ϕ.

V_ψ: the value function, $Q_{\theta_{1,2}}$ the double Q-functions, π_ϕ the policy

training V_ψ depends on $\theta_{1,2}$ (both, taking min, as explained on the right) and ϕ

training Q_{θ_i} depends only on the low-pass of ψ! $\to$ very stable

training π_ϕ depends on $\theta_{1,2}$ (both, taking min) but not ψ

2 The Reparametrization Trick

We typically write a conditional density as $p(x|y)$. If that depends on parameters (to be trained), we may write this as $p_\theta(x|y)$ or $p(x|y;\theta)$.

The reparametrization trick states that any (conditional) distribution $p(x|y;\theta)$ can instead be represented as a deterministic function $x = f(y, \epsilon; \theta),\ \epsilon \sim p(\epsilon)$.

a) Given a Gaussian distribution $p_\theta(x) = \mathcal{N}(x|\mu, \Sigma)$ with parameters $\theta = (\mu, \Sigma)$, $\mu \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$, how can you rewrite this as deterministic $x = f_\theta(\epsilon)$ with $\epsilon \sim \mathcal{N}(0, \mathbf{I}_n), \epsilon \in \mathbb{R}^n$?

First, $\epsilon \sim \mathcal{N}(0, 1)$ is n-dim Gaussian. (Sorry if that was not clear.)

Given $\epsilon \sim \mathcal{N}(0, 1)$, we have (for invertible C):

$$\mathcal{N}(\epsilon,\ 0;\ \mathbf{I}) = |C|\ \mathcal{N}(C\epsilon,\ 0;\ CC^\top) = |C|\ \mathcal{N}(C\epsilon + \mu,\ \mu;\ CC^\top)\ . \tag{1}$$

Therefore $C\epsilon + \mu$ is distributed Gaussian with mean μ and covariance matric $CC^\top$. Therefore, define $f_\theta(\epsilon) = C\epsilon + \mu$ where C is the Cholesky decomp of Σ.

Note, this way of manipulating Gaussians is perhaps not common. You really think of $\mathcal{N}(x, a; A) = \frac{1}{|2\pi\Sigma|^{1/2}} \exp\{-\frac{1}{2}(x - a)^\top A^{-1}(x - a)\}$ as just an expression, and the above equalities clearly hold for this expression.

b) Given discrete (aka. categorical) distribution $p(x)$ over a discrete $x \in \{1, .., M\}$. How can you rerepresent sampling $x \sim p(x)$ as a deterministic function $x = f(\epsilon)$ with $\epsilon \sim \mathcal{U}[0, 1]$ uniformly in the real inverval $[0, 1]$?

Let $F(z) = p(x \le z)$ be the accumulated distribution (e.g., $F(M) = 1$, $F(1) = p(1)$. Think of $F(z)$ as partitioning the interval $[0, 1]$ in M segments, each with size $p(z)$. Then define $f(\epsilon) = \min\{z \in \{1, .., M\} : F(z) \ge \epsilon\}$, i.e., the smallest integer z such that $F(z) \ge \epsilon$.

[This is called a "trick" in a particular context: Sometimes there is a sampling step within an architecture, i.e., within a computation graph. E.g. $x \mapsto z \sim p_\theta(z|x)$, $z \mapsto y = g_\theta(z)$, which is a VAC example, where the latent variable z is sampled in the "middle" of the architecture. Gradients in principle don't propagate *through* a sampling operation, and standard training would not be possible. But representing this as $x \mapsto z = f_\theta(x, \epsilon)$, $z \mapsto y = g_\theta(z)$ with the sampling $\epsilon \sim p(\epsilon)$ done *outside the architecture*, gradients flow through f and g as usual, and the training process has to sample ϵ's as if it was data.]

3 Mountain Car RL using SAC

Use the SAC implementation in Stable Baselines3 to solve the Continuous Mountain Car problem: https://stable-baselines3.readthedocs.io/en/master/modules/sac.html.

a) First, run SAC off-the-shelf, with default parameters using the example code provided on the above URL. In the tutorial, be able to demonstrate the final policy: Run multiple test rollouts, and compute the discounted total return (directly from the reward observations) for each test rollout.

b) Monitoring the training process is generally important in RL. Follow https://stable-baselines3.readthedocs.io/en/master/guide/examples.html#callbacks-monitoring-training to plot the training process (and generally learn about the Callback mechanism).

c) The SAC method has a ton of parameters. Try:
   - Fixing `ent_coef` to one particular value (e.g. 10; or check the SAC paper for common choices), and report on the difference.
   - The discounting factor `gamma`, e.g. to $\gamma = 0.999$.
   - The network architecture (by default `net_arch = [256, 256]`). You'll have to look into code to understand the parameter, esp. the `get_actor_critic_arch` method in https://github.com/DLR-RM/stable-baselines3/blob/master/stable_baselines3/common/torch_layers.py. Try smaller networks.

References

[1] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870, 2018. URL: https://proceedings.mlr.press/v80/haarnoja18b.