(soft, smoth, stable)

SAC： 引入最大熵，使action探索能力加强，在PPO里，当在线和离线策略的KL最小，则取样一致，

SAC是学习一个分布，每条数据的不确定性 $=-\log\pi(a|s)$，尽量选 Entropy 大的 batch

```python
def actor_learn(self, obs):
    action, log_pi = self.sample(obs)
    qf1_pi, qf2_pi = self.critic.value(obs, action)
    min_qf_pi = layers.elementwise_min(qf1_pi, qf2_pi)
    cost = log_pi * self.alpha - min_qf_pi
    cost = layers.reduce_mean(cost)
    optimizer = fluid.optimizer.AdamOptimizer(self.actor_lr)
    optimizer.minimize(cost, parameter_list=self.actor.parameters())

    return cost
```

采样扰动 $\{ N=\eta(u, e^{logstd})$; $x = N.sample()$; action $= \tanh(x)\cdot a_{max}$; logprob $= -\sum \log[a_{max}\cdot(1-\tanh(x)^2)+\varepsilon]$

$\alpha = 0.2$

在TD3里，actor cost $=-Q(s,a)$，这里加了entropy

```python
def critic_learn(self, obs, action, reward, next_obs, terminal):
    next_obs_action, next_obs_log_pi = self.sample(next_obs)
    qf1_next_target, qf2_next_target = self.target_critic.value(
        next_obs, next_obs_action)
    min_qf_next_target = layers.elementwise_min(
        qf1_next_target, qf2_next_target) - next_obs_log_pi * self.alpha

    terminal = layers.cast(terminal, dtype='float32')
    target_Q = reward + (1.0 - terminal) * self.gamma * min_qf_next_target
    target_Q.stop_gradient = True

    current_Q1, current_Q2 = self.critic.value(obs, action)
    cost = layers.square_error_cost(current_Q1,
                                    target_Q) + layers.square_error_cost(
                                        current_Q2, target_Q)
    cost = layers.reduce_mean(cost)
    optimizer = fluid.optimizer.AdamOptimizer(self.critic_lr)
    optimizer.minimize(cost)
    return cost
```

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D}}\left[\mathrm{D}_{\mathrm{KL}}\left(\pi_\phi(\cdot|\mathbf{s}_t)\,\middle\|\,\frac{\exp(Q_\theta(\mathbf{s}_t,\cdot))}{Z_\theta(\mathbf{s}_t)}\right)\right]$$

$\mathbf{a}_t = f_\phi(\epsilon_t;\mathbf{s}_t)$, 采样增加扰动

分母为析分函数，为分母的积分，只与s有关，与φ无关

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D},\epsilon_t\sim\mathcal{N}}\left[\log\pi_\phi(f_\phi(\epsilon_t;\mathbf{s}_t)|\mathbf{s}_t) - Q_\theta(\mathbf{s}_t, f_\phi(\epsilon_t;\mathbf{s}_t))\right]$$

$$\hat\nabla_\phi J_\pi(\phi) = \nabla_\phi \log\pi_\phi(\mathbf{a}_t|\mathbf{s}_t) + (\nabla_{\mathbf{a}_t}\log\pi_\phi(\mathbf{a}_t|\mathbf{s}_t) - \nabla_{\mathbf{a}_t}Q(\mathbf{s}_t,\mathbf{a}_t))\nabla_\phi f_\phi(\epsilon_t;\mathbf{s}_t)$$

Algorithm 1 Soft Actor-Critic　　$A(s)$ = mean, logstd $\in(-20, 2)$

```
Input: θ1, θ2, φ                                          ▷ Initial parameters
  θ̄1 ← θ1, θ̄2 ← θ2                              ▷ Initialize target network weights
  D ← ∅                                          ▷ Initialize an empty replay pool
  for each iteration do
    for each environment step do
      a_t ~ π_φ(a_t|s_t)                           ▷ Sample action from the policy
      s_{t+1} ~ p(s_{t+1}|s_t, a_t)     ▷ Sample transition from the environment
      D ← D ∪ {(s_t, a_t, r(s_t, a_t), s_{t+1})}  ▷ Store the transition in the replay pool
    end for
    for each gradient step do
      θ_i ← θ_i − λ_Q ∇̂_{θ_i} J_Q(θ_i) for i ∈ {1, 2}   ▷ Update the Q-function parameters
      φ ← φ − λ_π ∇̂_φ J_π(φ)                              ▷ Update policy weights
      α ← α − λ ∇̂_α J(α)                                  ▷ Adjust temperature
      θ̄_i ← τθ_i + (1 − τ)θ̄_i for i ∈ {1, 2}         ▷ Update target network weights
    end for
  end for
Output: θ1, θ2, φ                                        ▷ Optimized parameters
```

$y = r + \gamma\cdot\min\{Q_\phi(s', a') - \log\pi\}$

a_t也是取样得到的，

$$J_V(\psi) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D}}\left[\tfrac{1}{2}\left(V_\psi(\mathbf{s}_t) - \mathbb{E}_{\mathbf{a}_t\sim\pi_\phi}[Q_\theta(\mathbf{s}_t,\mathbf{a}_t) - \log\pi_\phi(\mathbf{a}_t|\mathbf{s}_t)]\right)^2\right]$$

s来自缓存池，a来自当前policy π的采样

$$\hat\nabla_\psi J_V(\psi) = \nabla_\psi V_\psi(\mathbf{s}_t)\left(V_\psi(\mathbf{s}_t) - Q_\theta(\mathbf{s}_t,\mathbf{a}_t) + \log\pi_\phi(\mathbf{a}_t|\mathbf{s}_t)\right)$$

行为策略： alg.sample(s) 左图

评价策略： $\tanh(A(s))\cdot a_{max}$ ← ??

(模型)　(更新)

前31万步， actor-lr : 1e-3　critic-lr : 1e-3　γ : 0.99　tau : 0.8

noise : 0.25　batch size : 256

后面： actor_lr : 5e-3　critic-lr : 5e-3　γ : 0.999　tau : 0.01

noise : 1　batch size : 256　不理解上面为啥动作噪音反而增加？

fc1 : 100　$A(s)\in(-1, 1)$

加噪： a = np.clip(gaussian(a, noise), -1, 1) 这噪音好大　探索加噪

For DDPG: $a' = Q_{\theta'}(s')$　$y = r + \gamma\cdot V_{\phi'}(s', a')$

DDPG

```python
def _critic_learn(self, obs, action, reward, next_obs, terminal):
    next_action = self.target_model.policy(next_obs)
    next_Q = self.target_model.value(next_obs, next_action)

    terminal = layers.cast(terminal, dtype='float32')
    target_Q = reward + (1.0 - terminal) * self.gamma * next_Q
    target_Q.stop_gradient = True

    Q = self.model.value(obs, action)
    cost = layers.square_error_cost(Q, target_Q)
    cost = layers.reduce_mean(cost)
    optimizer = fluid.optimizer.AdamOptimizer(self.critic_lr)
    optimizer.minimize(cost)
    return cost
```

TD3算法（延迟更新）

```python
def critic_learn(self, obs, action, reward, next_obs, terminal):
    noise = layers.gaussian_random_batch_size_like(
        action, shape=[-1, action.shape[1]])
    noise = layers.clip(
        noise * self.policy_noise, #0.2
        min=-self.noise_clip, #0.5
        max=self.noise_clip)
    next_action = self.target_model.policy(next_obs) + noise
    next_action = layers.clip(next_action, -self.max_action,
                              self.max_action)

    next_Q1, next_Q2 = self.target_model.value(next_obs, next_action)
    next_Q = layers.elementwise_min(next_Q1, next_Q2)

    terminal = layers.cast(terminal, dtype='float32')
    target_Q = reward + (1.0 - terminal) * self.gamma * next_Q
    target_Q.stop_gradient = True

    current_Q1, current_Q2 = self.model.value(obs, action)
    cost = layers.square_error_cost(current_Q1,
                                    target_Q) + layers.square_error_cost(
                                        current_Q2, target_Q)
    cost = layers.reduce_mean(cost)
    optimizer = fluid.optimizer.AdamOptimizer(self.critic_lr)
    optimizer.minimize(cost)
    return cost
```

noise = clip(0.2ε, -0.5, 0.5)

buffer : 1e6　batch : 256　γ = 0.95　tau = 0.005　$A(s)\in(-max, max)$

actor_lr : 1e-3　critic_lr : 1e-3　fc1 : 300　fc2 : 300

OU-noise : $\mu=0$, $\theta=0.15$, $\sigma=0.2$　就用这套吧

(两)次critic更新，行为策略才更新一次　　min{a, b} : elementwise min

For TD3: $a' = \mathrm{clip}(V_{\theta'}(s') + noise)$　$V_1', V_2' = Q_{\phi'}(s', a')$

$y = r + \gamma\cdot\min\{V_1', V_2'\}$　$\tilde y_1, \tilde y_2 = Q_\phi(s, a)$

$cost = MSE(\tilde y_1, y) + MSE(\tilde y_2, y)$

如果在前20万step训练中，test分数没有上到1000，train分数没上到0就放弃

行为策略： $\mathrm{clip}(a + \eta(0, 0.1a_{max}), -a_{max}, a_{max})$

在价值函数的更新目标上，也增加了扰动，使价值函数更robust

critic 可以相比 actor 学慢一点，SAC 貌似最好，可以和 DDPG 比一下，