Week 7: Policy Optimization II: Actor-Critic and Natural Policy Gradient

Bolei Zhou

The Chinese University of Hong Kong

October 19, 2020

Announcement

1. Lecturer's office hour is switched to Monday after class (11:15 am to 12:00 pm)
2. Tomorrow TA office hour will be the Assignment 2 review session

Today's Plan

1. Review on policy gradient
2. Actor critic and advantage actor critic
3. Natural policy gradient
4. Sample code

Review on Policy Gradient

1. In policy optimization, for the policy function π_θ the objective is to maximize

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

   1. $R(\tau)$ could be any reasonable reward function on τ
2. So the gradient is

$$\begin{aligned}\nabla_\theta J(\theta) =& \nabla_\theta \sum_\tau P(\tau;\theta) R(\tau) \\ =& \mathbb{E}_\tau[R(\tau) \nabla_\theta \log P(\tau;\theta)]\end{aligned}$$

   1. **log likelihood trick**: $\nabla_\theta f_\theta(x) = f_\theta(x) \nabla_\theta \log f_\theta(x)$
   2. Trajectory distribution: $P(\tau;\theta) = \mu(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$
   3. **canceling trick**: $\nabla_\theta \log P(\tau;\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)$

Review on Policy Gradient

1. The gradient is

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_\tau P(\tau;\theta) R(\tau) = \mathbb{E}_\tau [R(\tau) \nabla_\theta \log P(\tau;\theta)]$$

   1. After decomposing τ we derive that

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \Big[\Big(\sum_{t=0}^{T-1} r_t \Big) \Big(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \Big) \Big]$$

   2. After applying the causality, we have

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \Big[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \Big]$$

Reduce Variance by a Baseline

1. The original update

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \Big[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \Big]$$

   1. $G_t = \sum_{t'=t}^{T-1} r_{t'}$ is the return for a Monte-Carlo trajectory which might have high variance

2. We subtract a baseline $b(s)$ from the actual return of the policy gradient to reduce variance

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \Big[\sum_{t=0}^{T-1} \big(G_t - b(s_t)\big) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \Big]$$

3. A good baseline is the expected return

$$b(s_t) = \mathbb{E}[r_t + r_{t+1} + ... + r_{T-1}]$$

Reduce Variance by a Baseline

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \Big[\sum_{t=0}^{T-1} \big(G_t - b_{\mathbf{w}}(s_t) \big) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \Big]$$

1. Baseline $b(s)$ can reduce variance, without changing the expectation
2. $b_{\mathbf{w}}(s)$ also has a parameter $\mathbf{w}$ to learn so that we have two set of parameters $\mathbf{w}$ and θ

Vanilla Policy Gradient Algorithm with Baseline

procedure POLICY GRADIENT(α)

Initialize policy parameters θ and baseline values $b(s)$ for all s, e.g. to 0

for iteration $= 1, 2, \ldots$ **do**

Collect a set of m trajectories by executing the current policy π_θ

for each time step t of each trajectory $\tau^{(i)}$ **do**

Compute the *return* $G_t^{(i)} = \sum_{t'=t}^{T-1} r_{t'}$

Compute the *advantage estimate* $\hat{A}_t^{(i)} = G_t^{(i)} - b(s_t)$

Re-fit the baseline to the empirical returns by updating $\mathbf{w}$ to minimize

$$\sum_{i=1}^{m} \sum_{t=0}^{T-1} \| b(s_t) - G_t^{(i)} \|^2$$

Update policy parameters θ using the policy gradient estimate $\hat{g}$

$$\hat{g} = \sum_{i=1}^{m} \sum_{t=0}^{T-1} \hat{A}_t^{(i)} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

with an optimizer like SGD ($\theta \leftarrow \theta + \alpha \cdot \hat{g}$) or Adam

return θ and baseline values $b(s)$

Reduce Variance Using a Critic

1. The update is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\Big]$$

2. In practice, G_t is a sample from Monte Carlo policy gradient, which is the unbiased but noisy estimate of $Q^{\pi_\theta}(s_t, a_t)$
3. Instead we can use a critic to estimate the action-value function,

$$Q_{\mathbf{w}}(s, a) \approx Q^{\pi_\theta}(s, a)$$

4. Then the update becomes

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{T-1} Q_{\mathbf{w}}(s_t, a_t) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\Big]$$

Actor-Critic Policy Gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{T-1} Q_{\mathbf{w}}(s_t, a_t) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\Big]$$

1. It becomes Actor-Critic Policy Gradient
   1. Actor: the policy function used to generate the action
   2. Critic: the value function used to evaluate the reward of the actions
2. Actor-critic algorithms maintain two sets of parameters
   1. Actor: Updates policy parameters θ, in direction suggested by critic
   2. Critic: Updates action-value function parameters $\mathbf{w}$

Estimate the Action-Value Function

1. Estimating the critic is solving a familiar problem: policy evaluation
   1. How good is policy π_θ under current parameter θ
2. Policy evaluation was explored in previous lectures, e.g.
   1. Monte-Carlo policy evaluation
   2. Temporal-Difference learning
   3. Least-squares policy evaluation

Action-Value Actor-Critic Algorithm

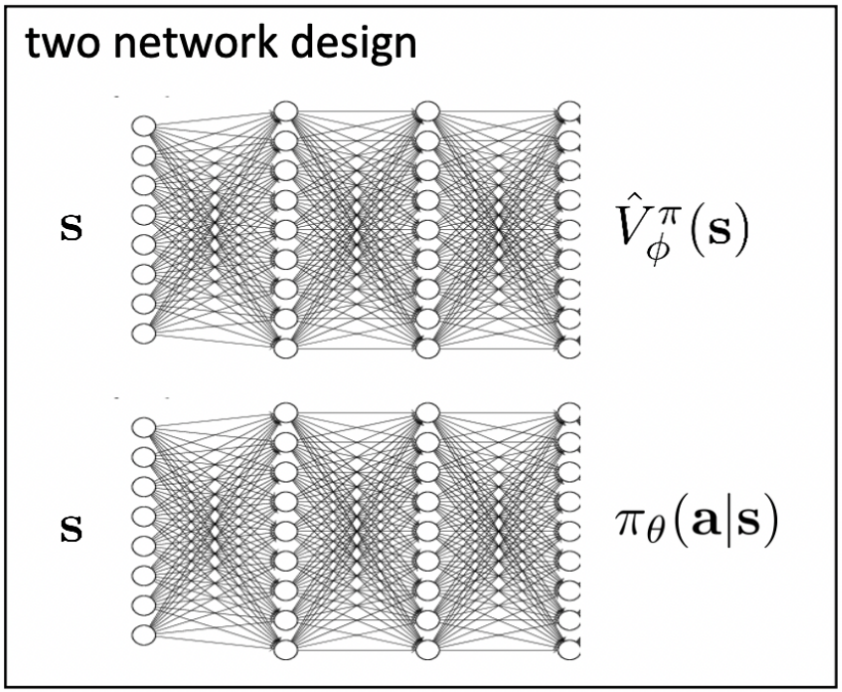
1. Using a linear value function approximation: $Q_{\mathbf{w}}(s,a) = \psi(s,a)^T \mathbf{w}$
   1. Critic: update $\mathbf{w}$ by a linear TD(0)
   2. Actor: update θ by policy gradient

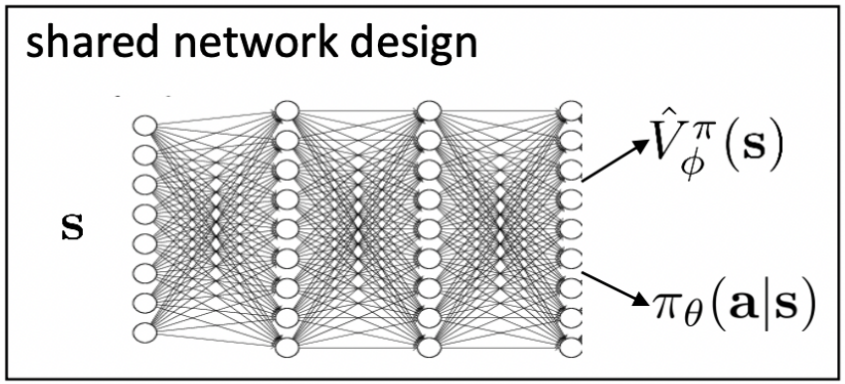
Algorithm 1 Simple QAC

for each step **do**
  generate sample s, a, r, s', a' following π_θ
  $\delta = r + \gamma Q_{\mathbf{w}}(s', a') - Q_{\mathbf{w}}(s, a)$ #TD error
  $\mathbf{w} \leftarrow \mathbf{w} + \beta \delta \psi(s, a)$
  $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_{\mathbf{w}}(s, a)$
end for

Actor-Critic Function Approximators

1. We can have two separate functions to approximate value function and policy function, or use a shared network design (feature extraction is shared but output two heads), as below:





Compatible Function Approximation

Theorem (Compatible Function Approximation Theorem)

If the following two conditions are satisfied:

1. *Value function approximator is compatible to the policy*

$$\nabla_w Q_w(s,a) = \nabla_\theta \log \pi_\theta(s,a)$$

2. *Value function parameters w minimize the mean-squared error*

$$\epsilon(w) = E_{\pi_\theta}[(Q^{\pi_\theta}(s,a) - Q_w(s,a))^2]$$

then the policy gradient is exact

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[Q_w(s,a)\nabla_\theta \log \pi_\theta(s,a)]$$

RS Sutton, et al. Policy gradient methods for reinforcement learning with function approximation. NIPS'01

Compatible Function Approximation

1. Given a parameterized policy, compatible function approximation theorem can be used to derive an appropriate form of the value-function parameterization.
   1. For example, consider a softmax policy as Gibbs distribution where ϕ_{sa} is the feature vector for state-action pair s,a

$$\pi_\theta(s,a) = \frac{e^{\theta^T \phi_{sa}}}{\sum_b e^{\theta^T \phi_{sb}}} \tag{1}$$

   2. Then meeting the compatibility condition requires

$$\nabla_w Q_w(s,a) = \nabla_\theta \log \pi_\theta(s,a) = \phi_{sa} - \sum_b \pi_\theta(s,b)\phi_{sb}$$

   3. so the compatible parameterization of $Q_w(s,a)$ is

$$Q_w(s,a) = w^T \left(\phi_{sa} - \sum_b \pi_\theta(s,b)\phi_{sb} \right) = w^T \nabla_\theta \log \pi_\theta(s,a)$$

Reduce the Variance of Actor-Critic by a Baseline

1. Recall Q-function / state-action-value function:

$$Q^{\pi,\gamma}(s,a) = \mathbb{E}_{\pi}[r_1 + \gamma r_2 + ...|s_1 = s, a_1 = a]$$

2. State value function can serve as a great baseline

$$\begin{aligned} V^{\pi,\gamma}(s) =& \mathbb{E}_{\pi}[r_1 + \gamma r_2 + ...|s_1 = s] \\ =& \mathbb{E}_{a\sim\pi}[Q^{\pi,\gamma}(s,a)] \end{aligned}$$

3. Advantage function: combining Q with baseline V

$$A^{\pi,\gamma}(s,a) = Q^{\pi,\gamma}(s,a) - V^{\pi,\gamma}(s)$$

4. Then the policy gradient becomes:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi,\gamma}(s,a)]$$

Advantage Actor-Critic

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)]$$

1. Critic should estimate the advantage function
2. Use two function approximators and two sets of parameter vectors,

$$V_{\mathbf{v}}(s) \approx V^{\pi}(s)$$
$$Q_{\mathbf{w}}(s,a) \approx Q^{\pi}(s,a)$$

3. Update both parameters by TD learning or MC

Advantage Actor-Critic

1. For the true value function $V^{\pi_\theta}(s)$, the TD error δ^{π_θ}

$$\delta^{\pi_\theta} = r(s,a) + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$

is an unbiased estimate of the advantage function

$$\begin{aligned}\mathbb{E}_{\pi_\theta}[\delta^{\pi_\theta}|s,a] =& \mathbb{E}_{\pi_\theta}[r + \gamma V^{\pi_\theta}(s')|s,a] - V^{\pi_\theta}(s)\\ =& Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)\\ =& A^{\pi_\theta}(s,a)\end{aligned}$$

2. So we can use the TD error to compute the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\delta^{\pi_\theta}]$$

3. It requires only one set of critic parameter κ that is estimated by TD

$$\delta_v = r + \gamma V_\kappa(s') - V_\kappa(s)$$

N-step estimators

1. We used the Monte-Carlo estimates of the reward
2. We can also us TD methods for the policy gradient update, or any intermediate blend between TD and MC methods:
3. Consider the following n-step returns for $n = 1, 2, \infty$

$$
\begin{aligned}
n = 1 (TD) \quad & G_t^{(1)} = r_{t+1} + \gamma v(s_{t+1}) \\
n = 2 \quad & G_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 v(s_{t+2}) \\
n = \infty (MC) \quad & G_t^{(\infty)} = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{T-t-1} r_T
\end{aligned}
$$

4. Then the advantage estimators become

$$
\begin{aligned}
\hat{A}_t^{(1)} &= r_{t+1} + \gamma v(s_{t+1}) - v(s_t) \\
\hat{A}_t^{(2)} &= r_{t+1} + \gamma r_{t+2} + \gamma^2 v(s_{t+2}) - v(s_t) \\
\hat{A}_t^{(\infty)} &= r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{T-t-1} r_T - v(s_t)
\end{aligned}
$$

$\hat{A}^{(1)}$ has low variance and high bias. $\hat{A}^{(\infty)}$ has high variance but low bias

Critic at Different Time-Scales

1. Critic can estimate linear value function $V_\kappa(s) = \psi(s)^T \kappa$ from many targets as follows
2. For MC, the update is

$$\Delta\kappa = \alpha(G_t - V_\kappa(s))\psi(s)$$

3. For TD(0), the update is

$$\Delta\kappa = \alpha(r + \gamma V_\kappa(s') - V_\kappa(s))\psi(s)$$

4. For k-step return, the update is

$$\Delta\kappa = \alpha(\sum_{i=0}^{k} \gamma^i r_{t+i} + \gamma^k V_\kappa(s_{t+k}) - V_\kappa(s_t))\psi(s)$$

Actors at Different Time-Scales

1. The policy gradient can also be estimated at many time-scales

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) A^{\pi_\theta}(s, a)]$$

2. Monte-Carlo Actor-Critic policy gradient uses error from complete return

$$\Delta\theta = \alpha(G_t - V_\kappa(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

3. TD Difference Actor-Critic policy gradient uses TD error

$$\Delta\theta = \alpha(r + \gamma V_\kappa(s_{t+1}) - V_\kappa(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

4. k-step return Actor-Critic policy gradient uses the k-step return error

$$\Delta\theta = \alpha(\sum_{i=0}^{k} \gamma^i r_{t+i} + \gamma^k V_\kappa(s_{t+k}) - V_\kappa(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

Summary of Policy Gradient Algorithms

1. The policy gradient has many forms

$$\begin{aligned}\nabla_\theta J(\theta) =& \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) G_t] \text{ — REINFORCE} \\ =& \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q^{\mathbf{w}}(s,a)] \text{ — Q Actor-Critic} \\ =& \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\mathbf{w}}(s,a)] \text{— Advantage Actor-Critic} \\ =& \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \delta] \text{— TD Actor-Critic}\end{aligned}$$

2. Critic uses policy evaluation (e.g. MC or TD learning) to estimate $Q^\pi(s,a)$, $A^\pi(s,a)$, or $V^\pi(s,a)$

Natural Policy Gradient

1. Policy gradient is the steepest ascend in **parameter space** with Euclidean metric:

$$d^* = \nabla_\theta J(\theta) = \lim_{\epsilon \to 0} \frac{1}{\epsilon} \arg\max J(\theta + d), s.t. ||d|| \leq \epsilon$$

   1. Drawback: it is sensitive to the parameterization of the policy function

2. Steepest ascend in **distribution space (policy output)** with KL-divergence as constraint

$$d^* = \arg\max J(\theta + d), s.t. KL[\pi_\theta || \pi_{\theta+d}] = c \tag{2}$$

   1. Fixing KL-divergence as a constant c make sure we move along the distribution space with constant speed, regardless the curvature (thus robust to the reparametrization of the model as we only care about the distribution induced by the parameter)

KL-Divergence as Metric for Two Distributions

1. KL-divergence measure the "closeness" of two distributions.

$$KL[\pi_\theta || \pi_{\theta'}] = E_{\pi_\theta}[\log \pi_\theta] - E_{\pi_\theta}[\log \pi_{\theta'}]$$

2. Although as KL-divergence is non-symmetric and thus not a true metric, we can use it anyway. This is because as d goes to zero, KL-divergence is asymptotically symmetric. So, within a local neighbourhood, KL-divergence is approximately symmetric
3. We can prove that the second-order Taylor expansion of KL-divergence is

$$KL[\pi_\theta || \pi_{\theta+d}] \approx \frac{1}{2} d^T F d$$

where F is the **Fisher Information Matrix** as second order derivative of KL divergence $E_{\pi_\theta}[\nabla \log \pi_\theta \nabla \log \pi_\theta^T]$

Natural Policy Gradient

$$d^* = \arg\max J(\theta + d), s.t. KL[\pi_\theta || \pi_{\theta+d}] = c$$

1. Write the above as Lagrangian form, with the $J(\theta + d)$ approximated by its first order Tyler expansion and the constraint KL-divergence approximated by its second order Tyler expansion

$$\begin{aligned} d^* &= \arg\max_d J(\theta + d) - \lambda(KL[\pi_\theta || \pi_{\theta+d}] - c) \\ &\approx \arg\max_d J(\theta) + \nabla_\theta J(\theta)^T d - \frac{1}{2}\lambda d^T F d + \lambda c \end{aligned}$$

2. To solve this maximization we set its derivative wrt. d to zero, then we have the natural policy gradient: $d = \frac{1}{\lambda} F^{-1} \nabla_\theta J(\theta)$

Natural Policy Gradient

1. Natural policy gradient is a **second-order** optimization that is more accurate and works regardless of how the model is parameterized (model invariant).

$$\theta_{t+1} = \theta_t + \alpha F^{-1} \nabla_\theta J(\theta)$$

   1. where $F = E_{\pi_\theta(s,a)}[\nabla \log \pi_\theta(s,a) \nabla \log \pi_\theta(s,a)^T]$ is the Fisher information matrix, also as the second-order derivative of the KL-divergence
   2. F measures the curvature of the policy(distribution) relative to the model parameter θ

2. Natural policy gradient produces the same policy change **regardless of the model parameterization**.
3. Idea behind TRPO. A detailed read on natural gradient: `https://wiseodd.github.io/techblog/2018/03/14/natural-gradient/`

Different Schools of Reinforcement Learning

1. Value-based RL: solve RL through dynamic programming
   1. Classic RL and control theory
   2. Representative algorithms: Deep Q-learning and its variant
   3. Representative researchers: Richard Sutton (no more than 20 pages on PG out of the 500-page textbook), David Silver, from DeepMind
2. Policy-based RL: solve RL mainly through learning
   1. Machine learning and deep learning
   2. Representative algorithms: PG, and its variants TRPO, PPO, and others
   3. Representative researchers: Pieter Abbeel, Sergey Levine, John Schulman, from OpenAI, Berkelely
3. Some random essay I wrote on Zhihu:`https://www.zhihu.com/question/316626294/answer/627373838`

请问DeepMind和OpenAI身后的两大RL流派有什么具体的区别?

周博磊

机器学习、深度学习（Deep Learning）、人工智能 话题的优秀回答者

1,681 人赞同了该回答

Resource on policy gradient

1. A very nice summary of policy gradient algorithms: `https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html`
2. Educational blog by Karpathy: `http://karpathy.github.io/2016/05/31/rl/`

Code example

1. REINFORCE code on CartPole: https://github.com/cuhkrlcourse/RLexample/blob/master/policygradient/reinforce.py
2. Policy Gradient on Pong : https://github.com/cuhkrlcourse/RLexample/blob/master/policygradient/pg-pong-pytorch.py
3. Policy Gradient with Baseline on Pong: https://github.com/cuhkrlcourse/RLexample/blob/master/policygradient/pgb-pong-pytorch.py
4. Actor Critic on Pong: https://github.com/cuhkrlcourse/RLexample/blob/master/policygradient/ac-pong-pytorch.py

Next Week: State of the Arts on Policy Optimization

1. Policy Graident→TRPO→ACKTR→PPO
   1. **TRPO**: Trust region policy optimization. Schulman, L., Moritz, Jordan, Abbeel. 2015
   2. **ACKTR**: Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba. 2017
   3. **PPO**: Proximal policy optimization algorithms. Schulman, Wolski, Dhariwal, Radford, Klimov. 2017
2. Q-learning→DDPG→TD3→SAC
   1. **DDPG**: Deterministic Policy Gradient Algorithms, Silver et al. 2014
   2. **TD3**: Addressing Function Approximation Error in Actor-Critic Methods, Fujimoto et al. 2018
   3. **SAC**: Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, Haarnoja et al. 2018