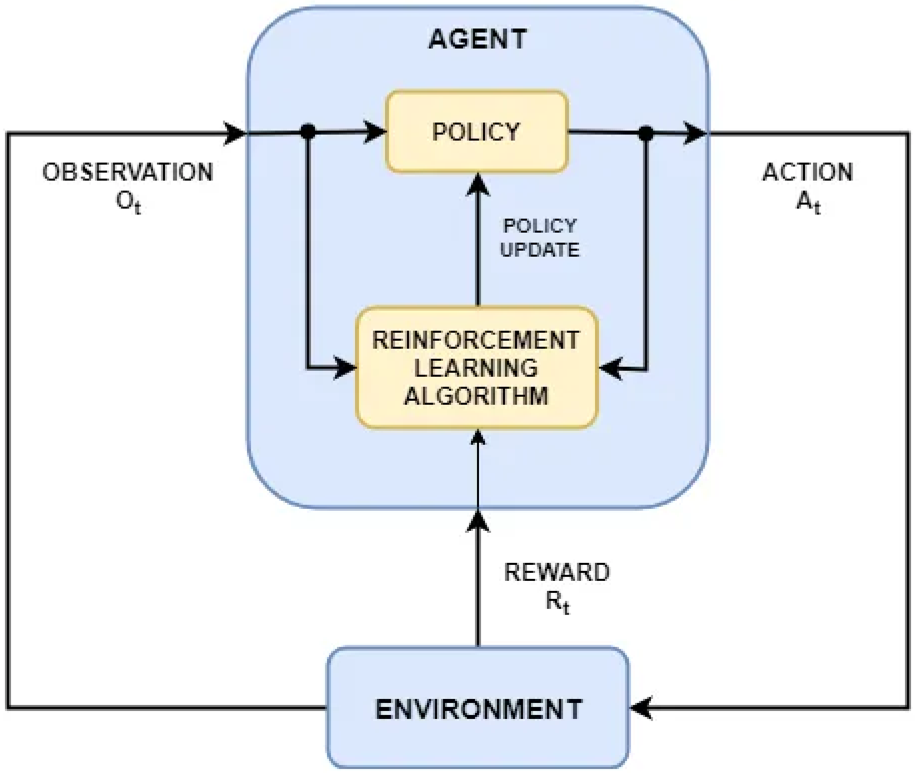
Project

RL: State, Action and Environment



State

- Representation of Environment.
- Focuses on changes due to actions.
- They are discrete and mutually exclusive.

The set of all valid actions in a given environment is called the action space

Reward

Determines how good was the action from state s to reach the next state

$$r_t = R(s_t, a_t, s_{t+1})$$

Agent Policy (π)

A function mapping State Space (S) to Action Space (A). It Can be Probabilistic or Deterministic.

Deterministic Action Policy

The deterministic policy output an action with probability one. Agent always choose an action without considering any uncertainties

$$\mu(s) = a$$

Stochastic (Probabilistic) Action Policy

Output the probability distribution over the actions from states

$$\pi(a, s) = Pr(a|s)$$

Trajectory τ is the sequence of states and actions.

P(s’, r **|** s, a): Probability of getting state s’ and reward r given state s and action a

Expectation Values: Probability of something multiplied by what we get

E(S) = Σr Σp(s’, r **|** s, a)

Markov Decision Process

State Depending on previous state and action only

- $P_a(s, s') = Pr(s(t+1) = s' | s_t = s, a_t = a)$ is probability that action a in state s leads to s’
- $R_a(s, s')$ is immediate reward received after transitioning from s to s’ due to a

The objective is to choose a policy π that will the expected discounted sum over a potentially infinite horizon:

$$V^{\pi}(s) = \mathbb{E}[\sum_{i=1}^{T} \gamma^{i-1} r_i] \quad \forall s \in \mathbb{S}$$

π

A lower discount factor motivates decision maker to favour taking actions early, rather than postpone them indefinitely

Policy Gradient Methods

Policy gradient methods are policy iterative method that means modelling and optimising the policy directly. The random policy is selected initially and find the value function of that policy in the evaluation step. Then find the new policy from the value function computed in the improve step. The process repeats until it finds the optimal policy.

The main challenge of policy gradient RL is the high gradient variance. The standard approach to reduce the variance in gradient estimate is to use baseline function

Value-Based Methods

In a value-based approach, the random value function is selected initially, then find new value function. This process repeated until it finds the optimal value function. The intuition here is the policy that follows the optimal value function will be optimal policy. Here, the policy is implicitly updated through value function.

Episodic Rewards

Episodes are sequence of states and actions. Hence, Episodic Rewards are discrete periods of play giving sequence of Rewards

Finite Horizon Undiscounted Return

Sum of reward from the current state to goal state which has a fixed time-step

$R(t) = \Sigma r_t$

Reward Discounting : Infinite-Horizon Discounted Return

γ Reflects Uncertainty in Future Rewards (0≤γ≤1).

γ = 0 : Agent only care about immediate rewards

γ = 1 : All future rewards are taken into consideration

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t.$$

So,

$R(t) = r(t+1) + \gamma R(t+1)$

State-Value Function

$V_\pi(s) = E_\pi[R_t | S_t = s]$

Gives us expected Total Reward of agents returning from time t and state s following policy π

$$V^\pi(s) = \mathbb{E}[\sum_{i=1}^{T} \gamma^{i-1} r_i] \quad \forall s \in \mathbb{S}$$

Optimal state-value function is the max of state-value function and on the policy on which it is obtained is optimal policy.

Action-Value Function

$Q_\pi(s, a) = E_\pi[R_t | S_t = s, A_t = a]$

Gives us expected return for an agent returning from time t, state s and action a following policy π. The state can have multiple actions, thus there will be multiple Q value in a state.

Optimal action-value function is the max of action-value function and on the Q-function on which it is obtained is optimal Q-function.

Q-Learning

Value-based off-policy temporal difference(TD) RL

Off-policy : An agent follows a behaviour policy for choosing the action to reach the next state $S(t+1)$ from state $S(t)$

$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a)$$

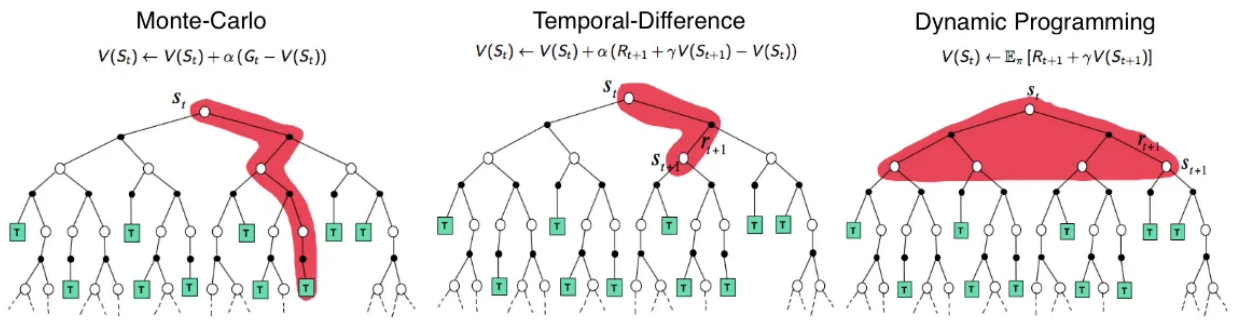
From S(t+1), it uses a policy π that is different from behaviour policy. In Q-learning, we take absolute greedy action as policy π from the next state S(t+1)

$$Q^{new}(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t,a_t)}_{\text{old value}} \right)}^{\text{TD error}}$$

Here, TD Error is new Q value - Old Q Value. TD Error is also called Advantage Function.

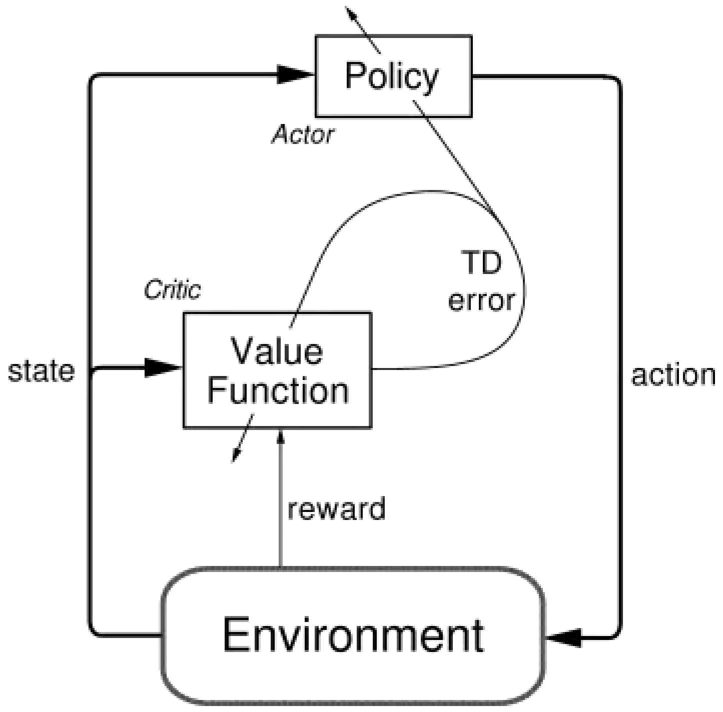
Actor-Critic Model

Belong to Temporal Difference (TD) Learning Algorithm of Policy Gradient which is combination of Monte-Carlo and Dynamic programming Methods. TD algorithm estimates difference in values of successive states.


Monte-Carlo
$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$
Temporal-Difference
$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
Dynamic Programming
$V(S_t) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1})]$
s_t
r_{t+1}
s_{t+1}
T

- Actor-critic is similar to a policy gradient algorithm called REINFORCE with baseline. Reinforce is the MONTE-CARLO learning that indicates that total return is sampled from the full trajectory. But in actor-critic, we use bootstrap.
- Learning of the actor is based policy gradient approach and critic is learned in value-based fashion.

Two Cost functions for actor and critic each



$$\delta = r_t + \gamma V(S(t+1)) - V(S(t))$$

Critic Loss : δ^2

Actor Loss: $\delta ln\pi(A_t|S_t)$

Drawbacks of Actor-Critic Model

- Convergence: High Degree of Hyper parameter Training
- High Sample Complexity

Pseudocode

```
1. Sample {$S_t, A_t$} using policy π from Actor Network
2. Evaluate Advantage Function $A_t$ (TD Error $\delta_t$) produced by Critic Network
3. Evaluate Gradient
       $\nabla J(\theta) \approx \sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t, s_t) A_{\pi_\theta}(s_t, a_t)$
4. Update Policy Parameters
       $\theta = \theta + \alpha \nabla J(\theta)$
5. Update the weights of the critic based value-based RL(Q-learning)
       $w = w + \alpha\delta_t$
6. Repeat 1-5 until we find optimal policy
```

Soft Actor Critic Model

Objective Function of Maximum Entropy RL:

$$J(\pi_\theta) = E_{\pi_\theta}[\textstyle\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t) + \alpha H(\pi(.|s_t))]$$

- Learning of Actor : Policy Gradient Based and Learning of Critic : Value Based

- Three networks: the first network represents state-value(V) parameterised by ψ, the second one is a policy function that parameterised by φ, and the last one represents soft q function parameterised by θ.

We are using five neural networks architecture consisting of

1. An Actor Network - Obtains probability distribution of the actions

2. Two V Critic Networks - Estimating State Values (One is Evaluation Network and other is Target Network and target network is updated using Soft Update)

3. Two Q Critic Networks - Estimating State Action Values (Trained using same update method and MSE Loss Function)

- A central feature of SAC is entropy regularisation. The policy is trained to maximise a trade-off between expected return and entropy, a measure of randomness in the policy. The entropy regularisation coefficient α explicitly controls the explore-exploit tradeoff, with higher α corresponding to more exploration, and lower α corresponding to more exploitation.

> Maximum Entropy RL enables the agent to explore more and avoid converging to local optima

- The model is similar to Q Learning with the Epsilon Greedy Method and incorporates reward scaling to model entropy. It utilises two critics like Double Q learning and employs a target value function for soft updating.

State-Value Function (V_ψ)

State-Value Function is learnt by minimising squared residual error.

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

So, the gradient objective of State-Value Function:

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

Q Function (Q_θ)

Q Function is learnt by minimising the squared difference between predicted Q value and reward plus the discounted expectation of state-value of next state.

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\theta(\mathbf{s}_t,\mathbf{a}_t) - \hat{Q}(\mathbf{s}_t,\mathbf{a}_t)\right)^2\right],$$

where,

$$\hat{Q}(\mathbf{s}_t,\mathbf{a}_t) = r(\mathbf{s}_t,\mathbf{a}_t) + \gamma\,\mathbb{E}_{\mathbf{s}_{t+1}\sim p}\left[V_{\bar{\psi}}(\mathbf{s}_{t+1})\right],$$

So, the gradient objective of Q Function:

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(\mathbf{a}_t,\mathbf{s}_t)\left(Q_\theta(\mathbf{s}_t,\mathbf{a}_t) - r(\mathbf{s}_t,\mathbf{a}_t) - \gamma V_{\bar{\psi}}(\mathbf{s}_{t+1})\right).$$

Policy Function (Π_Φ)

Policy function is learnt by

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

where,

$$\mathbf{a}_t = f_\phi(\epsilon_t;\mathbf{s}_t),$$

So, the gradient objective of Policy Function:

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

Hence, policy learning depends on the Q-value to reduce the variance as it is in typical Actor-Critic Model.

Pseudocode

Algorithm 1 Soft Actor-Critic

```
1: Input: initial policy parameters $\theta$, Q-function parameters $\phi_1$, $\phi_2$, empty replay buffer $\mathcal{D}$
2: Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
3: repeat
4:    Observe state $s$ and select action $a \sim \pi_\theta(\cdot|s)$
5:    Execute $a$ in the environment
6:    Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
7:    Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
8:    If $s'$ is terminal, reset environment state.
9:    if it's time to update then
10:       for $j$ in range(however many updates) do
11:          Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12:          Compute targets for the Q functions:
                $y(r, s', d) = r + \gamma(1-d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$
13:          Update Q-functions by one step of gradient descent using
                $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$
14:          Update policy by one step of gradient ascent using
                $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s) |\, s\right)\right),$
             where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt $\theta$ via the
             reparametrization trick.
15:          Update target networks with
                $\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1-\rho)\phi_i \qquad \text{for } i = 1, 2$
16:       end for
17:    end if
18: until convergence
```

Policy Optimisation

Optimal policy that maximises the sum of the cumulative reward value and the policy's entropy value:

$$\pi^* = \arg\max_\pi \mathop{\mathrm{E}}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t \left(R(s_t, a_t, s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right)\right)\right],$$

where α>0 is temperature coefficient (trade-off coefficient)

And, The entropy H of x is computed from its distribution P according to

$$H(P) = \underset{x \sim P}{\mathrm{E}} \left[-\log P(x) \right].$$

We can now define the slightly different value functions in this setting. V^{π} is changed to include the entropy bonuses from every time-step:

$$V^{\pi}(s) = \underset{\tau \sim \pi}{\mathrm{E}} \left[\sum_{t=0}^{\infty} \gamma^t \bigg(R(s_t, a_t, s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right) \bigg) \bigg| s_0 = s \right]$$

And, Q^{π} is changed to include the entropy bonuses from every time-step *except the first*

$$Q^{\pi}(s,a) = \underset{\tau \sim \pi}{\mathrm{E}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) + \alpha \sum_{t=1}^{\infty} \gamma^t H\left(\pi(\cdot|s_t)\right) \bigg| s_0 = s, a_0 = a \right]$$

So,

$$V^{\pi}(s) = \underset{a \sim \pi}{\mathrm{E}} \left[Q^{\pi}(s,a) \right] + \alpha H\left(\pi(\cdot|s)\right)$$

And, the Bellman Equation becomes

$$\begin{aligned} Q^{\pi}(s,a) &= \underset{\substack{s' \sim P \\ a' \sim \pi}}{\mathrm{E}} \left[R(s,a,s') + \gamma \left(Q^{\pi}(s',a') + \alpha H\left(\pi(\cdot|s')\right) \right) \right] \\ &= \underset{s' \sim P}{\mathrm{E}} \left[R(s,a,s') + \gamma V^{\pi}(s') \right]. \end{aligned}$$

Experience Replay

It is a replay memory technique used in reinforcement learning where we store the agent’s experiences at each time-step, in a data-set, pooled over many episodes into a replay memory. We then usually sample the memory randomly for a mini-batch of experience, and use this to learn off-policy. Experience Data has Temporal Dependency.

Environmental Agent and Classifier

Environmental Agent is added in Original RL Framework to address dataset imbalance issue by doing dynamic and intelligent resampling of dataset during training.

Initially, environment agent is given random training data. As training proceeds, the environment agent learns and samples the training data that maximises its reward.

Classifier acts as actor and environmental agent as critic which is rewarded when classifier fails to recognise otherwise classifier is rewarded

The environment agent and classifier agent perform adversarial learning around obtaining the maximum reward. The environment agent gets the maximum reward by sampling more data that the classifier cannot recognise or classifies incorrectly. Conversely, classifiers receive a larger reward by recognising or correctly classifying more
data.