Study Note: Soft Actor Critic

Yanqing Wu

Viwistar Robotics

Update: December 13, 2021

Since some parts are not addressed directly from the literature, the explanations in this note are the best of my understanding.

1 Introduction

Model-free deep reinforcement learning (RL) algorithms are proven to work on decision making and control tasks. However, these methods are hindered by two major challenges: very high sample complexity (requires at least millions of steps of data collection) and brittle convergence properties (extremely sensitive to hyperparameters). One cause for the high sample complexity (or poor sample efficiency) is on-policy learning. On-policy learning methods, such as TRPO, PPO, or A3C, require collecting new samples for each gradient step and which is extremely expensive. To improve sample efficiency, we need to reuse past experience (or 'experience replay' in literature), which requires off-policy algorithms Mnih et al. (2013). However, off-policy variants based on soft Q-learning require complex approximate inference procedures in continuous action spaces.

Soft Actor Critic Haarnoja et al. (2018), or SAC, was introduced to provide both sample efficiency and stability in continuous action spaces, and extendability to complex and high-dimensional tasks. SAC is an off-policy actor-critic deep RL algorithm based on the maximum entropy framework. In this framework, the actor aims to maximize expected reward as well as entropy. That is, to succeed at the task while acting as randomly as possible. Prior deep RL methods based on this framework have been formulated as Q-learning methods. SAC combines off-policy updates with a stable stochastic actor-critic formulation.

2 SAC

2.1 Implementation

Three key components in SAC:

- An actor-critic architecture with separate policy and value function networks;
- An off-policy formulation that enables reuse of previously collected data for efficiency;
- Entropy maximization to enable stability and exploration.

The policy is trained with the **objective** to maximize the expected return and the entropy at the same time:

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_{\pi_\theta}} [r(s_t, a_t) + \alpha\mathbb{H}(\pi_\theta(\cdot|s_t))]$$

where $\mathbb{H}(\cdot)$ is the entropy measure and α (temperature parameter) controls how important the entropy term is. The entropy maximization leads to policies that can (1) explore more and (2) capture multiple modes of near-optimal strategies (i.e., if there exist multiple options that seem to be equally good, the policy should assign each with an equal probability to be chosen).

Precisely, SAC aims to learn three functions:

- π_θ, the policy with parameter θ
- Q_w, soft Q-value function parameterized by w
- V_ψ, soft state-value function parameterized by ψ

Soft Q-value (or soft action value) and **soft state value** are defined as:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma\mathbb{E}_{s_{t+1}\sim\rho_\pi(s)}[V(s_{t+1})] \qquad \text{; according to Bellman equation.}$$
$$\text{where } V(s_t) = \mathbb{E}_{a_t\sim\pi}[Q(s_t, a_t) - \underbrace{\alpha \log \pi(a_t|s_t)}_{\text{entropy}}] \qquad \text{; soft state value function.}$$

$$\text{Thus, } Q(s_t, a_t) = r(s_t, a_t) + \gamma\mathbb{E}_{(s_{t+1},a_{t+1})\sim\rho_\pi}[Q(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1}|s_{t+1})]$$

$\rho_\pi(s_t)$ and $\rho_\pi(s_t, a_t)$ denote the state and state-action marginals of the trajectory (state) distribution included by a policy $\pi(a_t|s_t)$.

The **soft state value function** is trained to minimize the mean square error:

$$J_V(\psi) = \mathbb{E}_{s_t\sim\mathcal{D}}[\frac{1}{2}\big(\underbrace{V_\psi(s_t)}_{\text{value}} - \mathbb{E}[\underbrace{Q_w(s_t, a_t)}_{\text{target}} - \underbrace{\log \pi_\theta(a_t|s_t)}_{\text{entropy}}]\big)^2]$$
$$\text{with gradient: } \nabla_\psi J_V(\psi) = \nabla_\psi V_\psi(s_t)\big(V_\psi(s_t) - Q_w(s_t, a_t) + \log \pi_\theta(a_t|s_t)\big)$$

where $\mathcal{D}$ is the replay buffer.

The **soft Q function** is trained to minimize the soft Bellman residual:

$$J_Q(w) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}[\frac{1}{2}\big(Q_w(s_t, a_t) - \underbrace{(r(s_t, a_t) + \gamma\mathbb{E}_{s_{t+1}\sim\rho_\pi(s)}[V_{\bar{\psi}}(s_{t+1})])}_{\text{target}}\big)^2]$$
$$\text{with gradient: } \nabla_w J_Q(w) = \nabla_w Q_w(s_t, a_t)\big(Q_w(s_t, a_t) - r(s_t, a_t) - \gamma V_{\bar{\psi}}(s_{t+1})\big)$$

where $\bar{\psi}$ is the target value function which is the exponential moving average (or only gets updated periodically in a ‘hard’ way), just like how the parameter of the target Q network is treated in DQN to stabilize the training.

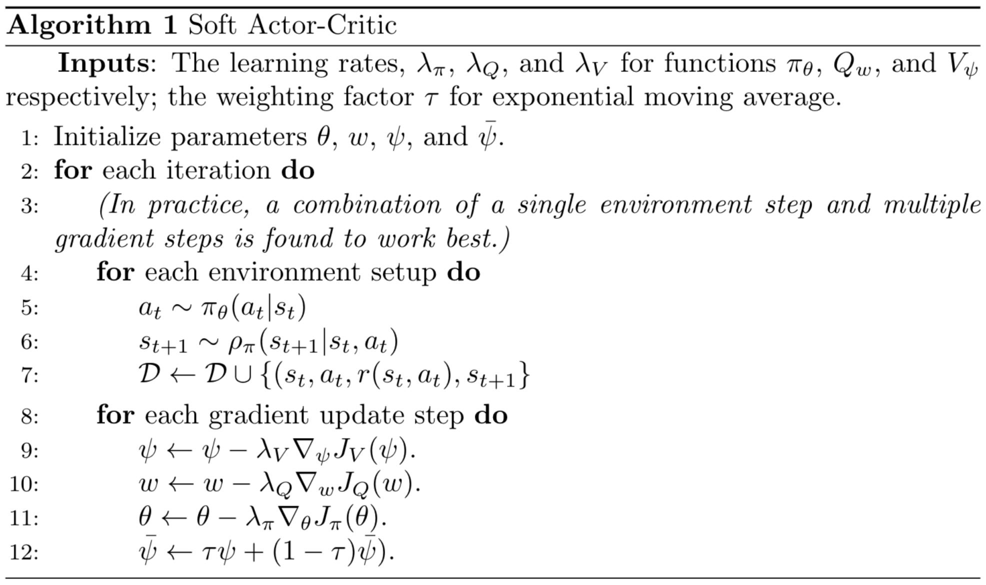
SAC updates the policy to minimize the KL-divergence:

$$\begin{aligned}
\pi_{\text{new}} &= \arg\min_{\pi'\in\Pi} D_{\text{KL}}\Big(\pi'(\cdot|s_t)\|\frac{\exp(Q^{\pi_{\text{old}}}(s_t,\cdot))}{Z^{\pi_{\text{old}}}(s_t)}\Big) \\
&= \arg\min_{\pi'\in\Pi} D_{\text{KL}}\big(\pi'(\cdot|s_t)\|\exp(Q^{\pi_{\text{old}}}(s_t,\cdot)-\log Z^{\pi_{\text{old}}}(s_t))\big) \\
\text{objective for update: } J_\pi(\theta) &= \nabla_\theta D_{\text{KL}}\big(\pi_\theta(\cdot|s_t)\|\exp(Q_w(s_t,\cdot)-\log Z_w(s_t))\big) \\
&= \mathbb{E}_{a_t\sim\pi}\Big[-\log\big(\frac{\exp(Q_w(s_t,a_t)-\log Z_w(s_t))}{\pi_\theta(a_t|s_t)}\big)\Big] \\
&= \mathbb{E}_{a_t\sim\pi}[\log\pi_\theta(a_t|s_t)-Q_w(s_t,a_t)+\log Z_w(s_t)]
\end{aligned}$$

where Π is the set of potential policies that we can model our policy as to keep them tractable; e.g., Π can be the family of Gaussian mixture distributions, expensive to model but highly expressive and still tractable. $Z^{\pi_{\text{old}}}(s_t)$ is the partition function to normalize the distribution. It is usually intractable but does not contribute to the gradient. How to minimize $J_\pi(\theta)$ depends our choice of Π.

This update guarantees that $Q^{\pi_{\text{new}}}(s_t,a_t)\geq Q^{\pi_{\text{old}}}(s_t,a_t)$, please check the proof on this lemma in the Appendix B.2 in Haarnoja et al. (2018).

Once we have defined the objective functions and gradients for soft action-state value, soft state value and the policy network, the soft actor-critic algorithm is as follows:



```
Algorithm 1 Soft Actor-Critic
Inputs: The learning rates, $\lambda_\pi$, $\lambda_Q$, and $\lambda_V$ for functions $\pi_\theta$, $Q_w$, and $V_\psi$
respectively; the weighting factor $\tau$ for exponential moving average.
 1: Initialize parameters $\theta$, $w$, $\psi$, and $\bar{\psi}$.
 2: for each iteration do
 3:     (In practice, a combination of a single environment step and multiple
        gradient steps is found to work best.)
 4:     for each environment setup do
 5:         $a_t \sim \pi_\theta(a_t|s_t)$
 6:         $s_{t+1} \sim \rho_\pi(s_{t+1}|s_t,a_t)$
 7:         $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t,a_t,r(s_t,a_t),s_{t+1}\}$
 8:     for each gradient update step do
 9:         $\psi \leftarrow \psi - \lambda_V \nabla_\psi J_V(\psi)$.
10:         $w \leftarrow w - \lambda_Q \nabla_w J_Q(w)$.
11:         $\theta \leftarrow \theta - \lambda_\pi \nabla_\theta J_\pi(\theta)$.
12:         $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$.
```

Figure 1: Screenshot of SAC algorithm

Source Code `https://github.com/haarnoja/sac`

2.2 SAC Improvement

2.2.1 SAC with automatically adjusted temperature

SAC is brittle with respect to the temperature parameter. Unfortunately it is difficult to adjust temperature, because the entropy can vary unpredictably both across tasks and during training as the policy becomes better. An improvement on SAC formulates a constrained optimization problem: while maximizing the expected return, the policy should satisfy a minimum entropy constraint Haarnoja et al. (2019).

3 Further reading

- L5 DDPG and SAC (Foundations of Deep RL Series) (by Pieter Abbeel)
  - "you can think of SAC as maximum entropy version of DDPG" by Pieter Abbeel
- Lecture 19 Off-Policy, Model-Free RL: DQN, SoftQ, DDPG, SAC – CS287-FA19 Advanced Robotics (by Pieter Abbeel)
- Lecture 20 Model-Based Reinforcement Learning – CS287-FA19 Advanced Robotics at UC Berkeley (by Pieter Abbeel)
- DDPG: `https://arxiv.org/pdf/1509.02971.pdf`
  - `https://spinningup.openai.com/en/latest/algorithms/ddpg.html`
  - DDPG explained blog 1, DDPG explained blog 2

Appendices

Appendix A Taxonomy of reinforcement learning algorithms

A.1 Model-free vs. Model-based

You may have seen 'model' referring to neural networks like in supervised learning. In RL, the term 'model' in 'model-free' or 'model-based' does NOT refer to neural networks or other statistical learning models. To avoid ambiguity, neural networks are referred as 'function approximators' in RL, which are often employed to learn and generalize value functions (such as Q values that predicts total return given a state-action pair).

Model-based RL has an agent try to understand the environment and create a 'model' represent it. **The transition function (probability distribution) from states T and the reward function R are called the 'model' of the environment** (or Markov decision process, MDP). From this model, the agent has a reference and can *plan* accordingly. Model-free agent does not learn a model but instead learn a policy directly.

A RL agent is not 'model-based' even there is a model of the environment implemented. A RL agent have to explicitly reference the 'model' to be 'model-based':

- 'model-free' algorithms: algorithms that purely sample from experience (e.g. Monte Carlo control, SARSA, Q-learning, Actor-Critic)
  - They rely on real samples from the environment and never use generated predictions of next state and next reward to alter behaviour. They might sample from experience memory (e.g. replay buffer), which is close to being a model.
- archetypical 'model-based' algorithms: Dynamic Programming (e.g. Policy Iteration, Value Iteration)
  - They use the model's predictions or distributions of enxt state and reward to calculate optimal actions. In DP specifically, the model must provide state transition probabilities, and expected reward from any state, action pair.
- other 'model-based' algorithms: basic TD learning that only uses state values
  - To pick the optimal action, it needs to ask a model that predicts what will happen on each action and implement a policy like $\pi(s) = argmax_a \sum_{s',r} p(s', r|s, a)(r + v(s'))$ where probability function $p(s', r|s, a)$ is essentially the model.

In short, model-based RL algorithms use models and planning to solve RL problems; model-free methods are explicitly trial-and-error learners (almost the opposite of planning).

A simple check to see if an RL algorithm is model-based or model-free is: *if, after learning, the agent can make predictions about what the next state and reward will be before it takes each action, it is a model-based RL algorithm. If it cannot, then it is a model-free algorithm.*

A.2 On-policy vs. Off-policy

By Prof. Yao, the key difference between the two is whether we use generated action to interact with the environment.

- On-policy learning: the same policy that is evaluated and improved is *also* used to select actions.
- Off-policy learning: the policy that is evaluated and improved (called estimation policy) is different from the policy that is used to select actions (called behaviour policy).

The on-policy methods, like SARSA, expects that the actions in every state are chosen based on the current policy of the agent, that usually tends to exploit rewards. By doing so, the policy gets better when we update our policy based on the last rewards. But, if we update our policy based on stored transitions, like in experience replay, we are actually evaluating actions from a policy that is no longer the current one.

An advantage of this separation is that the estimation policy may be deterministic (e.g. greedy), while the behaviour policy can continue to sample all possible actions. Hence, off-policy gives us better exploration and helps us use data samples more efficiently.

Within off-policy methods, there are two types: Q learning and Q-based policy gradient (or Q-based actor critic) Abbeel (2021).

A.3 Policy-based vs. Value-based

- policy-based: we explicitly build a representation of a policy (mapping $\pi : s \to a$) and keep it in memory during learning
- value-based: we don't store any explicit policy, only a value function. The policy is here implicit and can be derived directly from the value function (pick the action with the best value)
- actor-critic: a mix of policy-based (actor) and value-based (critic).

A.3.1 Q-learning vs. Policy Gradient

A more specific and symbolic example of policy-based vs value-based methods is: Q-learning (value-balsed) and Policy Gradient (policy-based).

In a post by Slater (2021): Both methods are theoretically driven by the Markov Decision Process construct, and as a result use similar notation and concepts. In addition, in simple solvable environments you should expect both methods to result in the same - or at least equivalent - optimal policies.

However, they are actually different internally. The most fundamental differences between the approaches is in how they approach action selection, both whilst learning, and as the output (the learned policy). In Q-learning, the goal is to learn a single deterministic action from a discrete set of actions by finding the maximum value. With policy gradients, and other direct policy searches, the goal is to learn a map from state to action, which can be stochastic, and works in continuous action spaces.

As a result, policy gradient methods can solve problems that value-based methods cannot:

- Large and continuous action space. However, with value-based methods, this can still be approximated with discretisation - and this is not a bad choice, since the mapping function in policy gradient has to be some kind of approximator in practice.
- Stochastic policies. A value-based method cannot solve an environment where the optimal policy is stochastic requiring specific probabilities, such as Scissor/Paper/Stone. That is because there are no trainable parameters in Q-learning that control probabilities of action, the problem formulation in TD learning assumes that a deterministic agent can be optimal.

However, value-based methods like Q-learning have some advantages too:

- Simplicity. You can implement Q functions as simple discrete tables, and this gives some guarantees of convergence. There are no tabular versions of policy gradient, because you need a mapping function $p(a|s, \theta)$ which also must have a smooth gradient with respect to θ.
- Speed. TD learning methods that bootstrap are often much faster to learn a policy than methods which must purely sample from the environment in order to evaluate progress.

There are other reasons why you might care to use one or other approach:

- You may want to know the predicted return whilst the process is running, to help other planning processes associated with the agent.
- The state representation of the problem lends itself more easily to either a value function or a policy function. A value function may turn out to have very simple relationship to the

state and the policy function very complex and hard to learn, or vice-versa.

Some state-of-the-art RL solvers actually use both approaches together, such as Actor-Critic. This combines strengths of value and policy gradient methods.

Appendix B Soft policy vs. Stochastic policy

- deterministic policy, $\pi(s) = a$
- stochastic policy, $\pi(a|s) = \mathbb{P}_\pi[A = a|S = s]$
- soft policy, $\pi(a|s) = \mathbb{P}_\pi[A = a|S = s] > 0$

The term 'soft policy' and 'stochastic policy' are not interchangeable. Soft policy are always stochastic, but not all stochastic policies are soft policies. For example, given $A = \{a, b, c\}$, then a policy $\pi(a) = 0.5, \pi(b) = 0.5, \pi(c) = 0$ is a stochastic policy, but it is not a soft policy.

A 'soft' policy is one that has some, usually small but finite, probability of selecting any possible action. Soft policies are important for practical purposes of exploring alternative actions, and they can give theoretical guarantees of convergence for RL algorithms. A common approach to create a soft policy is by ϵ-greedy action selection over $Q(s, a)$, where the action with highest value estimate is used with $p = 1 - \epsilon$, or with $p = \epsilon$, a random action is chosen with equal chance of any action. You may also see the term ϵ-soft policy, which is a policy where every action has at least $p = \frac{\epsilon}{|A|}$ chance of being selected. The ϵ-greedy policy is also an ϵ-soft policy.

References

Abbeel, P. (2021). *University of California, Berkeley CS287-FA19 Advanced Robotics*. https://www.youtube.com/watch?v=QASqaj_HUZw [Accessed: 2021-12-07].

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. (2019). Soft actor-critic algorithms and applications.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

Slater, N. (2021). *What is the relation between Q-learning and policy gradients methods?* https://ai.stackexchange.com/a/6199 [Accessed: 2021-12-13].