Soft Actor Critic

Reading

- Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." In *International conference on machine learning*, pp. 1861-1870. PMLR, 2018.

Overview

Rensselaer

- Standard model-free RL methods need A LOT OF samples
  - Model-free RL algorithms are ones that do not try to recover the underlying MDP structure
    - Will discuss model-based approaches later
  - Reasons?
    - Poor exploration
    - Off-policy learning, which results in inefficient gradients
    - On-policy learning, which requires a lot of data to re-compute values for new policies
- The soft actor-critic (SAC) method tries to alleviate these challenges
  - Back to stochastic policies and a slightly modified objective that encourages exploration

Overall Approach

Rensselaer

- What is the objective in classic RL methods so far?
- Maximize the state values

$$\max_\pi v_\pi(s) = \mathbb{E}_\pi[G_0|S_0 = s]$$
$$= \sum_{t=1}^{T} \mathbb{E}_\pi[R_t|S_0 = s]$$

  – Ignoring discounting to keep notation simple
- Suppose we want to encourage policies that explore more
  – How can we modify the objective?
  – Also maximize policy entropy

$$\max_\pi J(\pi) = \sum_{t=1}^{T} \mathbb{E}_\pi\left[R_t + \alpha H\big(\pi(\cdot\,|S_t)\big)\middle|S_0 = s\right]$$

Benefits of maximizing entropy

- Recall the definition of entropy

$$H(X) = -\sum_{x} p(x) \log[p(x)] = -\mathbb{E}[\log[p(X)]]$$

- Similarly,

$$H\big(\pi(\cdot\,|S_t)\big) = -\sum_{a} \pi(a|S_t) \log[\pi(a|S_t)] = -\mathbb{E}_{A_t}[\log[\pi(A_t|S_t)]]$$

  – Encourages more exploration

- Discourages getting stuck in local minima
  – More exploration makes this unlikely
- Incentivizes learning multiple ways to maximize the reward
  – May make the policy more robust in parts of the state space that haven't been explored yet

Soft Policy Evaluation

Rensselaer

- What is a soft policy?
  - Any stochastic policy derived from a deterministic policy
  - E.g., an ϵ-greedy policy
- How did we evaluate policies in the finite-MDP case?
  - Iterative policy iteration
  - Apply the Bellman operator iteratively

$$v_k(\boldsymbol{s}) = R(\boldsymbol{s}) + \gamma \boldsymbol{P} v_{k-1}(\boldsymbol{s})$$

Soft Policy Evaluation, cont'd

- Notice that the new reward is

$$R_{\pi,t}(S_t, A_t) = R(S_t, A_t) + \alpha H\big(\pi(\cdot\,|S_t)\big)$$

  - we'll omit α from now on (paper does it for simplicity)
  - where R is the deterministic reward function and

$$H\big(\pi(\cdot\,|S_t)\big) = \mathbb{E}_\pi\big[-\log\big(\pi(A_t|S_t)\big)\big]$$

- The value functions are now defined as

$$v_\pi(s) = \mathbb{E}_\pi\big[H\big(\pi(\cdot\,|S_t)\big) + R(S_t, A_t) + \cdots + H\big(\pi(\cdot\,|S_T)\big) + R(S_T, A_T)\big|S_t = s\big]$$

$$q_\pi(s,a) = \mathbb{E}_\pi\big[R(S_t, A_t) + \cdots + \alpha H\big(\pi(\cdot\,|S_T)\big) + R(S_T, A_T)\big|S_t = s, A_t = a\big]$$

  - Note that $H\big(\pi(\cdot\,|S_t)\big)$ is known before the next action

- The Bellman equations are the same as before
  - Same derivation as usual, just use $R_{\pi,t}$ as the reward

$$v_\pi(s) = \mathbb{E}_\pi\big[R_{\pi,t}(S_t, A_t) + v_\pi(S_{t+1})\big|S_t = s\big]$$

$$q_\pi(s,a) = \mathbb{E}_\pi\big[R_{\pi,t}(S_t, A_t) + q_\pi(S_{t+1}, A_{t+1})\big|S_t = s, A_t = a\big]$$

Soft Policy Evaluation, cont'd

- The value functions are now defined as

$$v_\pi(s) = \mathbb{E}_\pi\left[H\big(\pi(\cdot\,|s)\big) + R(S_t, A_t) + \cdots H\big(\pi(\cdot\,|S_T)\big) + R(S_T, A_T)\middle|S_t = s\right]$$
$$q_\pi(s, a) = \mathbb{E}_\pi\left[R(S_t, A_t) + \cdots \alpha H\big(\pi(\cdot\,|S_T)\big) + R(S_T, A_T)\middle|S_t = s, A_t = a\right]$$

- The Bellman equations are the same as before
  - Same derivation as usual, just use $R_{\pi,t}$ as the reward

$$v_\pi(s) = \mathbb{E}_\pi\left[R_{\pi,t}(S_t, A_t) + v_\pi(S_{t+1})\middle|S_t = s\right]$$
$$q_\pi(s, a) = \mathbb{E}_\pi\left[R_{\pi,t}(S_t, A_t) + q_\pi(S_{t+1}, A_{t+1})\middle|S_t = s, A_t = a\right]$$

- Note that the state value function can be written as

$$\begin{aligned} v_\pi(s) &= H\big(\pi(\cdot\,|s)\big) + \mathbb{E}_\pi\left[R(S_t, A_t) + \cdots H\big(\pi(\cdot\,|S_T)\big) + R(S_T, A_T)\middle|S_t = s\right] \\ &= H\big(\pi(\cdot\,|s)\big) + \mathbb{E}_\pi[q(s, A_t)|S_t = s] \\ &= \mathbb{E}_\pi\left[q_\pi(s, A_t) - \log\big(\pi(A_t|s)\big)\,\middle|S_t = s\right] \end{aligned}$$

  - Again, recall $H\big(\pi(\cdot\,|S_t)\big) = \mathbb{E}_\pi\left[-\log\big(\pi(A_t|S_t)\big)\right]$

Soft Policy Improvement

- Recall the standard policy improvement idea

$$\pi'(s) = arg\max_a q_\pi(s,a)$$

  - The new policy is guaranteed to be better than the old one
  - If we find a better policy, iterate until convergence

- Recall the q value is now

$$v_\pi(s) = \mathbb{E}_\pi\left[q_\pi(s,A_t) - \log\big(\pi(A_t|s)\big)\,\middle|S_t = s\right]$$
$$= -D_{KL}\left(\pi(A_t|s)||\eta\exp\big(q_\pi(s,A_t)\big)\right)$$

  - where η is a normalizing constant such that

$$\eta\sum_a \exp\big(q_\pi(s,a)\big) = 1$$

  - Also recall

$$D_{KL}(p||q) = \mathbb{E}\left[\log\left[\frac{p(X)}{q(X)}\right]\right]$$
$$= \mathbb{E}\left[\log\big(p(X)\big) - \log\big(q(X)\big)\right]$$

Soft Policy Improvement, cont'd

- To apply the policy improvement theorem, choose the maximizing action, i.e., minimize the KL divergence

$$\pi'(\cdot\,|s) = arg\min_{\pi\in\Pi} D_{KL}(\pi(A_t|s)||\eta\exp(q_\pi(s, A_t)))$$

  – where Π is the set of all considered policies, e.g., neural nets

- Why is π' better than π?
  – Recall $v_\pi(s) = -D_{KL}(\pi(A_t|s)||\eta\exp\big(q_\pi(s, A_t)\big))$
  – In the paper, they prove this implies that $q_{\pi'}(s, a) \geq q_\pi(s, a)$ for all actions
  – Proof is similar to the standard policy improvement proof we saw earlier

Soft Policy Iteration

- Given the policy evaluation and policy improvement results, the authors aim to apply standard policy iteration
  - Evaluate policy
  - Improve policy by minimizing $D_{KL}(\pi(\cdot\,|s)||\eta\exp\big(q(s, A_t)\big))$
- Authors prove that by applying policy iteration
  1. The process converges because it's bounded from above by the optimal value
  2. The final policy is the optimal as it satisfies the Bellman optimality equation

Implementation

- Of course, once neural networks are used, using policy iteration is extremely inefficient
  - Finding the optimal neural net at each step is not necessary
    - Won’t find the optimal neural net anyway
  - We’ll use approximators for the q function also, which means the gradients may be noisy or wrong
- So the authors use the standard neural net solution
  - Gradient descent with policy gradients!

Implementation, cont'd

- Authors choose to have two critics
  - One each approximating the v and q function, respectively
  - Stabilizes training
- Train each critic using least squares
  - Value critic parameterized by $\boldsymbol{\psi}$

$$\min_{\boldsymbol{\psi}} \left(V_{\boldsymbol{\psi}}(S_t) - Q_{\boldsymbol{\theta}}(S_t, A_t) + log\pi_{\boldsymbol{\phi}}(A_t|S_t) \right)^2$$

    - Essentially learns the policy entropy over all actions
    - Targets bootstrapped using action-value critic and policy
  - Action value critic parameterized by $\boldsymbol{\theta}$

$$\min_{\boldsymbol{\theta}} \left(Q_{\boldsymbol{\theta}}(S_t, A_t) - R_t - \gamma V_{\boldsymbol{\psi}}(S_{t+1}) \right)^2$$

    - Targets bootstrapped using value critic

Implementation, cont'd

- The policy is trained by minimizing the KL divergence
  - i.e., following the gradient of

$$v_{\pi_{\boldsymbol{\phi}}}(s) = \mathbb{E}_\pi \left[q_{\pi_{\boldsymbol{\phi}}}(s, A_t) - \log\left(\pi_{\boldsymbol{\phi}}(A_t|s)\right) \middle| S_t = s \right]$$

- To learn a complex policy, $\pi_{\boldsymbol{\phi}}$ has to be (based on) a neural net
  - How do we train a neural network with random outputs?
  - **Reparameterization trick**: train a neural net $f_{\boldsymbol{\phi}}$ to output the parameters of a known distribution (e.g., Gaussian)
    - Given an input state S_t
      - first sample $\epsilon_t \sim \mathcal{N}(0,1)$ and set $f_{\boldsymbol{\phi}}(S_t) = \left(\mu_{\boldsymbol{\phi}}, \sigma_{\boldsymbol{\phi}}\right)$
      - Now set $A_t = \mu_{\boldsymbol{\phi}} + \epsilon_t \sigma_{\boldsymbol{\phi}}$ and note $A_t \sim \mathcal{N}\left(\mu_{\boldsymbol{\phi}}, \sigma_{\boldsymbol{\phi}}\right)$
      - Finally, $\pi_{\boldsymbol{\phi}}(A_t|S_t) = p_{\mu_{\boldsymbol{\phi}}, \sigma_{\boldsymbol{\phi}}}(A_t)$, where $p_{\mu_{\boldsymbol{\phi}}, \sigma_{\boldsymbol{\phi}}}$ is the pdf of $\mathcal{N}\left(\mu_{\boldsymbol{\phi}}, \sigma_{\boldsymbol{\phi}}\right)$
    - Can backpropagate through sampling: deep learning black magic

Final Algorithm

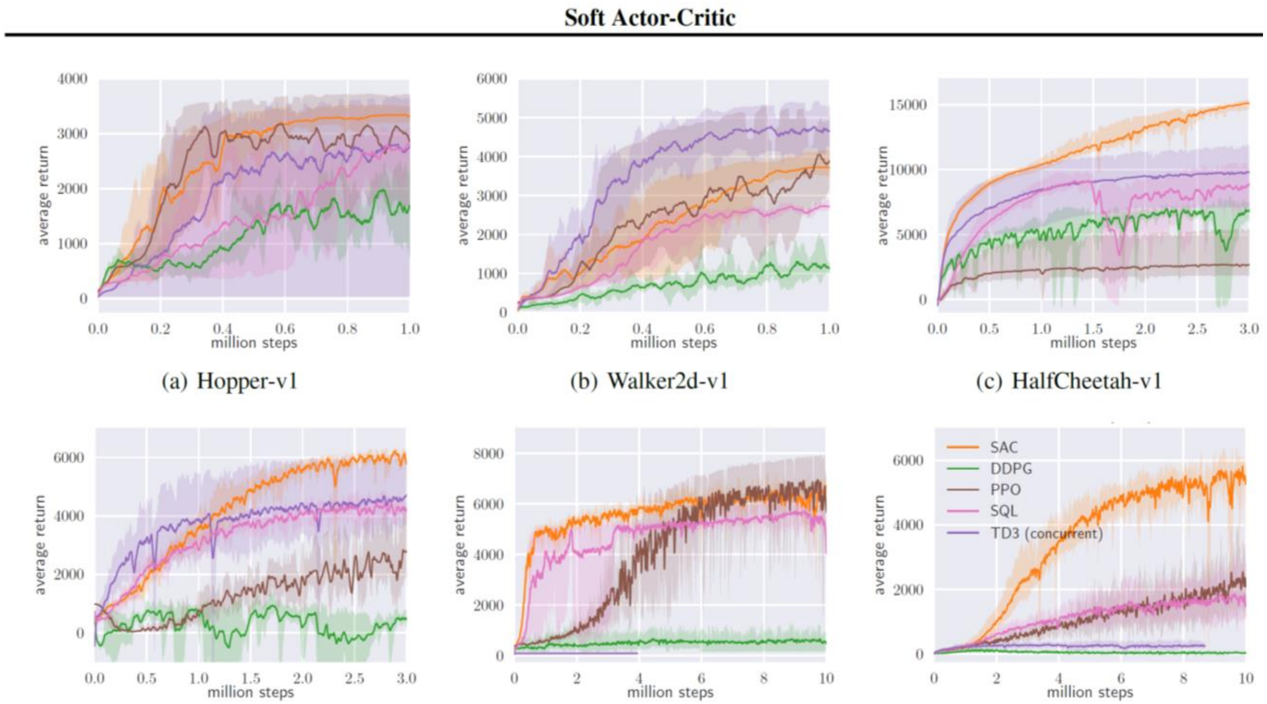
- Used two sets of critics, as in TD3
- Used target networks, as in TD3 and DDPG
- Main difference is the policy gradient
  - i.e., the definition of the value function

Algorithm 1 Soft Actor-Critic

Initialize parameter vectors ψ, $\bar{\psi}$, θ, ϕ.
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$
  end for
end for

Evaluation

- Performance seems very promising, even compared to TD3
  - Much better performance on humanoid tasks where TD3 crashes
    - Humanoid (rllab) not a standard benchmark, though

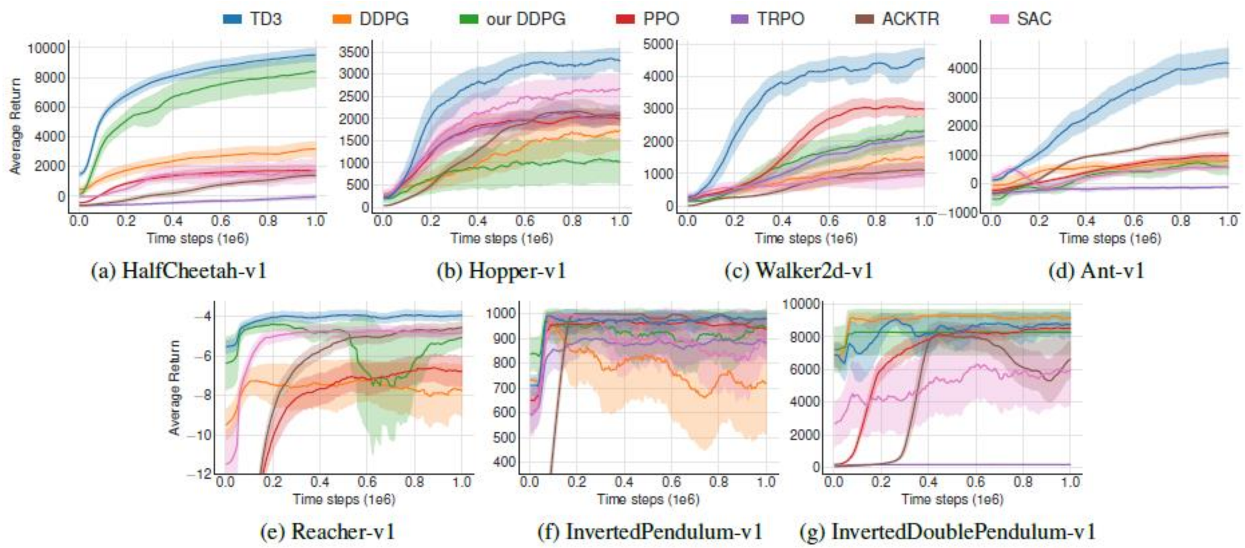


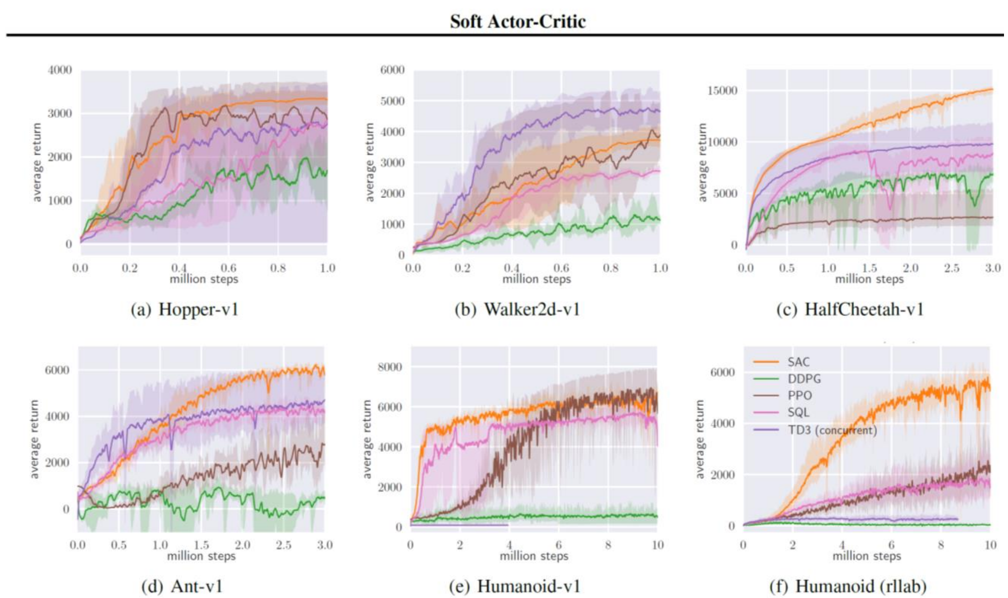
(a) Hopper-v1 (b) Walker2d-v1 (c) HalfCheetah-v1

(d) Ant-v1 (e) Humanoid-v1 (f) Humanoid (rllab)

Compare and Contrast

- Trends?
  - TD3 is the same in both
  - SAC wins mostly in their own evaluation
  - Why?
    - Likely hyperparemeter tuning





Summary

Rensselaer

- Actor-critic methods are state-of-the-art in model-free RL
- Can now be used in high-dimensional settings with images
- Training is very volatile but can be made to work if you know what you're doing
- Algorithms depend heavily on hyperparameter selection
- I prefer TD3 as it seems to be the most stable as of now
- Main limitation of model-free RL is that it requires massive amounts of data
  - Mostly works for simulators/games
- Offline RL is a more realistic setting, but much harder
  - More next

Implementation, cont'd

- The policy is trained by minimizing the KL divergence
  - i.e., following the gradient of

$$v_{\pi_{\boldsymbol{\phi}}}(s) = \mathbb{E}_{\pi}\left[q_{\pi_{\boldsymbol{\phi}}}(s, A_t) - \log\left(\pi_{\boldsymbol{\phi}}(A_t|s)\right)\middle|S_t = s\right]$$

- The gradient with respect to $\boldsymbol{\phi}$ is (in the finite action case)

$$\nabla_{\boldsymbol{\phi}} v_{\pi_{\boldsymbol{\phi}}}(s) = \sum_a \nabla_{\boldsymbol{\phi}}\left[\pi_{\boldsymbol{\phi}}(a|s)\left(q_{\pi_{\boldsymbol{\phi}}}(s, a) - \log\left(\pi_{\boldsymbol{\phi}}(a|s)\right)\right)\right]$$

- As usual, authors approximate it using a batch of points $(S_1, A_1), \ldots, (S_M, A_M)$
  - Essentially, just calculate the average gradient

$$\sum_{t=1}^{M} q_{\pi_{\boldsymbol{\phi}}}(S_t, A_t) - \log\left(\pi_{\boldsymbol{\phi}}(A_t|S_t)\right)$$

Implementation, cont'd

- The gradient with respect to $\boldsymbol{\phi}$ is (in the finite action case)

$$\nabla_{\boldsymbol{\phi}} v_{\pi_{\boldsymbol{\phi}}}(s) = \sum_{a} \nabla_{\boldsymbol{\phi}} \left[\pi_{\boldsymbol{\phi}}(a|s) \left(q_{\pi_{\boldsymbol{\phi}}}(s,a) - \log\left(\pi_{\boldsymbol{\phi}}(a|s)\right)\right)\right]$$

- As usual, authors approximate it using a batch of points
  - E.g., note for a Gaussian:

$$\log\left(\pi_{\boldsymbol{\phi}}(a|s)\right) = \log\left(\frac{1}{\sqrt{2\pi f_{\boldsymbol{\theta},2}(s)}} \exp\left(\frac{\left(a - f_{\boldsymbol{\theta},1}(s)\right)^2}{2 f_{\boldsymbol{\theta},2}(s)^2}\right)\right)$$

$$= -\frac{1}{2}\log(2\pi) - \frac{1}{2}\log\left(f_{\boldsymbol{\theta},2}(s)\right) - \frac{\left(a - f_{\boldsymbol{\theta},1}(s)\right)^2}{2 f_{\boldsymbol{\theta},2}(s)^2} := c - \pi_{1,\boldsymbol{\phi}}(s) - \pi_{2,\boldsymbol{\phi}}(s,a)$$

  - Thus the gradient becomes

$$-\nabla_{\boldsymbol{\phi}} \pi_{1,\boldsymbol{\phi}}(S_t) + \left(\left(-\nabla_a \pi_{2,\boldsymbol{\phi}}(S_t,a) + \nabla_a q_{\pi_{\boldsymbol{\phi}}}(S_t,a)\,\Big|_{a=A_t}\right)\right) \nabla_{\boldsymbol{\phi}} f_{\boldsymbol{\phi}}(S_t)$$

  - Similar to standard policy gradient theorem