

Chapter 7. Soft Actor-Critic (SAC)

SAMSUNG AI

Reinforcement Learning

Jun 20, 2022

Kihun Kim, Mingyu Park



CORE
Control + Optimization Research Lab

SAC - Review



SAC - Review

How to incentivize exploration?

idea : augment reward as follows:

$$\sum_{t=0}^{T-1} \left(r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)) \right)$$

where $\mathcal{H}(\pi(\cdot|s_t))$: **entropy** of action distribution $\pi(\cdot|s_t)$

How to solve new MDP with this novel reward criterion?

→ soft Q-learning, soft Bellman equation, etc.

SAC - Implementation

SAC -Implementation

Gaussian actor network $a \sim \mathcal{N}(\mu_\phi(s), \sigma_\phi(s))$

two critic networks $Q_1(s, a; \theta_1), Q_2(s, a; \theta_2)$

target & loss construction

critic target : Remind: Entropy $H(X) = -\sum_{x \in X} p(x) \log p(x) = E[-\log p(x)]$

$$y_j = r_j + \gamma \mathbb{E}_{a \sim \pi(\cdot | s_j)} \left(Q(s'_j, a) - \alpha \log \pi(a | s_j) \right)$$

$$\rightarrow y_j = r_j + \gamma \mathbb{E}_{a \sim \pi_{\phi^-}(\cdot | s_j)} \left(\min_{i=1,2} Q_i(s'_j, a; \theta_i^-) - \alpha \log \pi_{\phi^-}(a | s_j) \right)$$

SAC -Implementation

actor loss :

$$\alpha \log \pi_{\phi}(f_{\phi}(\epsilon_j, s_j) | s_j) - \min_{i=1,2} Q_i(s_j, f_{\phi}(\epsilon_j, s_j))$$

Remark. π_{ϕ} : probability density, f_{ϕ} : actor network

SAC - Implementation

actor definition - 1st step

```
def forward(self, state, eval=False, with_log_prob=False):
```

```
    x = F.relu(self.fc1(state))
```

```
    x = F.relu(self.fc2(x))
```

```
    mu = self.fc3(x)
```

```
    log_sigma = self.fc4(x)
```

```
    # clip value of log_sigma, as was done in Haarnoja's implementation of SAC:
```

```
    # https://github.com/haarnoja/sac.git
```

```
    log_sigma = torch.clamp(log_sigma, -20.0, 2.0)
```

what we are doing here : compute **distribution params** $\mu_\phi(s)$ and $\sigma_\phi(s)$

```
    sigma = torch.exp(log_sigma)
```

```
    distribution = Independent(Normal(mu, sigma), 1)
```



SAC - Implementation

actor definition - 2nd step

```
if not eval:
```

```
    # use rsample() instead of sample(), as sample() does not allow back-propagation through params
```

```
    u = distribution.rsample()
```

```
    if with_log_prob:
```

```
        log_prob = distribution.log_prob(u)
```

```
        log_prob -= 2.0 * torch.sum((np.log(2.0) + 0.5 * np.log(self.ctrl_range) - u - F.softplus(-2.0 * u)), dim=1)
```

Reparameterization trick to ease computation of $\nabla \log \pi(a|s)$

```
    else:
```

```
        log_prob = None
```

```
else:
```

```
    u = mu
```

```
    log_prob = None
```

```
# apply tanh so that the resulting action lies in (-1, 1)^D
```

```
a = self.ctrl_range * torch.tanh(u)
```

$$u \sim \mathcal{N}(\mu_\phi(s), \sigma_\phi(s)) \longrightarrow a = \tanh(u) \sim \boxed{?}$$

```
return a, log_prob
```

$$\text{softplus}(x) = \log(1 + e^x)$$

SAC - Implementation

```
def __init__(self, dimS, dimA, hidden1, hidden2):  
    super(DoubleCritic, self).__init__()  
    self.fc1 = nn.Linear(dimS + dimA, hidden1)  
    self.fc2 = nn.Linear(hidden1, hidden2)  
    self.fc3 = nn.Linear(hidden2, 1)  
  
    self.fc4 = nn.Linear(dimS + dimA, hidden1)  
    self.fc5 = nn.Linear(hidden1, hidden2)  
    self.fc6 = nn.Linear(hidden2, 1)
```

```
def forward(self, state, action):
```

```
    x = torch.cat([state, action], dim=1)  
    x1 = F.relu(self.fc1(x))  
    x1 = F.relu(self.fc2(x1))  
    x1 = self.fc3(x1)  
  
    x2 = F.relu(self.fc4(x))  
    x2 = F.relu(self.fc5(x2))  
    x2 = self.fc6(x2)
```

```
    return x1, x2
```

```
def Q1(self, state, action):  
    x = torch.cat([state, action], dim=1)  
    x = F.relu(self.fc1(x))  
    x = F.relu(self.fc2(x))  
    x = self.fc3(x)  
  
    return x
```

this is how we define twin critics!



SAC - Implementation

```
1  with torch.no_grad():
2      # Get action with log(pi(a|s)) (also gradient)
3      next_actions, log_probs = agent.pi(next_obs_batch, with_log_prob=True)
4
5      # To calculate TQ, we need Q(s',pi(s'))
6      target_q1, target_q2 = agent.target_Q(next_obs_batch, next_actions)
7
8      # To mitigate overestimation! - Idea from TD3
9      target_q = torch.min(target_q1, target_q2)
10
11     #  $TQ^{\pi} = r + \gamma [ Q(s', \pi(s')) - \alpha H(\pi(s')) ]$ 
12     # Recall :  $H = \sum [ -P(x) * \log(P(x)) ] = E [ -\log(P(x)) ]$ 
13     # Recall :  $H \approx -\log(P(x))$ 
14     TQ = rew_batch + agent.gamma * masks * (target_q - agent.alpha * log_probs)
15
16     # Calculate MSELoss
17     Q1, Q2 = agent.Q(obs_batch, act_batch)
18     Q_loss1 = torch.mean((Q1 - TQ)**2)
19     Q_loss2 = torch.mean((Q2 - TQ)**2)
20     Q_loss = Q_loss1 + Q_loss2
21
22     # Gradient descent
23     agent.Q_optimizer.zero_grad()
24     Q_loss.backward()
25     agent.Q_optimizer.step()
```

trick! (why?)

← training critics



SAC - Implementation

```
1  actions, log_probs = agent.pi(obs_batch, with_log_prob=True)
2
3  freeze(agent.Q)
4  q1, q2 = agent.Q(obs_batch, actions)
5  q = torch.min(q1, q2)
6
7  # Need to perform gradient ascent, so (-) is required
8  # TODO: build policy loss
9  #pi_loss = torch.mean( #TODO )
10 pi_loss = torch.mean(agent.alpha * log_probs - q)
11
12 # Gradient ascent
13 agent.pi_optimizer.zero_grad()
14 pi_loss.backward()
15 agent.pi_optimizer.step()
```

← training actor



SAC - Experiment



SAC - Experiment



Is it successful on BipedalWalker-v3? (state dim : 24 / action dim : 4)

Thank you



CORE
Control + Optimization Research Lab