$$\mathcal{L}_\pi = Loss\ function\ used\ to\ train\ the\ policy\ (actor)\ network\ \pi_\theta$$

$$\mathcal{L}_\pi = \mathbb{E}_{\tilde{a}\sim\pi_\theta}[\alpha \log \pi_\theta(\tilde{a} \mid o) - Q_\theta(o, \tilde{a})]$$

"α" *is a coefficient controls the amount of randomness (exploration) in actor's behaviours*

- $\alpha \log \pi_\theta(\tilde{a}|o)$ *is called Entropy term, it is usually negative which encouraging exploration*!
- $-Q_\theta(o, \tilde{a})$ *with a negative sign indicates that encourages high* quality $(Q - value)actions$!

The value of a state *o under policy* $\pi =$:

$$V^\pi(o) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \,|o_0 = o\right]$$

"*The value of a state o under policy* π *is defined as* :

the total expected (weighted) rewards starting from state o and following policy π *forever*."

$\gamma \in [0,1]$ *Discount factor − controls how much future rewards are worth.*

It models how much less future rewards are worth compared to now,

and helps how far ahead the agent should look.

$$\sum_{t=0}^{\infty} \gamma^t r_t = r_0 + \gamma^1 r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \cdots$$

which ***shrinks*** *the weights of the future rewards.*

Recall: it is actually a convergent Geometric series with $0 < \gamma < 1$, *converges to* $\frac{r_t}{1-\gamma}$.

Bellman equation *is a recursive decomposition of* $V^{\pi}(o)$:

$$V^{\pi}(o) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} r_t \,|o_0 = o\right]$$

$$= \mathbb{E}_{\pi}[r_0 + \gamma^1 r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \cdots |o_0 = o]$$

$$= \mathbb{E}_{a\sim\pi, o', r}[r + \gamma \cdot V^{\pi}(o')]$$

Bellman equation for the Q function under policy π:

$$Q^{\pi}(o,a) = \mathbb{E}_{o',r}\left[r + \gamma \mathbb{E}_{a'\sim\pi(\cdot|o')}[Q^{\pi}(o',a')]\right]$$

Bellman residue:

$$\delta(o,a) = Q_{\phi}(o,a) - \left(r + \gamma \mathbb{E}_{a'\sim\pi(\cdot|o')}\left[Q_{\phi}(o',a')\right]\right)$$

$$\mathbb{E}_{a'\sim\pi(\,\cdot\,|o')}:$$

"*Take the expected value over action* a' *sampled from policy* π, *conditioned on being in state* o'."

$$a'\sim\pi(\,\cdot\,|o'):$$

"*Sample one action* a' *out a distribution* π *of the possible actions* $a_1, a_2, a_3, ..,$

given that the agent is in state o'."

$$\mathcal{L}_Q = \left(Q_{\phi}(o,a) - \left(r + \gamma \mathbb{E}_{a'\sim\pi(\,\cdot\,|o')}\left[Q_{\phi}(o',a')\right]\right)\right)^2$$

$Running\ statistics\ normalization$:

$$\bar{o}_t = RSNorm(o_t) = \frac{o_t - \mu_t}{\sqrt{\sigma^2 + \epsilon}}$$

$Note$: $In\ the\ late\ stage\ of\ trainging, as\ policy\ coverges, i.e. deterministic, the\ variance\ \sigma^2$

$would\ become\ very\ small\ (close\ to\ zero),$

$so + \epsilon\ is\ important\ for\ numerical\ safety\ and\ regularizatiion\ purpose$

$(prevent\ value\ from\ blowing\ up).$

$Running\ mean:\ \mu_t = \mu_{t-1} + \frac{(o_t - \mu_{t-1})}{t} = \mu_{t-1} + \frac{\delta_t}{t}$

$\mu_t \in \mathbb{R}^{|\mathcal{O}|}\ where\ |\mathcal{O}|\ is\ the\ dimensionality\ of\ the\ observation\ space.$

$\mu_t\ is\ a\ vetor\ of\ length\ equals\ to\ the\ dimensionality\ of\ the\ observation\ space\ |\mathcal{O}|.$

$$h_t^0 = W_h^0 \bar{o}_t$$

$h_t^0\ is\ the\ latent\ feature\ vector$

$(i.e., the\ hidden\ representation\ of\ the\ normalized\ observation\ \bar{o}_t)\ at\ layer\ 0.$

$Conclusion$: $A\ combination\ of\ normalization\ and\ linear\ projection,$

$which\ together\ transform\ raw, high - dimensional\ observations$

$into\ learnable, low -\ or\ fixed$

$- dimensional\ feature\ vectors\ suitable\ for\ downstream\ processing.$

MLP + ℓ_2-Norm

$$\widetilde{\boldsymbol{h}}_t^l = \ell_2\text{-Norm}\Big(\boldsymbol{W}_{h,2}^l \, ReLU\big((\boldsymbol{W}_{h,1}^l \boldsymbol{h}_t^l) \odot \boldsymbol{s}_h^l\big)\Big)$$

Where:

$$\boldsymbol{h}_t^l \in \mathbb{R}^{|\mathcal{O}|+1}$$

$$\boldsymbol{W}_{h,1}^l \in \mathbb{R}^{4d_h \times d_h}$$

$$\boldsymbol{W}_{h,2}^l \in \mathbb{R}^{d_h \times 4d_h}$$

$$\boldsymbol{s}_h^l \in \mathbb{R}^{4d_h}$$

LERP + ℓ_2-Norm

$$\boldsymbol{h}_t^{l+1} = \ell_2\text{-Norm}\Big((\mathbf{1} - \alpha^l) \odot \boldsymbol{h}_t^l + \alpha^l \odot \widetilde{\boldsymbol{h}}_t^l\Big)$$

Where:

$$\mathbf{1} \in \mathbb{R}^{d_h}$$

learnable interpolation vector

$$\alpha^l \in \mathbb{R}^{d_h}$$

$$p_{t,\boldsymbol{a}} = \text{softmax}\left(z_{t,a}\right)$$

Distributional Critic

$$\left\{\delta_i = G_{min} + \frac{(i-1)(G_{max} - G_{min})}{n_{atom} - 1} \mid i = 1, \dots, n_{atom}\right\}$$

Where:

$$G_{min} = the\ minimum\ possible\ returns$$

$$G_{min} = the\ maximum\ possible\ returns$$

$$n_{atom} = the\ number\ of\ discrete\ atoms$$

$\delta_i =$ a possible value that the total future reward could be

$$\mathcal{Z}_t = W_{o,2}\left(\left(\boldsymbol{W}_{o,1}\, h_t^L\right) \odot \boldsymbol{s}_o\right)$$

Where:

$$\mathcal{Z}_t \in \mathbb{R}^{|\mathcal{A}| \times n_{atom}}$$

$$\boldsymbol{W}_{o,1} \in \mathbb{R}^{d_h \times d_h}$$

$$\boldsymbol{W}_{o,2} \in \mathbb{R}^{|\mathcal{A}| \times n_{atom} \times d_h}$$

$$\boldsymbol{s}_o \in \mathbb{R}^{d_h}$$

$\boldsymbol{h}_t^L$: The encoded representation of observation o_t at layer L

Or the final latent/feature vector, produced by the last layer of encoder.

Q-value (the expected return under $p_{t,\boldsymbol{a}}$)

$$\mathrm{Q}(\boldsymbol{o_t}, \boldsymbol{a}) = \sum_{i=1}^{n_{atom}} \delta_i p_{t,\boldsymbol{a},i}\,.$$

Two kinds of randomness in Reinforcement Learning

Actions have randomness.

- Given state s, the action can be random, denoted as $A \sim \pi(\cdot \mid s)$.
  E.g., $\pi(\text{ "left" } \mid s) = 0.2,\ \pi(\text{ "right" } \mid s) = 0.1, \pi(\text{ "up" } \mid s) = 0.7.$

State transitions have randomness.

- Given state $S = s$ and action $A = a$, the environment randomly generates a new state S'.

Discounted return (aka cumulative discounted future reward).

$$U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+1} + \gamma^3 R_{t+1}$$

Action-value function for policy π:

$$Q_\pi(s_t, a_t) = \mathbb{E}[U_t \mid S_t = s_t, A_t = a_t]$$

For policy π, $Q_\pi(s_t, a_t)$ evaluates how good it is for an agent to pick action a while being in state s.

Note: $Q_\pi(s_t, a_t) = Q_\pi(s, a)$ when the "environment" does not change over time.

To fully understand the Action-value function for policy π, we consider $Q_\pi(s_t, a_t)$ to measure the expected total reward if we take action a in state s now and follow policy π from the next step onward.

So

$Q_\pi(s_t, a_t) = \mathbb{E}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+1} + \gamma^3 R_{t+1} | S_t = s_t, A_t = a_t] = r(s, a) + future\ rewards$

$By\ Bellman\ Expectation\ Equation$:

$Q_\pi(s_t, a_t)$

$= \mathbb{E}_{s_{t+1} \sim P(\cdot|s_t, a_t)} \left[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi(\cdot|s_{t+1})} [Q_\pi(s_{t+1}, a_{t+1})] \right]$

$= r(s_t, a_t) + \gamma \cdot \mathbb{E}_{s_{t+1}, a_{t+1}} [Q_\pi(s_{t+1}, a_{t+1})]$

$\mathbb{E}_{s_{t+1}\sim P(\cdot|s_t,a_t)}\,[\,r(s_t,a_t)] = r(s_t,a_t)$?

Yes, because when you are taking the expectation over s_{t+1}, the $r(s_t,a_t)$ can be treated as a scalar since it does not depend on s_{t+1}.

$Q_\pi(s,a)$ is the expected total discounted reward the agent will receive starting from state s, taking action a immediately, and following policy π afterward.

Optimal action-value function for policy π:

$$Q^*(s_t,a_t) = \mathbb{E}[U_t|S_t = s_t, A_t = a_t].$$

State-value function:

$$V_\pi(s) = \mathbb{E}_A[Q_\pi(s,A)] = \sum_a \pi(a|s_t)\cdot Q_\pi(s_t,a). \qquad \text{(Actions are discrete.)}$$

$$V_\pi(s) = \mathbb{E}_A[Q_\pi(s,A)] = \int \pi(a|s_t)\cdot Q_\pi(s_t,a)\,da. \quad \text{(Actions are continuous.)}$$

- For fixed policy π, $V_\pi(s)$ evaluates how good the situation is in state s.
- $\mathbb{E}_s[V_\pi(S)]$ evaluate how good the policy π is.

Note that the key use of the state-value function is to compare the effectiveness of different policies, by measuring how much expected reward each policy can achieve starting from a given state.

In value-based RL, the state-value functions are required; While in the policy-based RL, the state-value functions are **optional**.

The Q-function $Q_\theta(o, \bar{a})$ is trained to reduce the Bellman residue loss:

Value-based reinforcement learning

The goal of value-based RL is to learn **the optimal action-value** function that tells the agent how good it is to take a certain action in a certain state, to **maximize** the cumulative rewards over time.

The value function can be more specific like an Action-value function under policy π

$$Q_\pi(s_t, a_t)$$

which evaluates **the state-action pairs**.

The input of the action-value function $Q_\pi(s_t, a_t)$ is a state s_t given and an action a_t, so totally two inputs.

s_t is the current state at time step t, and a_t is the current action at time step t. Note that in each episode, there is exactly one sate at each time step.

Keep in mind that $Q_\pi(s_t, a_t) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t, a_t]$. since all future rewards are incorporated through the expectation, so theoretically, the quality of action taken currently in the current state theoretically depends on all possible future states and future actions that may follow. However, we do not need to wait until the end of the episode to evaluate this action — **the expected value summarizes that information in advance**.

Policy π, under which the action-value function is defined, while is not considered as an input of $Q_\pi(s_t, a_t)$. "π" defines which version of the function you're using. Whenever we use the action-value function to evaluate state-action pairs under a fixed policy π, the policy π is fixed prior to evaluation. (i.e. π is not a runtime input.)

$$Q^*(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t)$$

This $Q^*(s_t, a_t)$ indicates the most (optimal) cumulative rewards the agent can possibly get by taking action a_t in state s_t at time step t and then following the optimal policy thereafter.

we do not know this $Q^*(s_t, a_t)$ in advance, the **goal of value-based RL is to learn or approximate $\boldsymbol{Q^*(s_t, a_t)}$**.

The policy-based learning

To learn θ maximizes the

$$J(\theta) = \mathbb{E}_{s\sim d^\pi}[V^\pi(s;\theta)]$$

Since

$$V^\pi[s;\theta] = \sum_a \pi(a|s;\theta)\cdot Q^\pi(s,a)$$

So

$$J(\theta) = \mathbb{E}_{s\sim d^\pi}[V^\pi(s;\theta)] = \mathbb{E}_{s\sim d^\pi}\left[\sum_a \pi(a|s;\theta)\cdot Q^\pi(s,a)\right]$$

$$J(\theta) = \mathbb{E}_{s\sim d^\pi}[V^\pi(S;\theta)] = \mathbb{E}_{s\sim d^\pi}\left[\int \pi(a \mid s;\theta)\cdot Q^\pi(s,a)\,da\right]$$

Where d^π stands for the state distribution under policy π.

$J(\theta)$ is called the "objective function" in policy-based reinforcement learning.

Here, by convention, mathematicians use **lower-case Greek letter to represent parameters**, and **lower-case letter to represent variables**; variables are inputs the function depends on dynamically, parameters are fixed or tunable quantities.

So $\boldsymbol{J(\theta)}$ means we could optimize the output of function $\boldsymbol{J}$ by tuning the parameter $\boldsymbol{\theta}$,(usually a vector); $\boldsymbol{J(x)}$ gives us an evaluation when the input is variable $\boldsymbol{x}$ given.

$$J(\theta) = \mathbb{E}_{s\sim d^{\pi}}[V^{\pi}(s;\theta)] = \mathbb{E}_{s\sim d^{\pi}}\left[\sum_{a} \pi(a \mid s;\theta)\cdot Q^{\pi}(s,a)\right]$$

Remember, the goal of Policy-Based RL is to find the policy π such that maximized the expected return $J(\theta)$(i.e. the best parameterized stochastic policy $\pi(a \mid s;|\theta)$).

Definition of Stochastic policy:

Choose actions randomly, based on learned probabilities. In contrast to Deterministic Policy, always picks the single best action based on current knowledge.

Furthermore, no action in Stochastic policy is guaranteed (probability = 1) given in state s, but it might be 0.9999... and works like a Deterministic policy.

The goal of Value-Based RL is to minimize the difference between the current Q-value estimate and the target value given by the Bellman equation.

TD target (Value-Based Learning):

$$y_t = r_t + \gamma \cdot Q^{\pi}(s_{t+1}, a_{t+1};\theta)$$
$$= r_t + \gamma \max_{a'} Q^{\pi}(s_{t+1}, a';\theta)$$
$$(\theta = \boldsymbol{W}_t)$$

$\boldsymbol{W}_t$ has an emphasis on time step.

TD difference (also called TD error) (Value-Based Learning):

$\delta_t = r_t + \gamma \cdot V(s_{t+1};\theta^-) - V(s_t;\theta)$ (State-Value-based TD learning)

Or $\delta_t = r_t + \gamma \cdot Q(s_{t+1}, a';\theta^-) - Q(s_t, a_t;\theta)$ (Action-value-based TD learning)

Recall $V(s_t) = \sum_a \pi(a \mid s_t)\cdot Q(s_t, a;\theta)$

$V(s_{t+1}) = \sum_{a'} \pi(a' \mid s_{t+1})\cdot Q(s_{t+1}, a';\theta^-)$

We use **a specific action $\boldsymbol{a}$'** instead of the Expectation in the Action-based TD learning, a' is either the greedy action $a' = \arg\max_{a} Q(s_{t+1}, a;\theta^-)$ (Q-learning), or the action taken at t+1(SARSA). The reason to do this is to allow practical learning without computing expensive expectations, since over many updates, **averages converge to true expected values**.

Note: the difference between focusing on state-value function $V(s)$ versus action-value functions $Q(s,a)$ is fundamental in RL. It shapes which algorithm you used, how you represent knowledge, and how you optimize policies.

Look at the θ^- in $V(s_{t+1};\theta^-)$, it is called a delayed copy of θ. The purpose of using a delayed copy θ^- is to stabilize learning by keeping TD targets from shifting too quickly, i.e. we update θ^- when θ has had time to meaningfully improve.

$$\boldsymbol{\mathcal{L}}(\theta) = [\delta_t]^2 = \begin{cases} (r_t + \gamma \cdot V(s_{t+1};\theta^-) - V(s_t;\theta))^2 & \text{(State-value-based TD learning)} \\ \left(r_t + \gamma \cdot Q(s_{t+1},a';\theta^-) - Q(s_t,a_t;\theta)\right)^2 & \text{(Action-value-based TD learning)} \end{cases}$$

As time t increases during an episode or across episodes in reinforcement learning, you accumulate more observations of s_t , a_t , r_t , s_{t+1}. Each new time step gives you one more sample to compute δ_t update your value function (state-value or action-value function depends on Algorithm designs.) based on that error δ_t.

To analyse the behaviour of TD error δ_t.

First notice that the goal in the value-based TD learning is to minimize the loss function $\boldsymbol{\mathcal{L}}(\theta)$, the minimization implies that we need to apply gradient descent on $\boldsymbol{\mathcal{L}}(\theta)$ as follow:

$$\theta \longleftarrow \theta - \alpha \cdot \nabla_\theta \, \boldsymbol{\mathcal{L}}(\theta)$$

$$\theta \longleftarrow \theta - \alpha \cdot \nabla_\theta \, [\delta_t]^2$$

$$\theta \longleftarrow \theta - \alpha \cdot 2 \cdot \delta_t \cdot \nabla_\theta Q(s_t,a_t;\theta) \quad \text{Chain rule!}$$

$$\theta \longleftarrow \theta - \alpha \cdot 2 \cdot \left(r_t + \gamma \cdot Q(s_{t+1},a_{t+1};\theta^-) - Q(s_t,a_t;\theta)\right) \cdot \nabla_\theta Q(s_t,a_t;\theta)$$

The goal of Value-Based Learning is to find the best value function (i.e. $Q^\pi(s,a;\theta)$), the action-value function. The policy itself is derived **implicitly** by choosing actions that maximize Q.

$$\pi(s) = arg\max_a Q^\pi(s,a;\theta)$$

Theoretically, the policy which is derived implicitly by choosing actions that maximize Q in Value-Based Learning should converge to the same optimal policy π^* we get from the Policy-Based Learning. **But in practice, we usually learn different policies**.

$$J(\theta) = \mathbb{E}_{s\sim d^{\pi}}[V^{\pi}(S;\theta)] = \mathbb{E}_{s\sim d^{\pi}}\left[\int \pi(a \mid s;\theta)\cdot Q^{\pi}(s,a)\,da\right]$$

For continuous variables like action a shown in the formula above, we do the integration calculation, but we usually do the integration by using Monte Carlo Estimation, which convert the integration back to summation of a finite but large number (N) of discrete points. See following for details of Monte Carlo estimations:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{N}\sum_{i=1}^{N} f(x_i) \quad \text{where } x_i \sim \text{Uniform}(a,b)$$

We don't need to explicitly know the distribution of x, we have the following basic information about that distribution:

Sample mean: $\mu = \frac{1}{N}\sum_{i=1}^{N} f(x_i)$ (Monte Carlo estimate)

Variance of the Sample mean (μ):

$$Var(\mu) = Var\left(\frac{1}{N}\sum_{i=1}^{N} f(x_i)\right)$$

$$= \frac{1}{N^2}\cdot Var\left(\sum_{i=1}^{N} f(x_i)\right)$$

$$= \frac{1}{N^2}\cdot N \cdot Var\big(f(x_i)\big)$$

$$= \frac{1}{N^2}\cdot N \cdot \sigma^2$$

$$= \frac{\sigma^2}{N} \quad \text{Where } \sigma \text{ is } Var\big(f(x_i)\big).$$

$$\text{Standard Error}(\mu) = \sqrt{\frac{\sigma^2}{N}} = \frac{\sigma}{\sqrt{N}}.$$

The standard error suggests that if one wants to reduce the error by a factor of 10

(i.e. New error $= \frac{1}{10} \times$ Original error),

you need to increase the number of samples by a factor of $10^2 = 100$

(i.e. 100 "**times**" more samples).

Actor-Critic Methods

Actor-Critic Method is a **combination** of the Policy-Based method (actor) and the Value-Based method (critic).

In Actor-Critic methods one needs to train **two** neural networks:

one for the policy function $\pi(a \mid s)$ and another for the action-value function $Q(s, a)$.

Use neural network $\pi(a \mid s; \boldsymbol{\theta})$ to approximate $\pi(a \mid s)$;

Use neural network $q(s, a; \boldsymbol{W})$ to approximate $Q^{\pi}(s, a)$.

Note: the "q" in "$q(s, a; \boldsymbol{W})$" implies its estimator nature of the true action-value function $Q^{\pi}(s, a)$.

Now recall the State-Value function:

$$V^{\pi}(s) = \left(\int_{a} or\right) \sum_{a} \pi(a \mid s) \cdot Q^{\pi}(s, a) \ (\text{or } da) \approx \left(\int_{a} or\right) \sum_{a \in \mathcal{A}} \pi(a \mid s; \boldsymbol{\theta}) \cdot q(s, a; \boldsymbol{W}) \ (\text{or } da)$$

> Recall that the policy function's estimator $\pi(a|s; \boldsymbol{\theta})$ is a PDF(Probability Density Function) or probability mass function (PMF) in the discrete case, and so it is defined w.r.t a given condition (i.e., its input), and that condition or conditions serves as its input or inputs when the PDF or PMF is conditional.

By the information given above, we know that the input of **policy network (actor)**:

$$\pi(a \mid s; \boldsymbol{\theta})$$

takes state s as its **only** input (e.g., a **screenshot** of Super Mario gameplay.)

$$\sum_{a \in \mathcal{A}} \pi(a \mid s; \boldsymbol{\theta}) = 1 \text{ (we use softmax activation to assure this).}$$

For value-network (critic), it takes two inputs:

- state s
- action a

Recall the value-network $q(s, a; \boldsymbol{W})$ is an estimator of the action-value function $Q^{\pi}(s, a)$:

$$q(s, a, \boldsymbol{W}) \approx Q^{\pi}(s, a)$$

The state s (the screenshot of the game play) goes through convolutional layers and output is feature vectors.

Convolutional layers are designed to extract **spatial features** from structure data like images or videos. For videos, we treat **each frame** as a still image and process them one by one using convolutional layers. There is basically *no difference* between videos and images when it comes to *spatial* feature extraction.

Filters (Kernels)

Each filter (also called a **kernel**) in a convolutional layer produces one **feature map**, each feature map is a 2D grid of numbers (real values) that represent the response (activation) of a convolutional filter applied over the input. A convolutional layer usually contains more than one filters (kernels), so it outputs more than one feature maps (A set of feature maps).

Activation of Kernel's output

Feature maps

The 2D grid of feature map is designed to preserve the spatial information from the input image or previous layer, **BUT** the dimensions of feature map are not necessarily the same as the input image resolution. (In practice, the dimensions of feature map (i.e. the number of grids) are usually smaller than the input resolution.)

In RL, a state

Experience Replay Technique

An advance data sampling strategy — how to use the collected data (i.e., transitions) more efficiently.

A transition is defined as a 4 elements tuple:

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

Store recent n transitions in a **replay buffer** (hyper-parameter $n = 10^5 \sim 10^6$ in practice).

Two major benefits of using a replay buffer (i.e., Experience Replay method):

- Break correlation between consecutive experiences
  (Consecutive frames share similarities with minor difference)
- Improve data efficiency
  (reusing previous experiences multiple times)

These two benefits help with improving the learning stability.

By randomly and uniformly sampling the transitions from the replay buffer, we not only break the correlation between consecutive experiences, but also simulate the variability of real-world test conditions — when the agent is "playing" the game, and the next state s_{t+1} is inherently uncertain (or stochastic).

Remove old transitions so that the buffer has at most n transitions.

Dueling-network technique

Reference:

1. Wang et al. Dueling network architectures for deep reinforcement learning. In *ICML*, 2016.

We use Dueling Network Architecture to improves the stability and efficiency of value-based RL methods like DQN.

Comparison:

Traditional DQN — approximating the $Q^*(s,a)$ **directly** by a neural network $Q(s,a;\boldsymbol{W})$.

Dueling-network — approximating the $Q^*(s,a)$ indirectly by two neural networks:

the $A^*(s,a)$ by a neural network $A(s,a;\boldsymbol{W}^A)$;

In addition, we also need a neural network $V(s;\boldsymbol{W}^V)$ to approximate $V^*(s)$.

Note:

- $A(s,a;\boldsymbol{W}^A)$ and $Q(s,a;\boldsymbol{W})$ share the same architecture.
- The output of $A(s,a,\boldsymbol{W}^A)$ is **a vector** with dimension equal to $|\mathcal{A}|$.
- The output of $V(s;\boldsymbol{W}^V)$ is **a real number** (i.e., A critic on the current state s).
- $A(s,a;\boldsymbol{W}^A)$ and $V(s;\boldsymbol{W}^V)$ can share the parameters in the convolutional layers, that process the state s into a feature vector.

The formulation of Dueling-network:

$$Q(s,a;\boldsymbol{W}^V,\boldsymbol{W}^A) = V(s;\boldsymbol{W}^V) + A(s,a;\boldsymbol{W}^A) - \max_a A(s,a;\boldsymbol{W}^A)\,.$$

Note:

- The addition of $V(s;\boldsymbol{W}^V)$ and $A(s,a;\boldsymbol{W}^A)$ is carried out in an element-wise manner.
- The subtraction is carried out in an element-wise manner as well.

We define the optimal advantage function as: $A^*(s,a) = Q^*(s,a) - V^*(s)$.

By theorem (State-value optimality):

$$V^*(s) = \max_a Q^*(s,a).$$

This equation tells us that knowing V^* is enough to extract the optimal policy.

$$V^*(s) = \max_a Q^*(s,a) = \max_a Q^{\pi^*}(s,a).$$

Definition of $Q^*(s_t, a_t)$:

$$Q^*(s_t, a_t) = Q^{\pi^*}(s_t, a_t) = \mathbb{E}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} | S_t = s_t, A_t = a_t, \pi = \pi^*]$$
$$= \mathbb{E}_{\pi^*}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} | S_t = s_t, A_t = a_t]$$

Which means the agent follow the optimal policy starting from state s_t, hence the optimal state-value function $V^*(s_t)$ at state s_t is the expectation, under the optimal policy π^*, of discounted return U_t, starting from state s_t onward.

$$V^*(s_t) = V^{\pi^*}(s_t) = \mathbb{E}_{\pi^*}[U_t | S_t = s_t] = \sum_a \pi^*(a \mid s_t) \cdot Q^{\pi^*}(s_t, a)$$
$$= \sum_a \pi^*(a \mid s_t) \cdot \left[\sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k} \mid S_t = s_t, A_t = a\right]$$

Since we have the optimal policy here, so

$\pi^*(a \mid s) = 1$ for the maximizing action a^*, and 0 elsewhere.

(i.e., the optimal policy $\pi^*$is deterministic by assumption)

We have $\sum_a \pi^*(a \mid s_t) = 1 + 0 + 0 \ldots = 1$, so

$$\sum_a \pi^*(a \mid s_t) \cdot \left[\sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k} \mid S_t = s_t, A_t = a\right] = \max_a (1) \cdot \left[\sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k} \mid S_t = s_t, A_t = a\right]$$
$$= \max_a Q^{\pi^*}(s_t, a)$$

Note: when we make the assumption of optimal functions $V^*(s)$ and $Q^*(s, a)$, the assumption includes the following two important parts:

- know the optimal policy π^*;
  (i.e., For every step agent know what the best action a^* is to be taken)
- the optimal policy π^* is deterministic and not stochastic.
  (i.e., the agent never makes mistakes, it will ONLY take action a^* and never fail to do so.)

So now we have two equations:

Definition of the Optimal Advantage Function

$$A^*(s,a) = Q^*(s,a) - V^*(s).$$

Now take the maximum on the definition of $A^*(s,a)$:

$$\max_a A^*(s,a) = \max_a [Q^*(s,a) - V^*(s)]$$

$$\max_a A^*(s,a) = \max_a Q^*(s,a) - V^*(s)$$

Note:

- $V^*(s)$ is independent of a.
- $\max_a Q^*(s,a) = V^*(s)$.

$$\max_a A^*(s,a) = 0.$$

Implementation of Dueling-network

Theoretically:

$$Q^*(s,a) = V^*(s) + A^*(s,a) - \max_{a'} A^*(s,a') \text{ (Theoretical Baseline)}$$

Practically:

$$Q(s,a;\boldsymbol{W}) = V(s;\boldsymbol{W}^V) + A(s,a;\boldsymbol{W}^A) - \frac{1}{|\boldsymbol{\mathcal{A}}|}\sum_a A(s,a;\boldsymbol{W}^A) \text{ (Practical Baseline)}$$

Or equivalently

$$Q(s,a;\boldsymbol{W}) = V(s,\boldsymbol{W}^V) + A(s,a;\boldsymbol{W}^A) - \operatorname{mean}_a A(s,a;\boldsymbol{W}^A)\,.$$

Note: The practical implementation applies $\operatorname{mean}_a A(s,a;\boldsymbol{W}^A)$ rather than $\max_a A(s,a;\boldsymbol{W}^A)$, as the baseline based on empirical effectiveness rather than theoretical justification.

Discrete Action Space versus **Continuous Action Space**

The action space is a continuous set.

We have infinite many actions in a continuous action set.

The first approach is **Discretization**:

> We discretize the continuous action space into the discrete Action Space which has finite many actions. (It might be many but it is finite.)

Problem of Discretization: when the number of degrees of freedom d becomes larger, the number of the discrete action points grows exponentially as the d increases.

For a d-dimentional action space, the total number of possible discrete action is:

$$k^d.$$

Deterministic Policy Gradient (DPG)

DPG methods are specifically designed to handle continuous action spaces, where traditional discrete action methods like Q-learning are not directly applicable.

DPG is also referred to as a Deterministic Actor-Critic method.

Features:

- The Policy function $\pi(a \mid s; \boldsymbol{\theta})$ outputs a real-valued vector of dimension d, where $d = |\boldsymbol{\mathcal{A}}|$, instead of probability distribution over $|\boldsymbol{\mathcal{A}}|$ actions.
- The continuous action space $\boldsymbol{\mathcal{A}} \subseteq \mathbb{R}^d$ (d is the degrees of freedom), while discrete action space $\boldsymbol{\mathcal{A}}$ has cardinality $|\boldsymbol{\mathcal{A}}|$,which is called the number of distinct actions. (e.g. $\boldsymbol{\mathcal{A}} = \{\text{left, right, jump, fire}\}$)
- Important concept: the output of the policy network follows the dimension of the action space $\boldsymbol{\mathcal{A}}$, so the output of the deterministic policy network is $\subseteq \mathbb{R}^d$, where d is the number of degrees of freedom in the action space.

Important insight:

In continuous action spaces, the action is typically <u>deterministic</u> across all d dimensions.

In contrast, a standard discrete action space is often <u>stochastic</u> but only over a single categorical dimension representing a finite set of atomic actions.

we can also have a discrete actions space in d dimensions, **BUT** that won't be used to solve the continuous control problems (e.g. controlling robot arms), it would only be used in solving the discrete actions space problem with degrees of freedom more than one.

First, we need to upgrade the Value-Network (i.e. the approximation of the Q-function).

To do that we use the TD (Temporal Difference) method:

Transition (s_t, a_t, r_t, s_{t+1})

$Prediction_t$ $\quad q_t = q(s_t, a_t; \mathbf{w})$

Pseudo $\quad a'_{t+1} = \pi(s_{t+1}; \boldsymbol{\theta})$

$Prediction_{t+1}$ $\quad q_{t+1} = q(s_{t+1}, a'_{t+1}; \mathbf{w})$

TD error $\quad \delta_t = q_t - (r_t + \gamma \cdot q_{t+1})$

Loss $\quad \boldsymbol{\mathcal{L}} = \frac{1}{2} \cdot \delta_t^2$

Update $\quad \mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \delta_t \frac{\partial q(s_t, a_t; \mathbf{w})}{\partial \mathbf{w}}$ $\left(\text{from } \mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \frac{\partial \boldsymbol{\mathcal{L}}}{\partial \mathbf{w}}\right)$

Or $\quad \mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \delta_t \nabla_w Q(s_t, a_t; \mathbf{w})$

Upgrade Policy Network by DPG (Deterministic Policy Gradient)

$$\boldsymbol{g} = \frac{\partial q(s, \pi(s; \boldsymbol{\theta}); \boldsymbol{w})}{\partial \boldsymbol{\theta}} = \frac{\partial a}{\partial \boldsymbol{\theta}} \cdot \frac{\partial q(s, a; \boldsymbol{w})}{\partial a}.$$

And then use the gradient ascent to update the $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \beta \cdot \boldsymbol{g}.$$

We upgrade the policy parameters $\boldsymbol{\theta}$ to ensure that the Q-function returns a higher value for the action selected by the policy in a given state s.

The *bootstrapping* issue in the TD method:

We use Q-function's estimate to update Q-function itself.

To see where this happens, recall the TD error :

$$\text{TD error} = q_t - (r_t + \gamma \cdot q_{t+1}) = q(s_t, a_t; \boldsymbol{w}) - \left(r_t + \gamma \cdot q(s_{t+1}, \pi(s_{t+1}; \boldsymbol{\theta}); \boldsymbol{w})\right)$$

The TD target $\left(r_t + \gamma \cdot q(s_{t+1}, \pi(s_{t+1}; \boldsymbol{\theta}); \boldsymbol{w})\right)$

contains $q(\cdot; \boldsymbol{w})$ again, the Q-network's own prediction guides its update.

To solve this issue, we use a **target network** to estimate the TD target.

the target network consists of two components:

- $\pi(s_{t+1}; \boldsymbol{\theta}^-) \longmapsto a'_{t+1}$
  (the target actor network outputs the next pseudo action)
- $q(s_{t+1}, a'_{t+1}; \boldsymbol{w}^-)$
  (the target critic\value network estimate the Q-value of that action)

Note: The target actor network and the target critic network have parameters $\boldsymbol{\theta}^-, \boldsymbol{w}^-$, which are distinct from the main training parameters $\boldsymbol{\theta}$ and $\boldsymbol{w}$ in the <u>online</u> actor and critic network.

One could see the target network **as a mirror reflection of the online network**; the **only** difference between these two networks lies in their parameters.

Transition:(s_t, a_t, r_t, s_{t+1})

To get q_{t+1}, now we use the target network instead:

$$\delta_t = q_t - (r_t + \gamma \cdot q_{t+1}) = q(s_t, a_t; \boldsymbol{w}) - \big(r_t + \gamma \cdot q(s_{t+1}, \pi(s_{t+1}; \boldsymbol{\theta}^-); \boldsymbol{w}^-)\big).$$

$$q_{t+1} = q_{target}(s_{t+1}, a'_{t+1}; \boldsymbol{w}^-) = q_{target}\big(s_{t+1}, \pi_{target}(s_{t+1}; \boldsymbol{\theta}^-); \boldsymbol{w}^-\big).$$

You can also write it as:

$$\delta_t = q_t - (r_t + \gamma \cdot q_{t+1}) = q(s_t, a_t; \boldsymbol{w}) - \Big(r_t + \gamma \cdot q_{target}\big(s_{t+1}, \pi_{target}(s_{t+1}; \boldsymbol{\theta}^-); \boldsymbol{w}^-\big)\Big).$$

Recall that the Loss function $\boldsymbol{\mathcal{L}} = \frac{1}{2}\delta_t^2$.

To upgrade $\boldsymbol{\theta}$ by DPG (Deterministic Policy Gradient) ascent:

$$\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} + \beta \cdot \frac{\partial q(s_t, a_t; \boldsymbol{w})}{\partial a_t}\bigg|_{a_t = \pi(s_t; \boldsymbol{\theta})} \cdot \frac{\partial \pi(s_t; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

To upgrade the **w** by gradient descent on TD:

$$\boldsymbol{w} \longleftarrow \boldsymbol{w} - \alpha \cdot \frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{w}} = \boldsymbol{w} - \alpha \cdot \delta_t \cdot \nabla_{\boldsymbol{w}} q(s_t, a_t; \boldsymbol{w}) = \boldsymbol{w} - \alpha \cdot \delta_t \cdot \frac{\partial q(s_t, a_t; \boldsymbol{w})}{\partial \boldsymbol{w}}.$$

We use the Gradient Descend here to make sure the value of Loss function $\boldsymbol{\mathcal{L}}$ is as small as possible.

There is a bridge between the online network parameters $\boldsymbol{w}$ and the target network parameters $\boldsymbol{w}^-$, determined by the hyperparameter $\boldsymbol{\tau} \in (0,1)$. We use $\boldsymbol{\tau}$ to update $\boldsymbol{w}^-$ in the target network via weighted averaging, also known as a soft update:

$$\boldsymbol{w}^- \leftarrow \tau \cdot \boldsymbol{w} + (1-\tau) \cdot \boldsymbol{w}^-;$$

$$\boldsymbol{\theta}^- \leftarrow \tau \cdot \boldsymbol{\theta} + (1-\tau) \cdot \boldsymbol{\theta}^-.$$

Notice that the target networks parameters still depend indirectly on the online parameters; the bootstrapping remains, but with much less influence on the online network's training, how much less effect depends on the hyperparameter $\boldsymbol{\tau}$.

In conclusion, the use of Target Network helps to partially reduce the instability caused by bootstrapping, but it does not eliminate it entirely.

Recall that we can also apply other techniques like Experience Replay and Multi-step TD targets to further improve the performance and the stability of online network.

For **Multi-step TD Targets**: