Soft Actor-Critic (SAC) Algorithm

The description below is mainly made of notes from the original paper here and the following documents:

- Berkeley implementation
- Medium
- OpenAI - Spinning Up

Description of SAC

- SAC is an off-policy actor-critic algorithm based on the maximum entropy RL framework

The maximum entropy objective has a number of conceptual and practical advantages. First, the policy is incentivized to explore more widely, while giving up on clearly unpromising avenues. Second, the policy can capture multiple modes of near-optimal behavior. In problem settings where multiple actions seem equally attractive, the policy will commit equal probability mass to those actions. In practice, we observe improved exploration with this objective

- The actor aims to simultaneously maximize expected return and entropy We extend SAC to incorporate a number of modifications that accelerate training and improve stability with respect to the hyperparameters, including a constrained formulation that automatically tunes the temperature hyperparameter.

Our soft actor-critic algorithm incorporates **three key ingredients**:

1. an actor-critic architecture with separate policy and value function networks
2. an off-policy formulation that enables reuse of previously collected data for efficiency
3. entropy maximization to encourage stability and exploration.

SAC is a loop of policy evaluation through Q-value updates of the critic (based on the modified objective including the entropy term) and policy improvement updates (actor). This loop is called a policy iteration.

Policy Evaluation - soft Q-Update

We apply the Bellman operator, augmented by an entropy term

$$T^{\pi}Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} -> p}[V(s_{t+1}]$$

where

$$V(s_t) = \mathbb{E}_{a_t -> \pi}[Q(s_t, a_t) - \alpha . log \pi(a_t, s_t)]$$

We define the following objective policy;

$$J_Q(\theta) = \mathbb{E}_{(s_t, a_t) \sim \mathcal{D}} \Big[\frac{1}{2} \big(Q_\theta(s_t, a_t) - q_t \big)^2 \Big],$$

$$\text{where } q_t = \mathbb{E}_{a \sim \pi_\omega(\cdot|s_{t+1})} \big[\big(r(s_t, a_t) + Q_\theta(s_{t+1}, a) - \alpha \log \pi_\omega(a'|s_{t+1}) \big) \big]$$

Alpha term represents the "entropy temperature," i.e. how much we weight the "randomness" of our policy versus the environment reward.

Alpha can be tuned automatically. Alpha varies according to the magnitude of the rewards (different by task but also durng training. Instead of tuning the temperature manually, we treat it as a constraint by setting a target temperature (another hyperparameter though!). "Our aim is to find a stochastic policy with maximal expected return that satisfies a minimum expected entropy constraint" See page 7 of the paper.

Policy Improvement

We update the policy distribution towards the softmax distribution for the current Q function. We want to minimize the distance (“divergence”) between the two distributions. This is accomplished by minimizing the Kullback-Leibler (KL) divergence between the two distributions:

$$\pi_{new} = \arg\min_{\omega} J(\omega),$$

$$\text{where the objective } J(\omega) = \mathbb{E} \Big[\mathrm{KL} \Big(\pi(\cdot|s_t) \Big\| \frac{\exp Q(s, \cdot)}{\sum_a \exp Q(s, \cdot)} \Big) \Big]$$

Haarnoja et al. uses the “re-parameterization trick” on the policy output to get a low variance estimator; in particular, we represent the actions as the hyperbolic tangent (tanh) applied to z-values sampled from the mean and log standard deviation outputted by the policy neural

network.

Step by step:

- We do a forward pass on our netwro to convert a state into the mean, log std of a state
- We sample an action from a normal distribution parameterized by mean,std
- We squeeze the action value betwen -1 and 1 with Tanh
- We calculate log_pi (see formula below)

$$\log \pi(a|s) := \log \mu(\mathbf{u}|s) - \sum_{i=1}^{D} \log(1 - \tanh^2(u_i))$$

Explanations from the paper: we apply an invertible squashing function (tanh) to the Gaussian samples, and employ the change of variables formula to compute the likelihoods of the bounded actions. In the other words, let u ∈ R^D be a random variable and μ(uls) the corresponding density with infinite support. Then a = tanh(u), where tanh is applied elementwise, is a random variable with support in (–1, 1).