Day 6. Soft Actor-Critic (SAC)

NPEX Reinforcement Learning

2020.09.02

Jaeuk Shin

CORE
Control + Optimization Research Lab

SAC - Review

Control + Optimization Research Lab

SAC - Review

How to incentivize exploration?

idea : augment reward as follows:

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

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

How to solve new MDP with this novel reward criterion?

$\rightarrow$ soft Q-learning, soft Bellman equation, etc.

CORE
Control + Optimization Research Lab

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 :

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

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

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

```python
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
```

this is how we define twin critics!

```python
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
```

SAC - Implementation

actor definition - 1^{st} 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)

    sigma = torch.exp(log_sigma)
    distribution = Independent(Normal(mu, sigma), 1)
```

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

CORE
Control + Optimization Research Lab

SAC - Implementation

actor definition - 2nd step

```python
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)

    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)

return a, log_prob
```

needed for reparametrization trick

tricky part

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

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

SAC - Implementation

```
with torch.no_grad():
    next_actions, log_probs = self.pi(next_obs_batch, with_log_prob=True)
    target_q1, target_q2 = self.target_Q(next_obs_batch, next_actions)
    target_q = torch.min(target_q1, target_q2)
    target = rew_batch + self.gamma * masks * (target_q - self.alpha * log_probs)

out1, out2 = self.Q(obs_batch, act_batch)

Q_loss1 = torch.mean((out1 - target)**2)
Q_loss2 = torch.mean((out2 - target)**2)
Q_loss = Q_loss1 + Q_loss2

self.Q_optimizer.zero_grad()
Q_loss.backward()
self.Q_optimizer.step()
```

$\longleftarrow$ training critics

trick! (why?)

CORE

Control + Optimization Research Lab

SAC - Implementation

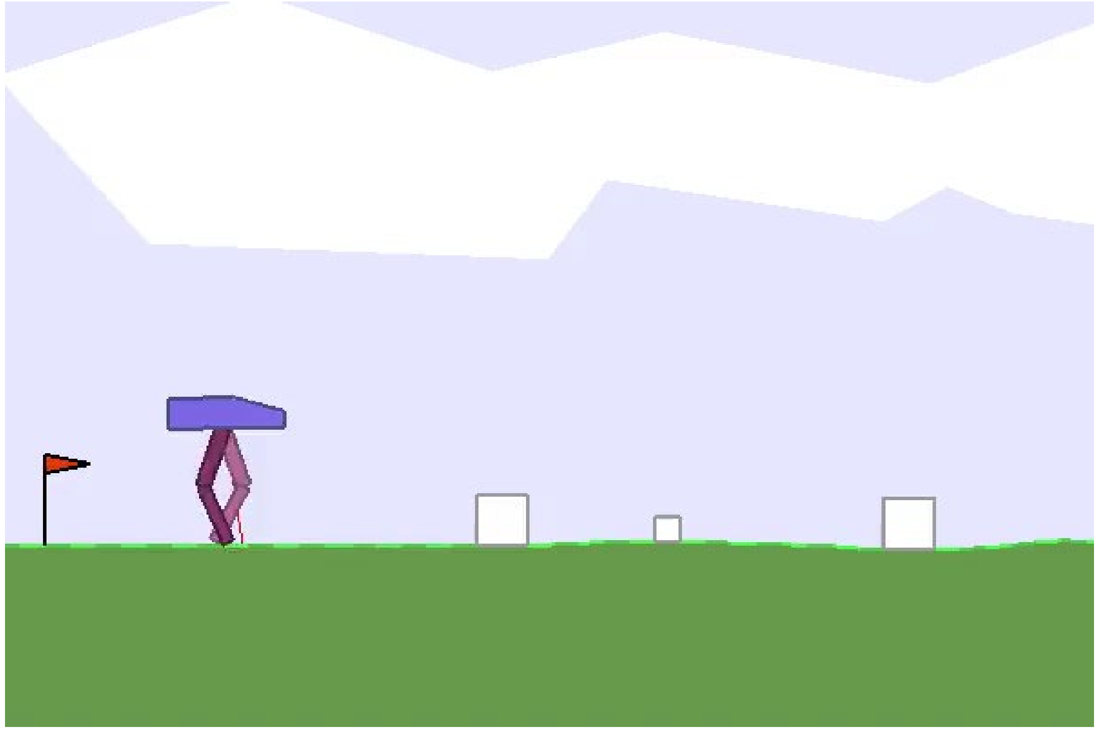
```
actions, log_probs = self.pi(obs_batch, with_log_prob=True)

freeze(self.Q)
q1, q2 = self.Q(obs_batch, actions)
q = torch.min(q1, q2)

pi_loss = torch.mean(self.alpha * log_probs - q)

self.pi_optimizer.zero_grad()
pi_loss.backward()
self.pi_optimizer.step()

unfreeze(self.Q)
```

$\longleftarrow$ training actor

CORE
Control + Optimization Research Lab

SAC - Experiment

SAC - Experiment



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

TRPO - Review

TRPO - Review

policy gradient (with importance sampling)

$$\nabla_\phi J(\phi) = \mathbb{E}_{s\sim\rho_{\phi_{\text{old}}},\ a\sim\pi_{\phi_{\text{old}}}}\left(A^{\pi_\phi}(s,a)\frac{\nabla_\phi \pi_\phi(a|s)}{\pi_{\phi_{\text{old}}}(a|s)}\right)$$

To estimate A^{π_ϕ}, we use a separate value function approximator $V(s;\theta)$, and apply **generalized advantage estimation(GAE)**!

Furthermore, we use a stochastic policy $\pi_\phi(a|s)$ (Gaussian in our case).

Plus, in TRPO, we have a lot of extra stuff to implement (Hessian-vector product, line search, etc.)

TRPO - Review

GAE?

Given a trajectory $(s_0, a_0, r_0, s_1, a_1, r_1, \cdots s_T)$ generated by executing the current policy, we first compute

$$\delta_t = r_t + \gamma V(s_{t+1}; \theta) - V(s_t; \theta)$$

Then, GAE is computed as follows:

$$\hat{A}(s_t, a_t) = \sum_{\tau=t}^{T-1} \gamma^{\tau-t} \delta_\tau$$

TRPO - Review

training $V(s;\theta)$?

given a trajectory $(s_0, a_0, r_0, s_1, a_1, r_1, \cdots s_T)$ generated from π_ϕ,
we can compute Monte-Carlo esimtates of $V(s_0), V(s_1), \cdots V(s_T)$
as follows:

$$V(s_t) = \sum_{\tau=t}^{T-1} \gamma^{\tau-t} r_\tau + \gamma^T V(s_T)$$

This is the **target** for value function update!

TRPO - Implementation

TRPO - Implementation

What info do we need when we implement **on-policy algorithms**?

1. (s_j, a_j, s'_j, r_j)
2. generalized advantage estimation(GAE)
3. MC estimates of value $V^{\pi_\phi}(s)$
4. probability $\pi_{\phi_{\text{old}}}(a_j|s_j)$

TRPO - Implementation

```
self._obs_mem = np.zeros(shape=(lim, dimS))
self._act_mem = np.zeros(shape=(lim, dimA))
self._rew_mem = np.zeros(shape=(lim,))
self._val_mem = np.zeros(shape=(lim,))
self._log_prob_mem = np.zeros(shape=(lim,))
```

collected during agent-env interaction

```
# memory of cumulative rewards which are MC-estimates of the current value function
self._target_v_mem = np.zeros(shape=(lim,))
# memory of GAE($\lambda$)-estimate of the current advantage function
self._adv_mem = np.zeros(shape=(lim,))
```

computed when sampling a single episode is done

Thank you