

SAC Code Practice

Dongmin Lee

RL Korea Bootcamp

October 27, 2019

Outline

- Soft Actor-Critic (SAC)
- Model
- Learning Process
- Train & Test

Soft Actor-Critic (SAC)

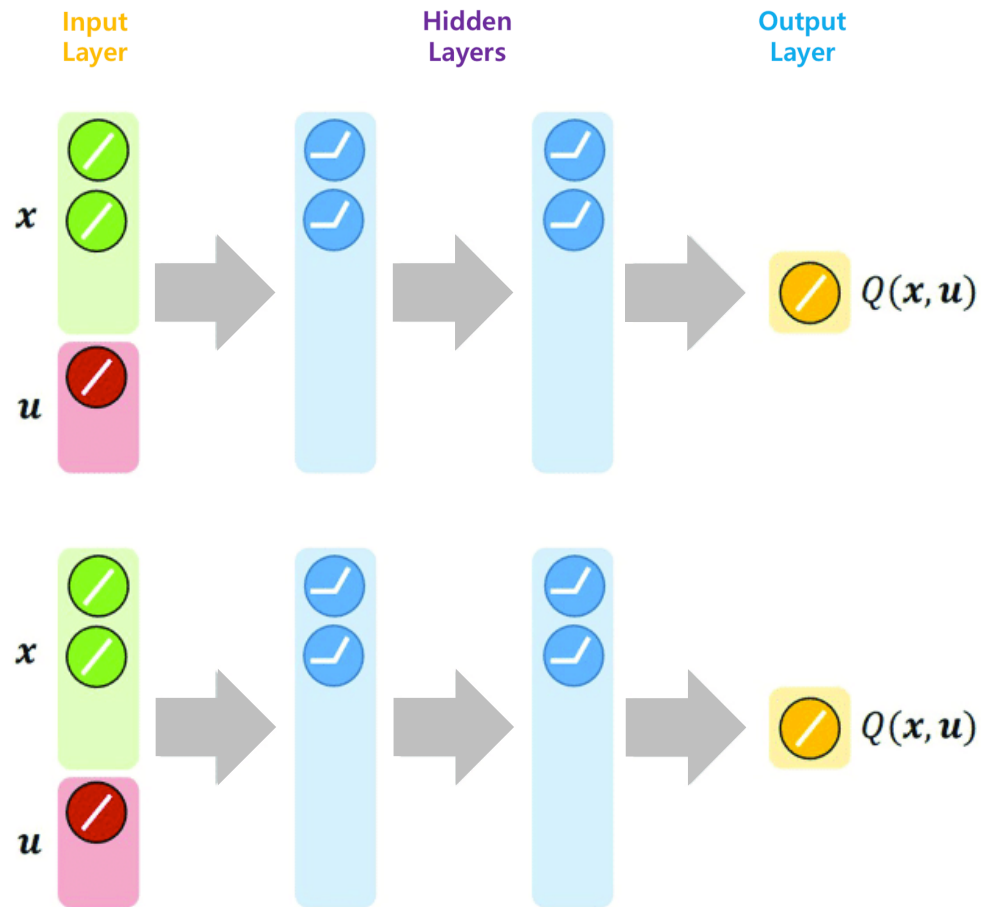
Automating entropy adjustment on SAC (ASAC)

Algorithm 1 Soft Actor-Critic

Input: θ_1, θ_2, ϕ ▷ Initial parameters
 $\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$ ▷ Initialize target network weights
 $\mathcal{D} \leftarrow \emptyset$ ▷ Initialize an empty replay pool
for each iteration **do**
 for each environment step **do**
 $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t | \mathbf{s}_t)$ ▷ Sample action from the policy
 $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$ ▷ Sample transition from the environment
 $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$ ▷ Store the transition in the replay pool
 end for
 for each gradient step **do**
 $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ ▷ Update the Q-function parameters
 $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ ▷ Update policy weights
 $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$ ▷ Adjust temperature
 $\bar{\theta}_i \leftarrow \tau \theta_i + (1 - \tau) \bar{\theta}_i$ for $i \in \{1, 2\}$ ▷ Update target network weights
 end for
end for
Output: θ_1, θ_2, ϕ ▷ Optimized parameters

Model

Critic network



Model

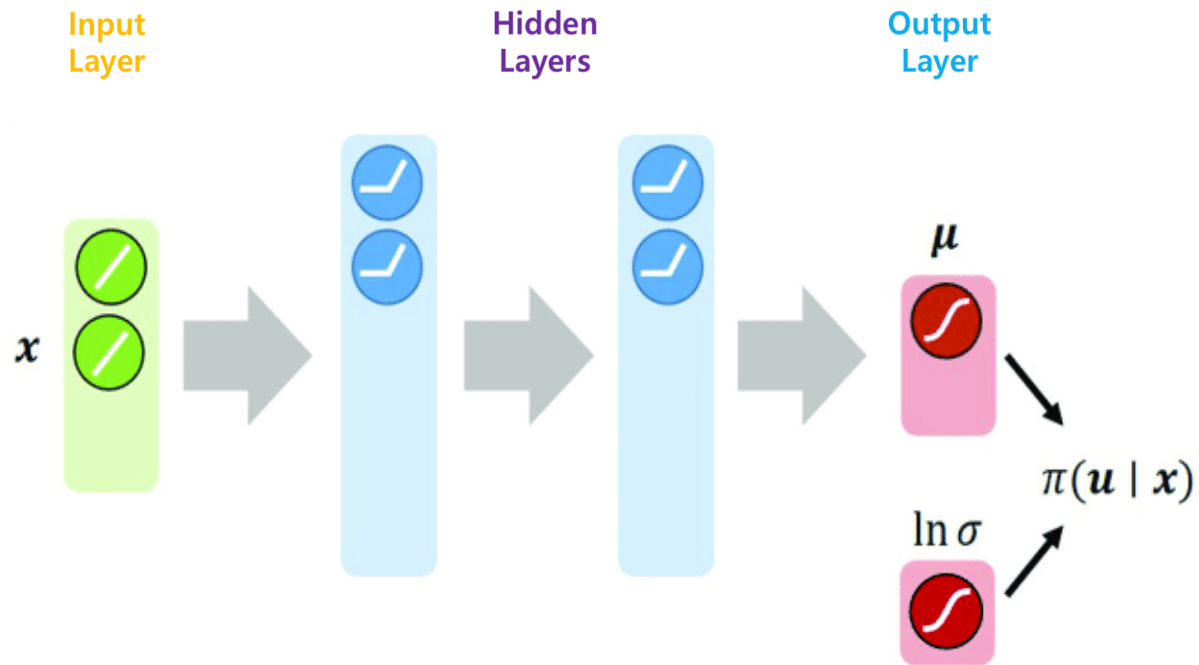
Critic network (model.py)

```
11 class MLP(nn.Module):
12     def __init__(self,
13                 input_size,
14                 output_size,
15                 hidden_sizes=(128,128),
16                 activation=F.relu,
17                 output_activation=identity,
18                 use_output_layer=True,
19     ):
20         super(MLP, self).__init__()
21
22         self.input_size = input_size
23         self.output_size = output_size
24         self.hidden_sizes = hidden_sizes
25         self.activation = activation
26         self.output_activation = output_activation
27         self.use_output_layer = use_output_layer
28
29         # Set hidden layers
30         self.hidden_layers = nn.ModuleList()
31         in_size = self.input_size
32         for next_size in self.hidden_sizes:
33             fc = nn.Linear(in_size, next_size)
34             in_size = next_size
35             self.hidden_layers.append(fc)
36
37         # Set output layers
38         if self.use_output_layer:
39             self.output_layer = nn.Linear(in_size, self.output_size)
40         else:
41             self.output_layer = identity
42
43     def forward(self, x):
44         for hidden_layer in self.hidden_layers:
45             x = self.activation(hidden_layer(x))
46         x = self.output_activation(self.output_layer(x))
47         return x
```

```
50 class FlattenMLP(MLP):
51     def forward(self, x, a):
52         q = torch.cat([x,a], dim=-1)
53         return super(FlattenMLP, self).forward(q)
```

Model

Actor network



Model

Actor network (model.py)

```
56 LOG_STD_MAX = 2
57 LOG_STD_MIN = -20
58
59 class GaussianPolicy(MLP):
60     def __init__(self,
61                 input_size,
62                 output_size,
63                 hidden_sizes=(128,128),
64                 activation=F.relu,
65             ):
66         super(GaussianPolicy, self).__init__(
67             input_size=input_size,
68             output_size=output_size,
69             hidden_sizes=hidden_sizes,
70             activation=activation,
71             use_output_layer=False,
72         )
73
74         in_size = hidden_sizes[-1]
75
76         # Set output layers
77         self.mu_layer = nn.Linear(in_size, output_size)
78         self.log_std_layer = nn.Linear(in_size, output_size)
79
80     def clip_but_pass_gradient(self, x, l=-1., u=1.):
81         clip_up = (x > u).float()
82         clip_low = (x < l).float()
83         clip_value = (u - x)*clip_up + (l - x)*clip_low
84         return x + clip_value.detach()
85
86     def apply_squashing_func(self, mu, pi, log_pi):
87         mu = torch.tanh(mu)
88         pi = torch.tanh(pi)
89         # To avoid evil machine precision error, strictly clip 1-pi**2 to [0,1] range.
90         log_pi -= torch.sum(torch.log(self.clip_but_pass_gradient(1 - pi.pow(2), l=0., u=1.) + 1e-6), dim=-1)
91         return mu, pi, log_pi
92
93     def forward(self, x):
94         x = super(GaussianPolicy, self).forward(x)
95
96         mu = self.mu_layer(x)
97         log_std = torch.tanh(self.log_std_layer(x))
98         log_std = LOG_STD_MIN + 0.5 * (LOG_STD_MAX - LOG_STD_MIN) * (log_std + 1)
99         std = torch.exp(log_std)
100
101         dist = Normal(mu, std)
102         pi = dist.rsample() # reparameterization trick (mean + std * N(0,1))
103         log_pi = dist.log_prob(pi).sum(dim=-1)
104         mu, pi, log_pi = self.apply_squashing_func(mu, pi, log_pi)
105
106         return mu, pi, log_pi
```

Learning Process

1. Collect experience (s_t, a_t, r_t, s_{t+1}) using some policy, add them to **Buffer**;
2. Sample a mini-batch $\{(s_i, a_i, r_i, s_{i+1})\}$ from **Buffer**;
3. Set **target** $y_i^- \triangleq r_i + \gamma(Q_{\theta^-}(s_{i+1}, \pi_{\phi}(s_{i+1})) - \alpha \log \pi_{\phi}(a_{i+1}|s_{i+1}))$;
4. Update **the critic network** by minimizing
$$J_Q(\theta) \triangleq \mathbb{E} \left[\frac{1}{2} (Q_{\theta}(s_i, a_i) - y_i^-)^2 \right];$$
5. Update **the actor network** by minimizing
$$J_{\pi}(\phi) \triangleq \mathbb{E} [\alpha \log \pi_{\phi}(a_i|s_i) - Q_{\theta}(s_i, a_i)];$$
6. Update **the alpha** by minimizing $\mathbb{E} [-\alpha \log \pi_{\phi}(a_i|s_i) - \alpha H_0]$
7. Update **the target networks**: $\theta^- \leftarrow \tau \theta + (1 - \tau) \theta^-$ with small τ ;

Learning Process

1. Collect experience (s_t, a_t, r_t, s_{t+1}) using some policy, add them to **Buffer**;
2. Sample a mini-batch $\{(s_i, a_i, r_i, s_{i+1})\}$ from **Buffer**;

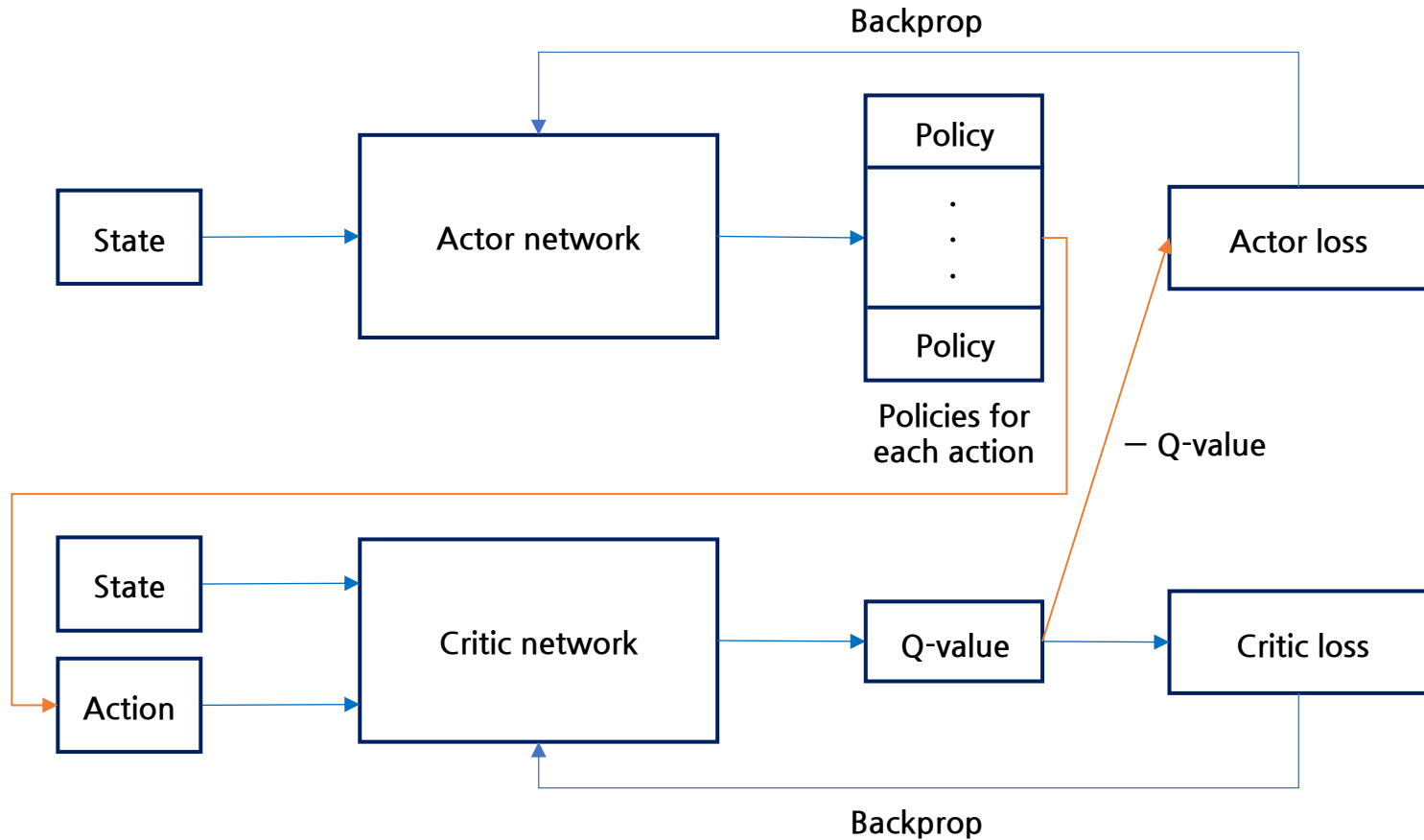
```
173 # Collect experience (s, a, r, s') using some policy
174 _, action, _ = actor(torch.Tensor(obs).to(device))
175 action = action.detach().cpu().numpy()
176 next_obs, reward, done, _ = env.step(action)
177
178 # Add experience to replay buffer
179 replay_buffer.add(obs, action, reward, next_obs, done)
180
181 # Start training when the number of experience is greater than batch size
182 if steps > args.batch_size:
183     batch = replay_buffer.sample(args.batch_size)
184     train_model(actor, qf1, qf2, qf1_target, qf2_target,
185               actor_optimizer, qf1_optimizer, qf2_optimizer,
186               batch, target_entropy, log_alpha, alpha_optimizer)
```

Learning Process

3. Set **target** $y_i^- \triangleq r_i + \gamma(Q_{\theta^-}(s_{i+1}, \pi_{\phi}(s_{i+1})) - \alpha \log \pi_{\phi}(a_{i+1}|s_{i+1}));$
4. Update **the critic network** by minimizing
$$J_Q(\theta) \triangleq \mathbb{E} \left[\frac{1}{2} (Q_{\theta}(s_i, a_i) - y_i^-)^2 \right];$$
5. Update **the actor network** by minimizing
$$J_{\pi}(\phi) \triangleq \mathbb{E} [\alpha \log \pi_{\phi}(a_i|s_i) - Q_{\theta}(s_i, a_i)];$$

```
54     # Prediction  $\pi(s)$ ,  $\log \pi(s)$ ,  $\pi(s')$ ,  $\log \pi(s')$ ,  $Q1(s,a)$ ,  $Q2(s,a)$ 
55     _, pi, log_pi = actor(obs1)
56     _, next_pi, next_log_pi = actor(obs2)
57     q1 = qf1(obs1, acts).squeeze(1)
58     q2 = qf2(obs1, acts).squeeze(1)
59
60     # Min Double-Q:  $\min(Q1(s, \pi(s)), Q2(s, \pi(s))), \min(Q1^-(s', \pi(s')), Q2^-(s', \pi(s')))$ 
61     min_q_pi = torch.min(qf1(obs1, pi), qf2(obs1, pi)).squeeze(1).to(device)
62     min_q_next_pi = torch.min(qf1_target(obs2, next_pi), qf2_target(obs2, next_pi)).squeeze(1).to(device)
63
64     # Targets for Q and V regression
65     v_backup = min_q_next_pi - args.alpha*next_log_pi
66     q_backup = rews + args.gamma*(1-done)*v_backup
67     q_backup.to(device)
68
69 > if 0: # Check shape of prediction and target...
70
71
72
73
74
75
76
77
78     # Soft actor-critic losses
79     actor_loss = (args.alpha*log_pi - min_q_pi).mean()
80     qf1_loss = F.mse_loss(q1, q_backup.detach())
81     qf2_loss = F.mse_loss(q2, q_backup.detach())
```

Learning Process



Learning Process

6. Update the alpha by minimizing $\mathbb{E}[-\alpha \log \pi_{\phi}(a_i | s_i) - \alpha H_0]$

```
148     # If automatic entropy tuning is True, initialize a target entropy, a log alpha and an alpha optimizer
149     if args.automatic_entropy_tuning:
150         target_entropy = -np.prod(act_dim,).item()
151         log_alpha = torch.zeros(1, requires_grad=True, device=device)
152         alpha_optimizer = optim.Adam([log_alpha], lr=args.alpha_lr)
153     else:
154         target_entropy = None
155         log_alpha = None
156         alpha_optimizer = None
```

```
97     # If automatic entropy tuning is True, update alpha
98     if args.automatic_entropy_tuning:
99         alpha_loss = -(log_alpha * (log_pi + target_entropy).detach()).mean()
100         alpha_optimizer.zero_grad()
101         alpha_loss.backward()
102         alpha_optimizer.step()
103
104     args.alpha = log_alpha.exp()
```

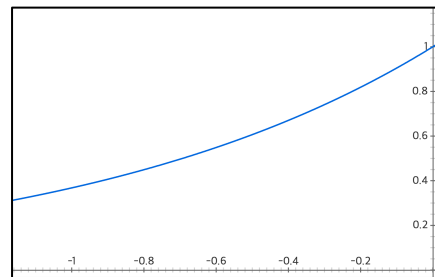
```
log_alpha tensor([0.], requires_grad=True)
alpha tensor([1.], grad_fn=<ExpBackward>)

log_alpha tensor([-0.0001], requires_grad=True)
alpha tensor([0.9999], grad_fn=<ExpBackward>)

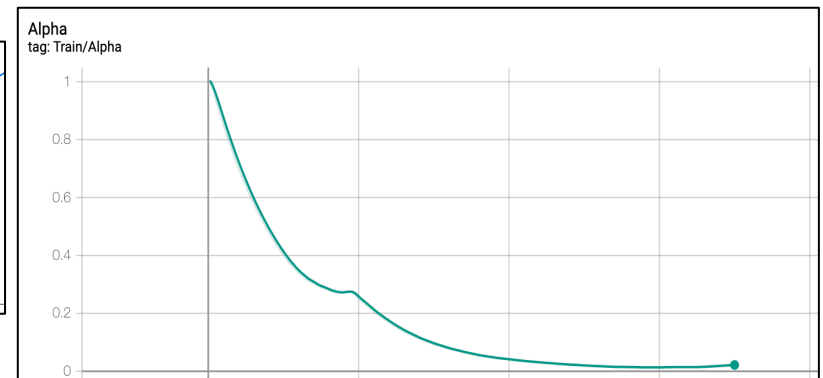
log_alpha tensor([-0.0002], requires_grad=True)
alpha tensor([0.9998], grad_fn=<ExpBackward>)

log_alpha tensor([-0.0003], requires_grad=True)
alpha tensor([0.9997], grad_fn=<ExpBackward>)

log_alpha tensor([-0.0004], requires_grad=True)
alpha tensor([0.9996], grad_fn=<ExpBackward>)
```



$y = \exp(x)$



Learning Process

7. Update the target networks: $\theta^- \leftarrow \tau\theta + (1 - \tau)\theta^-$ with small τ ;

```
106     # Polyak averaging for target parameter
107     soft_target_update(qf1, qf1_target)
108     soft_target_update(qf2, qf2_target)
```

```
34 def soft_target_update(main, target, tau=0.005):
35     for main_param, target_param in zip(main.parameters(), target.parameters()):
36         target_param.data.copy_(tau*main_param.data + (1.0-tau)*target_param.data)
```

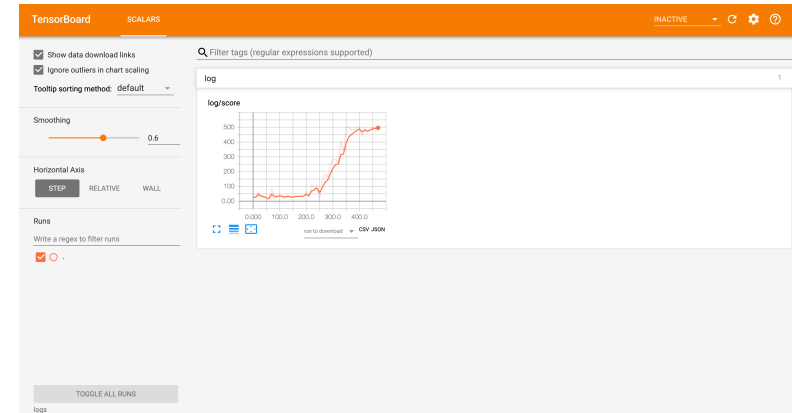
Train & Test

Terminal A - train

- `conda activate env_name`
- `python train.py`
- `python test.py` (after training)

Terminal B - Tensorboard

- `conda activate env_name`
- `tensorboard --logdir=runs`
- `localhost:6006` (in a web browser)



Thank You!