Reinforcement Learning with Deep Energy-Based Policies

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, Sergey Levine

ICML2017

May 14,2019

Outline

Introduction

Deep RL training is sensitive to ...

- randomness in the environment
- initialization of the policy
- the algorithm implementation

Example

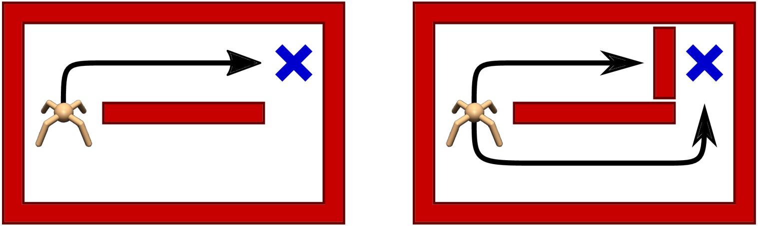

Figure 1: A robot navigating a maze

Introduction

RL employs a (stochastic) policy (π) to select actions, and finds the best policy that maximizes the cumulative reward it collects throughout an episode of length T.

$$\pi^* = \text{argmax}_\pi \, \mathbb{E}_\pi \left[\sum_{t=0}^{T} r_t \right]$$

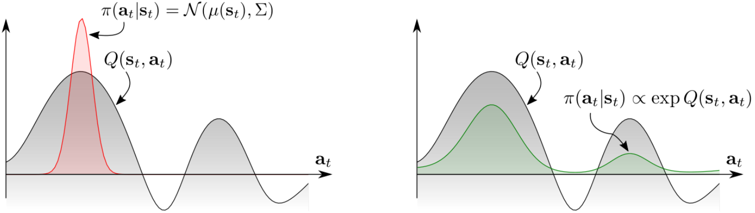

Figure 2: A multimodal Q-function.

Introduction

Energy Based Model and Boltzmann Distribution and Softmax

- Energy Based Model: $P(x) = \frac{e^{f(x)}}{\sum_{x \in X} e^{f(x)}}$
- Boltzmann Distribution : $P(x) = \frac{e^{-E(x)}}{\sum_{x \in X} e^{-E(x)}}$
- Softmax function : $\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$

In practice, we prefer maximum-entropy models as they assume the least about the unknowns while matching the observed information.

Preliminaries

Maximum Entropy RL

the state and state-action marginals of the trajectory distribution induced by a policy $\pi(a_t|s_t)$ are:

$$\rho_\pi\left(\mathbf{s}_t\right) \text{ and } \rho_\pi\left(\mathbf{s}_t, \mathbf{a}_t\right)$$

the standard RL objective is:

$$\pi_{\mathrm{std}}^* = \arg\max_\pi \sum \mathbb{E}_{\left(\mathbf{s}_t, \mathbf{a}_t\right) \sim \rho_\pi}\left[r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right]$$

the maximum entropy RL objective is:

$$\pi_{\mathrm{MaxEnt}}^* = \arg\max_\pi \sum_t \mathbb{E}_{\left(\mathbf{s}_t, \mathbf{a}_t\right) \sim \rho_\pi}\left[r\left(\mathbf{s}_t, \mathbf{a}_t\right) + \alpha \mathcal{H}\left(\pi\left(\cdot|\mathbf{s}_t\right)\right)\right]$$

Preliminaries

Energy-Based Models

soft Q-function is:

$$Q^{*}_{\text{soft}}\left(\mathbf{s}_t, \mathbf{a}_t\right) = r_t + \mathbb{E}_{(\mathbf{s}_{t+1},\ldots)\sim\rho_\pi}\left[\sum_{l=1}^{\infty}\gamma^{l}\left(r_{t+l} + \alpha\mathcal{H}\left(\pi^{*}_{\text{MaxEnt}}\left(\cdot|\mathbf{s}_{t+l}\right)\right)\right)\right]$$

and it satisfies the soft Bellman equation:

$$Q^{*}_{\mathrm{soft}}\left(\mathbf{s}_t, \mathbf{a}_t\right) = r_t + \gamma\mathbb{E}_{\mathbf{s}_{t+1}\sim p_{\mathrm{s}}}\left[V^{*}_{\mathrm{soft}}\left(\mathbf{s}_{t+1}\right)\right],$$

where the soft value function is:

$$V^{*}_{\mathrm{soft}}\left(\mathbf{s}_t\right) = \alpha\log\int_{\mathcal{A}}\exp\left(\frac{1}{\alpha}Q^{*}_{\mathrm{soft}}\left(\mathbf{s}_t, \mathbf{a}'\right)\right)d\mathbf{a}'$$

the optimal policy is:

$$\pi^{*}_{\mathrm{MaxEnt}}\left(\mathbf{a}_t|\mathbf{s}_t\right) = \exp\left(\frac{1}{\alpha}\left(Q^{*}_{\mathrm{soft}}\left(\mathbf{s}_t, \mathbf{a}_t\right) - V^{*}_{\mathrm{soft}}\left(\mathbf{s}_t\right)\right)\right)$$

Soft Q-Learning

q-learning: $Q(s_t, a_t) \leftarrow r_t + \gamma \max_a Q(s_{t+1}, a)$
soft q-learning: $Q_{\mathrm{soft}}(\mathbf{s}_t, \mathbf{a}_t) \leftarrow r_t + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim p_{\mathbf{s}}} \left[V_{\mathrm{soft}}(\mathbf{s}_{t+1}) \right], \forall \mathbf{s}_t, \mathbf{a}_t$
$V_{\mathrm{soft}}(\mathbf{s}_t) \leftarrow \alpha \log \int_{\mathcal{A}} \exp\left(\frac{1}{\alpha} Q_{\mathrm{soft}}(\mathbf{s}_t, \mathbf{a}') \right), d\mathbf{a}', \forall \mathbf{s_t}$

In fact, the q-learning uses hard-max of Q, the soft q-learning uses soft-max of Q.

In continuous domains, there are two major challenges of soft-q-learning.

- First, exact dynamic programming is infeasible
- Second, the optimal policy is defined by an intractable energy-based distribution, which is difficult to sample from.

Soft Q-Learning

Integrating problem

we use samples from rollouts of the current policy, and use sample sum instead of integrating.

- Model the soft Q-function with a function approximator, and denote it as $Q^{\theta}_{\text{soft}}\left(\mathbf{s}_t, \mathbf{a}_t\right)$
- Express the soft value function in terms of an expectation via importance sampling:

$$V^{\theta}_{\mathrm{soft}}\left(\mathbf{s}_t\right) = \alpha \log \mathbb{E}_{q_{\mathrm{a}'}}\left[\frac{\exp\left(\frac{1}{\alpha} Q^{\theta}_{\mathrm{soft}}\left(\mathbf{s}_t, \mathbf{a}'\right)\right)}{q_{\mathrm{a}'}\left(\mathbf{a}'\right)}\right]$$

- Minimize the objective:

$$J_Q(\theta) = \mathbb{E}_{\mathbf{s}_t \sim q_{\mathbf{s}_t}, \mathbf{a}_t \sim q_{\mathbf{a}_t}}\left[\frac{1}{2}\left(\hat{Q}^{\bar{\theta}}_{\mathrm{soft}}\left(\mathbf{s}_t, \mathbf{a}_t\right) - Q^{\theta}_{\mathrm{soft}}\left(\mathbf{s}_t, \mathbf{a}_t\right)\right)^2\right]$$

Soft Q-Learning

Sample problem

We use the amortized Stein variational gradient descent(SVGD) to to train an inference network to generate approximate samples

- Construct a state-conditioned stochastic neural network $\mathbf{a}_t = f^{\phi}(\xi; \mathbf{s}_t)$
- Minimize the distance between the induced distribution and the energy-based distribution:

$$J_\pi(\phi; \mathbf{s}_t) = \mathrm{D_{KL}}\left(\pi^\phi(\cdot|\mathbf{s}_t) \,\|\, \exp\left(\frac{1}{\alpha}\left(Q^\theta_{\mathrm{soft}}(\mathbf{s}_t, \cdot) - V^\theta_{\mathrm{soft}}\right)\right)\right)$$

Soft Q-Learning

We can use the chain rule and backpropagate the Stein variational gradient into the policy network according to

$$\frac{\partial J_\pi(\phi; \mathbf{s}_t)}{\partial \phi} \propto \mathbb{E}_\xi \left[\Delta f^\phi(\xi; \mathbf{s}_t) \frac{\partial f^\phi(\xi; \mathbf{s}_t)}{\partial \phi} \right]$$

where $\Delta f^\phi(\cdot; \mathbf{s}_t)$ is Stein variational gradient descent(SVGD):

$$\begin{aligned}\Delta f^\phi(\cdot; \mathbf{s}_t) = & \mathbb{E}_{\mathbf{a}_t \sim \pi^\phi} \left[\kappa\left(\mathbf{a}_t, f^\phi(\cdot; \mathbf{s}_t)\right) \nabla_{\mathbf{a}'} Q^\theta_{\mathbf{soft}}\left(\mathbf{s}_t, \mathbf{a}'\right)\Big|_{\mathbf{a}'=\mathbf{a}_t} \right. \\ & \left. + \alpha \nabla_{\mathbf{a}'} \kappa\left(\mathbf{a}', f^\phi(\cdot; \mathbf{s}_t)\right)\Big|_{\mathbf{a}'=\mathbf{a}_t} \right],\end{aligned}$$

and κ is a kernel function.

Besides, we can use any gradient-based optimization method to learn the optimal sampling network parameters.

Algorithm

Algorithm 1 Soft Q-learning

$\theta, \phi \sim$ some initialization distributions.
Assign target parameters: $\bar{\theta} \leftarrow \theta$, $\bar{\phi} \leftarrow \phi$.
$\mathcal{D} \leftarrow$ empty replay memory.

for each epoch **do**
  for each t **do**
    Collect experience
      Sample an action for $\mathbf{s}_t$ using f^ϕ:
        $\mathbf{a}_t \leftarrow f^\phi(\xi; \mathbf{s}_t)$ where $\xi \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.
      Sample next state from the environment:
        $\mathbf{s}_{t+1} \sim p_\mathbf{s}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$.
      Save the new experience in the replay memory:
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$.
    Sample a minibatch from the replay memory
      $\{(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)}, r_t^{(i)}, \mathbf{s}_{t+1}^{(i)})\}_{i=0}^N \sim \mathcal{D}$.
    Update the soft Q-function parameters
      Sample $\{\mathbf{a}^{(i,j)}\}_{j=0}^M \sim q_{\mathbf{a}'}$ for each $\mathbf{s}_{t+1}^{(i)}$.
      Compute empirical soft values $\hat{V}_{\mathrm{soft}}^{\bar{\theta}}(\mathbf{s}_{t+1}^{(i)})$ in (10).
      Compute empirical gradient $\hat{\nabla}_\theta J_Q$ of (11).
      Update θ according to $\hat{\nabla}_\theta J_Q$ using ADAM.
    Update policy
      Sample $\{\xi^{(i,j)}\}_{j=0}^M \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ for each $\mathbf{s}_t^{(i)}$.
      Compute actions $\mathbf{a}_t^{(i,j)} = f^\phi(\xi^{(i,j)}, \mathbf{s}_t^{(i)})$.
      Compute Δf^ϕ using empirical estimate of (13).
      Compute empiricial estimate of (14): $\hat{\nabla}_\phi J_\pi$.
      Update ϕ according to $\hat{\nabla}_\phi J_\pi$ using ADAM.
  end for
  if epoch *mod* update_interval $= 0$ **then**
    Update target parameters: $\bar{\theta} \leftarrow \theta$, $\bar{\phi} \leftarrow \phi$.
  end if
end for

Experiments

Multi-Goal Environment

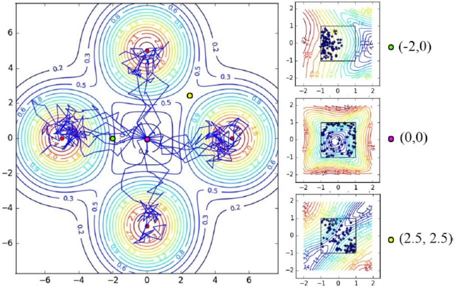

Figure 3: Illustration of the 2D multi-goal environment

Experiments

Task-specific initialization

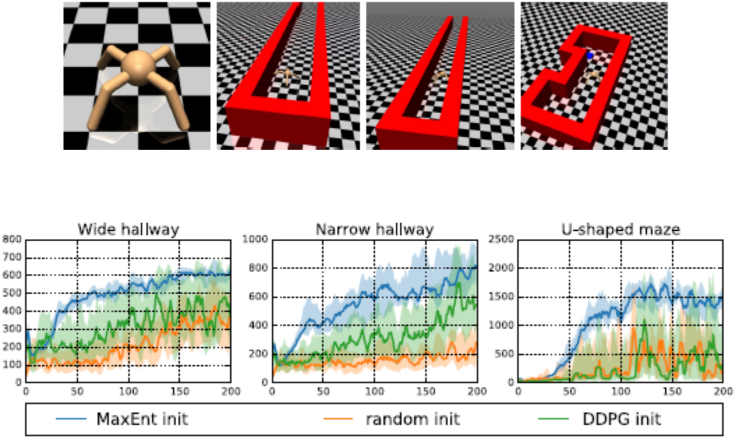

Figure 4: Performance in the downstream task with fine-tuning (MaxEnt) or training from scratch (DDPG).

Conclusion

- Energy-based Model and Softmax
- Use Importance Sampling to deal with Integrating problem
- Construct sample network

Thanks