Reinforcement Learning

- Episodic vs Continuing tasks ⟶

  * Episodic ⟶ Interaction ends after some time step T.

    $S_0, A_0, \ldots, R_T, S_T$ will define every point.

  * Continuous ⟶ Interaction continues without limit.

    $S_0, A_0, \ldots, R_T, S_T, \ldots$ Never ending task.

- Rewards ⟶

  * Agent should maximize reward over time, not immediate.

    $G_t = R_{t+1} + \ldots + R_{t+1000} + \ldots$ ⟵ Return.

  * Discounted return ⟶

    - Modeling rewards over time, the more immediate are more likely to happen.

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+1}$$

- Markov Decision Process ⟶ MDPs

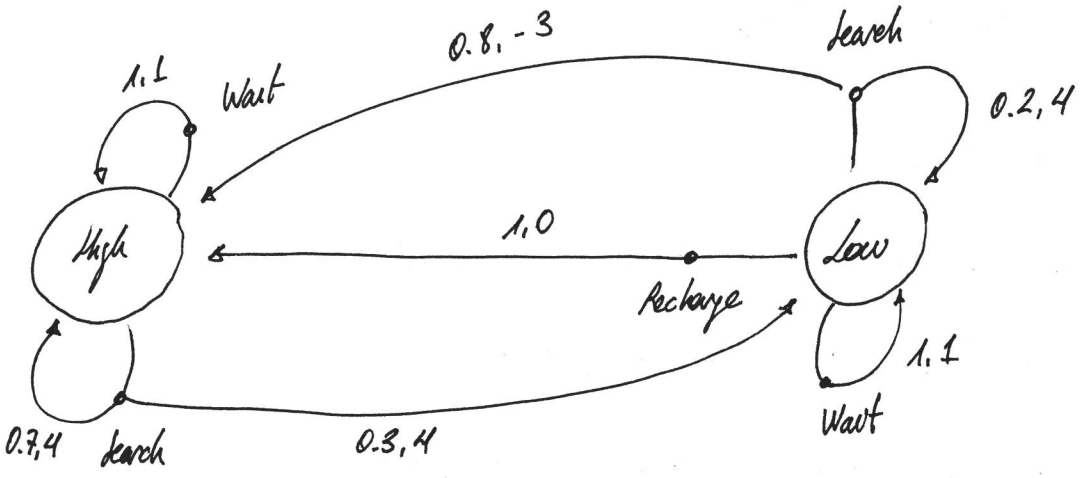
  $S \equiv$ All non-terminal states

  $S^+ \equiv$ All states, including terminal (Episodic task).

  $A \equiv$ Action space of all possible actions.

  $A(s) \equiv$ Set of actions available in state $s \in S$

$$A \equiv \begin{bmatrix} \text{Search} \\ \text{Recharge} \\ \text{Wait} \end{bmatrix} \qquad S \equiv \begin{bmatrix} \text{High} \\ \text{Low} \end{bmatrix}$$



* Environment doesn't remember any previous steps, actions, states except ⟶

$$p(s', r \mid s, a) \doteq \mathbb{P}(S_{t+1} = s',\ R_{t+1} = r \mid S_t = s,\ A_t = a)$$

↳ One-step dynamics

Function $p : S \times R \times S \times A \longrightarrow [0,1]$ ordinary deterministic function.

$$\sum_{s' \in S} \sum_{r \in R} p(s', r \mid s, a) = 1, \quad \text{for all } s \in S,\ a \in A(s).$$

* State transition probability ⟶

$$p(s' \mid s, a) = \mathbb{P}(S' = S_{t+1} \mid S_t = s,\ A_t = a) = \sum_{r \in R} p(s', r \mid s, a)$$

* Expected reward for state-action pair ⟶

$$r : S \times A \longrightarrow \mathbb{R}$$

$$r(s,a) = \mathbb{E}\left[R_t \mid S_{t-1} = s,\ A_{t-1} = a\right] = \sum_{r \in R} r \underbrace{\sum_{s' \in S} p(s', r \mid s, a)}_{p(r \mid s, a)}$$

* Expected reward for state-action-next-state triple ⟶

$$r : S \times A \times S \longrightarrow \mathbb{R}$$

$$r(s,a,s') = \mathbb{E}\left[R_t \mid S_{t-1} = s,\ A_{t-1} = a,\ S_t = s'\right] = \sum_{r \in R} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)}$$

- Policies ⟶

* Mapping from set of states to set of possible actions per stage.

* Agent will need to learn for the state which are the appropriate actions.

* Deterministic policy ⟶

$$\pi : S \longrightarrow A$$

$$\pi(\text{low}) = \text{Recharge}$$

$$\pi(\text{high}) = \text{Search}$$

* Stochastic policy ⟶

$$\pi : S \times A \longrightarrow [0,1]$$

$$\pi(a|s) = P(A_t = a \mid S_t = s)$$

- Probability that agent takes actions a while in state s.

$$\pi(\text{recharge} \mid \text{low}) = 0.5$$

$$\pi(\text{wait} \mid \text{low}) = 0.4$$

$$\pi(\text{search} \mid \text{low}) = 0.1$$

$$\pi(\text{search} \mid \text{high}) = 0.9$$

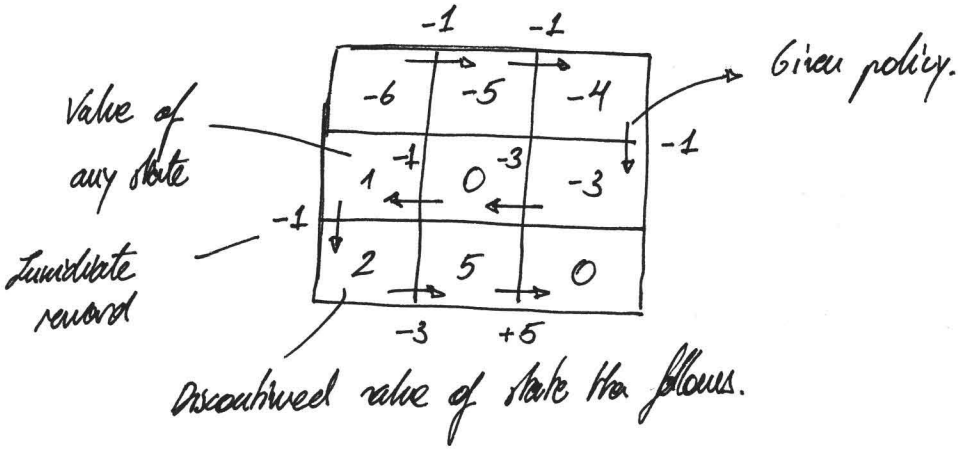
$$\pi(\text{wait} \mid \text{high}) = 0.1$$

- State - Value function →

* For each state, state-value function yields expected return, if the agent started in that state, and follow the policy for all the steps.

$\mathbb{E}_\pi[.]$ ≡ Expected value of a random variable.

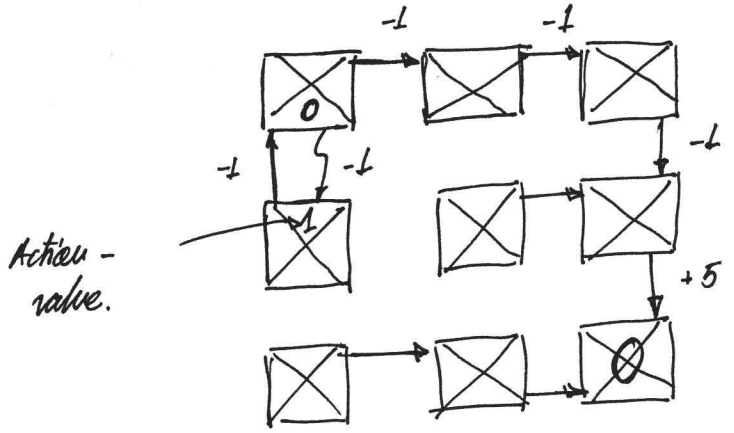
$$V_\pi(s) = \mathbb{E}_\pi[G_t / S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} / S_t = s\right], \text{ for all } s \in \mathcal{S}$$



- Action - value function →

* For each state, action; the action-value function yields expected return, if the agent started in that state, does the action, and then follows the policy for all next steps.

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t / S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} / S_t = s, A_t = a\right], \text{ for all } s \in \mathcal{S}, a \in A(s)$$



* Deterministic policy →

$$V_\pi(s) = q_\pi(s, \pi(s))$$

- Bellman Expectations Equations →

* Value - state →

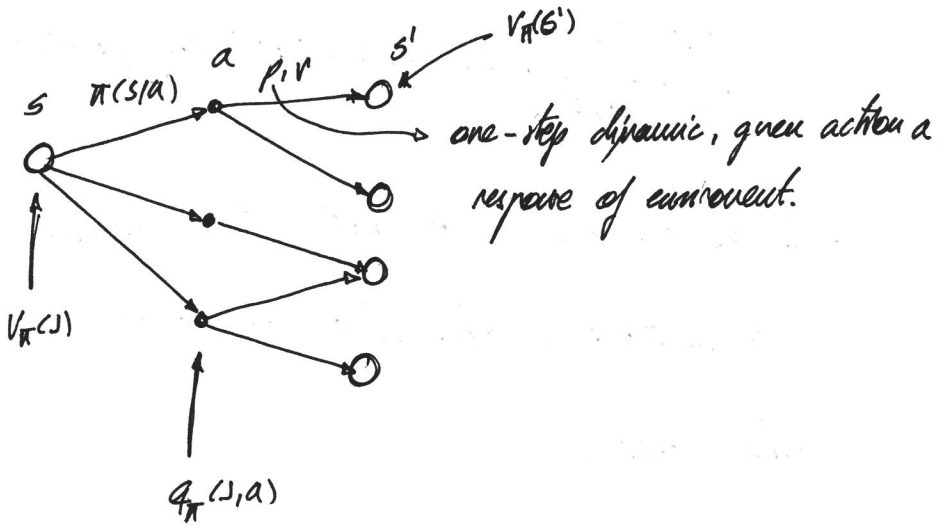
$$V_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s \right] =$$

$$= \sum_{a \in A(s)} \pi(a/s) \sum_{s' \in S} \sum_{r \in R} p(s', r / s, a)\left(r + \gamma V_\pi(s')\right)$$

* Action - Value →

$$q_\pi(s,a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s,\ A_t = a \right] =$$

$$= \sum_{s' \in S} \sum_{r \in R} p(s', r / s, a)\left(r + \gamma \sum_{a' \in A(s')} \pi(a'/s')\, q_\pi(s'/a')\right)$$



- Bellman Optimality Equations →

+ State - value →

$$v_*(s) = \max_{a \in A(s)} \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) / S_t = s \right] =$$

$$= \max_{a \in A(s)} \sum_{s' \in S} \sum_{r \in R} p(s', r / s, a)\left(r + \gamma v_*(s') \right)$$

- Expresses the value of the state under the optimal policy, in terms of the successor states.

+ Action - value →

$$q_*(s, a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a' \in A(S_{t+1})} q_*(S_{t+1}, a') / S_t = s, A_t = a \right] =$$

$$= \sum_{s' \in S} \sum_{r \in R} p(s', r / s, a)\left(r + \gamma \max_{a' \in A(s)} q_*(s', a') \right)$$

- Expresses the value of the state-action pair s, a under an optimal policy in terms of values of the successor state-action s', a'. Under optimal policy.

+ Optimal policy →

π_* satisfies $\pi_* \geq \pi$ for all π.

$\pi' \geq \pi$ if and only if $v_{\pi'}(s) \geq v_\pi(s)$ for all $s' \in S$.

- Deriving Bell Equations →

* Probability space →

Mathematical construct that models a real-world process consisting of states that occur randomly.

Consists:

$\Omega \equiv$ sample space, set of possible outcomes.

$F \equiv$ set of events, each event is a set containing 0 or more outcomes.

$P \equiv$ Assignment probabilities to the events.

* Law of total expectation →

$X \equiv$ Random variable of probability space (Ω, F, P)

$Y \equiv$ Random variable of probability space (Ω, F, P)

$$E[X] = E[E[X|Y]]$$

- If the set of values is finite for X, Y. It can be proofed as →

$$E[X] = \sum_i E[X|A_i] P(A_i)$$

$$E[E[X|Y]] = E\left[\sum_x x P(X=x|Y)\right] = \sum_y \sum_x x P(X=x|Y) \cdot P(Y=y)$$

$$= \sum_y \sum_x x P(X=x, Y=y) = \sum_x x \sum_y P(X=x|Y=y) = \sum_x x P(X=x)$$

$$= E[X]$$

If they are finite, we can switch variables.

* Starting point →

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \Big[\ \text{Law of total expectation: } \mathbb{E}[X] = \sum_i \mathbb{E}[X \mid A_i]\, P(A_i)\ \Big] =$$

$$= \sum_{s' \in S} \sum_{r \in R} P(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a)\, \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a, S_{t+1} = s', R_{t+1} = r]$$

$$= \sum_{s' \in S} \sum_{r \in R} p(s', r \mid s, a)\, \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a, S_{t+1} = s', R_{t+1} = r] =$$

$$\Big[\ \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a, S_{t+1} = s', R_{t+1} = r] = \mathbb{E}_\pi[G_t \mid S_{t+1} = s', R_{t+1} = r]\ \Big]$$

$$= \sum_{s' \in S} \sum_{r \in R} p(s', r \mid s, a)\, \mathbb{E}_\pi[G_t \mid S_{t+1} = s', R_{t+1} = r] =$$

$$= \sum_{s' \in S} \sum_{r \in R} p(s', r \mid s, a)\, \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_{t+1} = s', R_{t+1} = r] =$$

G_{t+1} doesn't depend on R_{t+1}, dropped.

$$= \sum_{s' \in S} \sum_{r \in R} p(s', r \mid s, a)\, \big[r + \gamma\, \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']\big] =$$

by definition.

$$= \sum_{s' \in S} \sum_{r \in R} p(s', r \mid s, a)\, \big[r + \gamma\, v_\pi(s')\big]$$

$$v_\pi(s) = \sum_{a \in A(s)} \pi(a \mid s)\, q_\pi(s,a)$$

→ With these two state-value, action-value functions.

* optimal equations →

$$q_*(s,a) = \sum_{s' \in S} \sum_{r \in R} p(s', r \mid s, a)\,(r + \gamma v_*(s'))$$

$$v_*(s) = \max_{a \in A(s)} q_*(s,a)$$

$$\pi_*(s) = \arg\max_{a \in A(s)} q_*(s,a)$$

for all $s \in S$.

with these two, optimal state-value, action-value functions can be derived.

Dynamic Programming

- Policy evaluation →

  * Assumption that the environment is finite MDP. S, A, R are finite and its dynamics are given by $p(s', r \mid s, a)$ where $s \in S$, $a \in A(s)$, $r \in R$, and $s' \in S^+$

  * DP can be applied to continuous state and value spaces, but an exact solution is not always guaranteed.

  * Main idea →

    The value functions are used to organize and structure the search for good policies.

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s' \in S^+} \sum_{r \in R} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$$

  - The existance and uniqueness for $v_\pi(s)$ is guaranteed as long as $\gamma < 1$ and eventual termination is guaranteed from all states under policy π.

  - If environmental dynamics are known $p(s', r \mid s, a)$ →

    $|S|$ simultaneous linear equations in $|S|$ unknown

    To compute this, it's really expensive.

- Iterative concept →

1. Assign first value to state-values →

$V_0(s_i) = x$ ← Policy at iteration 0.

2. For each iteration, update state-value function →

$$V_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\,[r + \gamma V_k(s')]$$

for all $s \in S$.

Policy at iteration $k+1$.

$V_{k+1}(s) \equiv$ state-value function under policy $k+1$, approximation to π with iter.

3. If $k \to \infty$, $V_k = V_\pi$. $\{V_k\}$ sequence will converge to V_π ⟹

Iterative policy evaluation.

- To compute this →

* Two arrays. $V_k(s)$ and $V_{k+1}(s)$

* One array, updates values in the same array.

Second option converges faster, since it uses the updated values as soon as they are available.

- Algorithm →

Inputs: MDP, policy π, small positive number θ.

Output: state-value function $\approx V_\pi(s)$

Initialize $V(s)$ for all $s \in S^+$ arbitrarily, except $V(\text{terminal}) = 0$.

Repeat until $\Delta < \theta$:

$\Delta \leftarrow 0$

for $s \in S$:

$v \leftarrow V(s)$

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\,[r + \gamma V(s')]$$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

- Estimation Action Values →

$$q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s, A_t = a]$$

$$= \sum_{s' \in S} \sum_{r \in R} p(s', r \mid s, a)[r + \gamma V_\pi(s')]$$

Input: State-value function V

Output: Action-value function Q

For $s \in S$:
  For $a \in A(s)$:
    $q(s,a) \leftarrow \sum_{s' \in S} \sum_{r \in R} p(s', r \mid s, a)(r + \gamma V(s'))$
  End
End

Return q.

- Policy Improvement →

* How to find a better policy? Consider state s, action a and then follow policy π.

$$q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma V_\pi(s') \mid S_t = s, A_t = a]$$

$$= \sum_{s', r} p(s', r \mid s, a)[r + \gamma V_\pi(s')]$$

* If $q_\pi(s, \pi'(a)) \geq V_\pi(s)$ →

$$V_\pi(s) \leq q_\pi(s, \pi'(s)) = \mathbb{E}_{\pi'}[R_{t+1} + \gamma V_\pi(s') \mid S_t = s, A_t = \pi'(s)] =$$

Deterministic policy. →

$$= \mathbb{E}_{\pi'}[R_{t+1} + \gamma V_\pi(s') \mid S_t = s] \leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(s)) \mid S_t = s] =$$

$$= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma V_\pi(S_{t+2})] \mid S_t = s] =$$

Unroll until all policies followed by π'

Rewards under policy π' ←

$$= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 V_\pi(S_{t+2}) \mid S_t = s] \leq \ldots = V_{\pi'}(s)$$

* New policy ⟶

$$\pi'(s) = \arg\max_a q_\pi(s,a)$$

* If $v_\pi(s) = v_{\pi'}(s)$, that means the policy reaches optimal.

* this case was for deterministic policy, it can be shown for stochastic also.

* Algorithm ⟶

Input: MDP, value function V

Output: Policy π'

For $s \in S$ do

  For $a \in A(s)$ do

    $Q(s,a) \leftarrow \sum_{s'r} p(s',r|s,a)(r + \gamma V(s'))$

  End

  $\pi'(s) \leftarrow \arg\max_{a \in A(s)} Q(s,a)$

End

Return π'

* If some state has more than one actions with same $q(s,a)$ ⟶

$$\pi'(a|s) = 0 \text{ if } a \notin \arg\max_{a' \in A(s)} Q(s,a')$$

Other, fall free.

- Policy Iteration ⟶

$$\pi_0 \xrightarrow[E]{} V_{\pi_0} \xrightarrow[I]{} \pi_1 \xrightarrow[E]{} V_{\pi_1} \xrightarrow[I]{} \cdots \xrightarrow[I]{} \pi_* \xrightarrow[E]{} V_*$$

$E \equiv$ Iterative policy evaluation.

$I \equiv$ Policy improvement

Because we have a finite MDP, it has a finite number of policies, this process will converge to the optimal.

* Algorithm ⟶

Input: MDP, small θ.

Output: policy $\pi \simeq \pi_*$

Initialization policy ⟶ $\pi(a|s) = 1/|A(s)|$ for all $s \in S$ and $a \in A(s)$.

Repeat

- $V(s) \leftarrow$ Policy evaluation.
- $\pi' \leftarrow$ Policy improvement.
- If $\pi = \pi'$ then break
- $\pi \leftarrow \pi'$

return π

- Truncated Policy Iteration ⟶

* On policy evaluation ⟶ Instead θ, replace with fix number of iterations.

Optimal Action-Value

	a_1	a_2	a_3
s_1	i	②	-3
s_2	-2	1	③
s_3	⑤	4	-1

Incorrect estimate Action-Value

	a_1	a_2	a_3
s_1	0	④	-2
s_2	-3	-2	⑥
s_3	②	1	0

⟶ have policy optimal.

$\pi_*(s_1) = a_2$

$\pi_*(s_2) = a_3$

$\pi_*(s_3) = a_1$

$A = \{a_1, a_2, a_3\}$; $S \in \{S, s_1, s_2, s_3\}$

I can leave with the estimate if I reach same optimal policy.

- Value Iteration ⟶ Same only one line

↓

Policy Iteration ⟶ for $s \in S$:

$$V(s) \leftarrow \sum_{s'} \sum_{r} p(s', r \mid s, \pi(s))\,(r + \gamma V(s'))$$

Policy Improvement ⟶ for $s \in S$:

for $a \in A(s)$:

This can be combined ⟵ (

$$Q(s,a) \leftarrow \sum_{s'} \sum_{r} p(s', r \mid s, a)\,(r + \gamma V(s'))$$

$$\pi'(s) \leftarrow \arg\max_{a \in A(s)} Q(s,a)$$

↓

Policy Iteration ⟶ for $s \in S$:

$$V(s) \leftarrow \sum_{s'} \sum_{r} p(s', r \mid s, \pi(s))\,(r + \gamma V(s'))$$

Policy Improvement ⟶ for $s \in S$: $f(a)$

$$\pi'(s) \leftarrow \arg\max_{a \in A(s)} \sum_{s'} \sum_{r} p(s', r \mid s, a)\,(r + \gamma V(s'))$$

Policy Iteration ⟶ for $s \in S$: $f(\pi(s))$

$$V(s) \leftarrow \sum_{s'} \sum_{r} p(s', r \mid s, \pi(s))\,(r + \gamma V(s'))$$

$$V(s) \leftarrow \arg\max_{a \in A(s)} \sum_{s'} \sum_{r} p(s', r \mid s, a)\,(r + \gamma V(s'))$$

* Algorithm ⟶

Input: MDP, small positive number θ.

Output: policy $\approx \pi_*$

$V(s) \leftarrow 0$ for all $s \in S^+$

Repeat until $\Delta < \theta$:

$\Delta \leftarrow 0$

loop over states to converge to the state-value functions.

For $s \in S$:

$v \leftarrow V(s)$

$$V(s) \leftarrow \max_{a \in A(s)} \sum_{s' \in S} \sum_{r \in R} p(s', r | s, a)(r + \gamma V(s'))$$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

For $s \in S$:

$$\pi(s) = \arg\max_{a \in A(s)} \sum_{s' \in S} \sum_{r \in R} p(s', r | s, a)(r + \gamma V(s'))$$

Obtain policy corresponding to the final value-function.

Return π

Monte Carlo Methods

- Restatement of goal ⟶

  * For any episode $S_0, A_0, \ldots, R_1, S_1, \ldots, S_T$; the agent's goal is to maximize the reward ⟶

    Find best policy π to max $\mathbb{E}_\pi\left[\sum_{t=1}^{T} R_t\right]$

- Prediction problem ⟶

  * Here the environment dynamics are unknown. Given policy π, determine v_π (or q_π) by interacting with the environment (Monte Carlo).

  * Off-policy ⟶

    - One policy to evaluate, another to interact with the environment.
    - Generates episodes following policy b, where $b \neq \pi$. Then use episodes to evaluate π.

  * On-policy ⟶

    - Agent interacts with the environment following policy π. From the generated episodes, estimate π.

- On-policy →

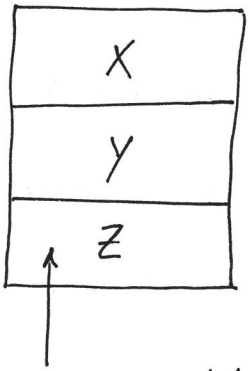
* MC predictions →

- Averaging through the sampled returns for each state would give the expected return, calculated by sampling as long as we have enough visits to the state.

- State values →

$S^+ = \{ x, y, z \}$

$A = \{ \uparrow, \downarrow \}$; Actions are not deterministic



- Evaluate policy through episodes generated following it →

$\pi(x) = \uparrow, \quad \pi(y) = \downarrow$

Episodes

1. $x, \uparrow, -2, y, \downarrow, 0, y, \downarrow, 3, z.$

2. $y, \downarrow, 2, y, \downarrow, 1, y, \downarrow, 0, z.$

3. $y, \downarrow, 1, x, \uparrow, -3, y, \downarrow, 3, z.$

$V_\pi(x) = E_\pi [G_t / S_t = x] \equiv$ Expected return after state x.

$V_\pi(x) = \dfrac{\text{Returns}}{\text{Ocurrences.}}$

* First-Visit MC Method →

- Only consider the first visit per episode.

$$V_\pi(X) = \frac{1+0}{2}$$

* Every-Visit MC Method →

- Consider all the visits in the same episode.

$$V_\pi(X) = \frac{\overline{3+3} + \overline{3+1+0} + \overline{1+3}}{7} = \frac{14}{7} = 2.$$

* Properties →

- Both methods converge to $V_\pi(s)$ as # visits converge to infinite.

- First-visit: each return is an independent event, identically distributed with finite variance of $V_\pi(s)$. By "Law of large numbers" the sequence of averages of these estimates converges to the expected value.

- First-visit unbiased, Every-visit biased.

- Initially, MSE lower of every visit with episodes counted first-visit.

* Algorithm ⟶

Input: Policy π, num_episodes.

Output: value-function V ($\approx v_\pi$ if num_episodes large enough).

Initialize $N(s) = 0$ for all $s \in S$.

Initialize return_sum$(s) = 0$ for all $s \in S$.

For $i = 1$ to num_episodes:

    Generate episode following π.

    For $t = 0$ to $T-1$:

        if S_t is a first visit (with return G_t) then

            $N(S_t) \leftarrow N(S_t) + 1$

            return_sum$(S_t) \leftarrow$ return_sum$(S_t) + G_t$

        end

    end

end

$V(s) \leftarrow$ return_sum$(s) / N(s)$ for all $s \in S$.

- Action-value function →

* If a model is available, state-value functions are enough to determine a policy → just look to the next step that maximizes reward.

* Without the model, we need to explicitly estimate the value of each action, in order for the values to be useful in suggesting a policy.

* Deterministic policy looking at each state-action pair →

Episode

1. $X, \uparrow, -2, Y, \downarrow, 0, Y, \downarrow, 3, Z.$

2. $Y, \uparrow, 2, Y, \downarrow, 1, Y, \downarrow, 0, Z.$

3. $Y, \downarrow, 1, X, \uparrow, -3, Y, \downarrow, 3, Z.$

* MC prediction is not used with a deterministic policy →

$\pi(X) = \uparrow$
$\pi(Y) = \downarrow$
→ $q_\pi(X, \downarrow) = ?$; $q_\pi(Y, \uparrow) = ?$
These would ever be used.

* To compare alternatives we need to estimate the value of all the actions for each state, not the ones we currently favour.

* Instead we stochastic. As number of episodes, wants to pair converge to ∞. the number of visits will approximate to the prob distribution based on MC.

* First-visit, Every-visit.

* Algorithm ⟶

Input: Policy π, num_episodes.

Output: Action-value function Q ($\approx q_\pi$ if num_episodes large).

Initialize $N(s,a) = 0$ for all $s \in S$, $a \in A(s)$.

Initialize return_sum$(s,a) = 0$ for all $s \in S$, $a \in A(s)$.

For $i = 1$ to num_episodes:

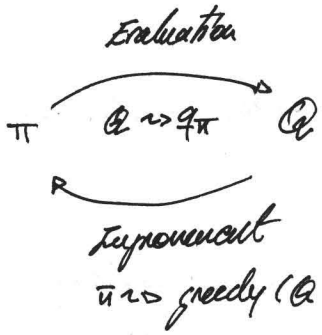
| Generate episode $S_0, A_0, \ldots, S_T$ using π.
| For $t = 0$ to $T-1$:
| | If (S_t, A_t) is a first visit (with return G_t) then
| | | $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
| | | return_sum$(S_t, A_t) \leftarrow$ return_sum$(S_t, A_t) + G_t$
| | End
| End

End

$Q(s,a) \leftarrow$ return_sum$(s,a) / N(s,a)$ for all $s \in S$, $a \in A(s)$.

return Q.

- Monte Carlo Control →



$\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \pi_2 \longrightarrow \dots$

* Policy improvement → greedy policy $\pi(s) \doteq \arg\max_a q(s,a)$

$$q_{\pi_{k+1}}(s, \pi_{k+1}(s)) = q_{\pi_{k+1}}(s, \arg\max_a q_{\pi_k}(s,a)) = \max_a q_{\pi_k}(s,a)$$

$$\geq q_{\pi_k}(s, \pi_k(s)) \geq v_{\pi_k}(s)$$

Each π_{k+1} is uniformly better than π_k, or as good.

* This is based in two assumptions → converges to optimal policy.

1. Episodes have exploring starts, so you can get samples for all state, actions instead of going greedy, which might just give you a local optimum.

2. Policy evaluation is done over an infinite number of episodes.

* Policy evaluation problems →

- As in Dynamic programming, two approches, doing policy evaluation to a number of iterations until we converge up to a level of approximation for the action and state-value functions.

- Do as in DP value-iteration, after each episode evaluate policy

* Incremental mean ⟶

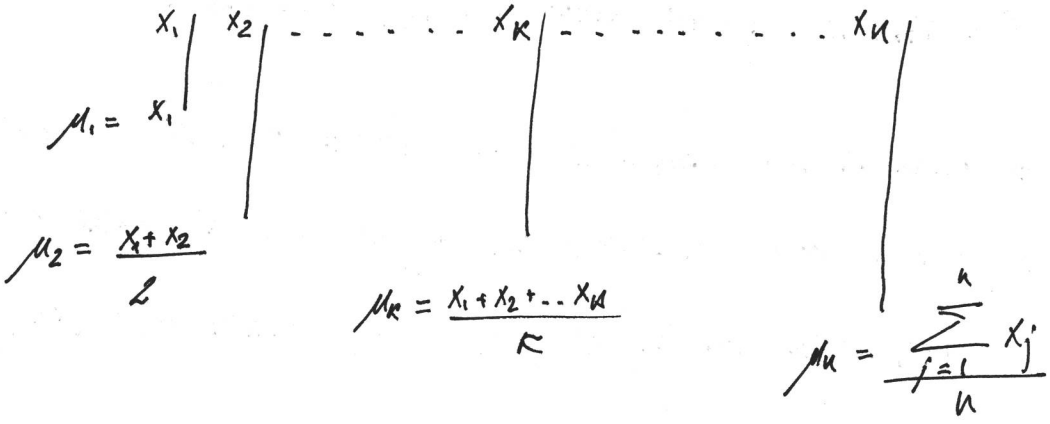
- Run n episodes.
- Keep track of reward following each state-value (action-value) visit per episode ⟶

$x_1, x_2, x_3, \ldots, x_n$ ⟶ reward per episode for a state.

- MC estimation ⟶

$$\underset{\text{MC}}{\text{action-value}} \equiv \mu_n = \frac{\sum_{j=1}^{n} x_j}{n}$$

- Incremental mean, update estimate after each visit ⟶



$$\mu_K = \frac{1}{K}\sum_{j=1}^{K} x_j = \frac{1}{K}\left(x_K + \mu_{K-1}(K-1)\right) \longrightarrow$$

$$\mu_K = \mu_{K-1} + \frac{1}{K}\left(x_K - \mu_{K-1}\right)$$

μ_{K-1} ⟶ previous mean

$\frac{1}{K}\left(x_K - \mu_{K-1}\right)$ ⟶ proportional update.

- Algorithm ⟶

$\mu \leftarrow 0$

$k \leftarrow 0$

while $k < n$:

$n \leftarrow k+1$

$\mu \leftarrow \mu + \frac{1}{k}(x_k - \mu)$

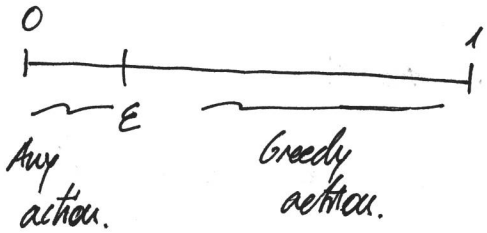
* Overall, you run policy evaluation & improvement so on, this concept can be used to cut all the iterations for the evaluation and update immediate.

* Epsilon - Greedy Policy ⟶

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A(s)|} & \text{if } a = \arg\max_{a' \in A(s)} Q(s,a') \\ \frac{\varepsilon}{|A(s)|} & \text{otherwise.} \end{cases}$$

$|A(s)| \equiv$ Number of possible actions from state $s \in S$.

- $\varepsilon \uparrow \equiv$ More likely to pick the non-greedy action.



* Exploration / Exploitation dilemma →

- One potential solution is to implement in a way that gradually reduces ε.
- Start from any exploration, along with the agent knowing the environment dynamics. Limit exploration →
  * Start with equiprobable random policy. → $\varepsilon = 1$.
  * One time agent becomes more greedy.

* Greedy in the Limit with Infinite Exploration ≡ GLIE →

- To ensure convergence to optimal policy π_*

1. Condition → Every state-action is visited infinitely many times.
2. Condition → Policy converges to a policy that is greedy with respect to action-value estimate Q.

- To satisfy this conditions →

  $\varepsilon_i \equiv \varepsilon$ at ith time step.

  * $\varepsilon_i > 0$ for all time steps i.
  * ε_i decays to 0 in the limit as the time step i approaches infinite → $\lim_{i \to \infty} \varepsilon_i = 0$.

- E.g → $\varepsilon_i = 1/i$

- MC Control: GLIE Algorithm ⟶

Input: positive integer num_episodes.

Output: policy π ($\approx \pi_*$ if num_episodes is large enough).

Initialize $Q(s,a) = 0$; $N(s,a) = 0$ for all $s \in S$, $a \in A(s)$.

for $i = 1$ to num_episodes:

$\varepsilon \leftarrow 1/i$

$\pi \leftarrow \varepsilon$-greedy($Q$)

Generate episode using π.

for $t = 0$ to $T-1$:

if (S_t, A_t) is a first visit (with return G_t)

$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t))$

end

end

end

return π.

you don't even know Q at the beginning, act on fly following policy to estimate Q

↳ After calculate policy

- MC Control : Constant-alpha ⟶ "forgetful mean"

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t))$$

* Learning on $Q(S_t, A_t)$ with two becomes smaller

Immediate return — Expected value of action-value

$G_t \equiv$ Error return.

Update by this difference

* Instead use α factor ⟶ "running mean"

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha (G_t - Q(S_t, A_t))$$

* With this, most later rewards are emphasized

* Setting α value ⟶

- $\alpha = 0$ ⟶ never updated.
- $\alpha = 1$ ⟶ immediate return.

} ⟶ $\alpha \in (0, 1)$

- Smaller values of α encourages the agent to consider a longer history, increasing more recent.

- If α is too close to one it may prevent for the agent to find the π_*.

- MC Control: GLIE α-contract $\longrightarrow$

Input: num_episodes, α

Output: policy π

Initialize Q arbitrarily (eg $Q(s,a)=0$ for all $s \in S$, $a \in A(s)$).

For $i=1$ to num_episodes:

$\varepsilon \leftarrow 1/i$

$\pi \leftarrow \varepsilon$-greedy$(Q)$

Generate episode following π.

For $t=0$ to $T-1$:

If (S_t, A_t) is a first-visit:

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t))$

end

end

end

return π.

Temporal-Difference Methods

- Temporal-Difference prediction ⟶

  * Given policy π, estimate v_π (or q_π).

  * Focus on update step ⟶

$$Q(S_t, A_t) \longleftarrow Q(S_t, A_t) + \alpha\left(G_t - Q(S_t, A_t)\right)$$

$$V(S_t) \longleftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

Bellman equation.

$$V(S_t) \longleftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_t) - V(S_t)\right)$$

  * Combines the Bellman equation expectation and the MC estimation.

  * Now, the state-value doesn't depend on the final state and reward, but just on the next one.

  * We can update immediately.

  * In practice, TD converges faster than MC.

  * TD is guaranteed to converge as long as the step-size is small enough.

* Comparing DP, MC and TD →

$$v_\pi(s) = \mathbb{E}_\pi[G_t / S_t = s] \quad (1)$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} / S_t = s]$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) / S_t = s]$$

- MC estimates function (1), it is an estimation because the expectation is unknown and it is calculated from sampling.

- Dynamic programming is an estimate because v_π is unknown and it is approximated using $V(s_t)$

- Temporal Difference is a difference estimate for both reasons, it samples the expected values and it uses the current state V instead of v_π.

* TD error → (→ Not available until next step.)

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

- Learning difference between estimated $V(S_t)$ and the one given by iteration, reward.

* MC error is the accumulated TD errors, if the $V(s_t)$ are not updated until the end of the episode.

$$\underbrace{G_t - V(s_t)}_{\text{MC Error}} = R_{t+1} + \gamma G_{t+1} - V(s_t) + \gamma V(s_{t+1}) - \gamma V(s_{t+1})) =$$

$$= R_{t+1} + \gamma V(s_{t+1}) + \gamma(G_{t+1} - V(s_{t+1})) =$$

$$\underbrace{\qquad\qquad -V(s_t)}$$

$$= \delta_t + \gamma(G_{t+1} - V(s_{t+1})) = \delta_t + \gamma\delta_{t+1} + \gamma^2(G_{t+2} - V(s_{t+2}))$$

$$= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+1} + \dots + \gamma^{T-t}\underbrace{(G_t - V(s_t))}_{0-0} =$$

$$= \sum_{k=t}^{T-1} \gamma^{k-t}\delta_t$$

* the identity is not exact if $V(s_t)$ is updated during the episode. But if the step-size α is small enough it may hold an approximation.

$$Q(s_0, A_0) \leftarrow Q(s_0, A_0) + \alpha(R_1 + \gamma(Q(s_1, A_1) - Q(s_0, A_0)))$$

- Sarsa →

* Determine π_* →

→ Initialize $Q(s,a) = 0$ for all $s \in S$, $a \in A(s)$

$\pi \leftarrow \varepsilon$-greedy$(Q)$; $\varepsilon = 1$

$S_0\ A_0\ R_1\ S_1\ A_1 \mid R_2\ S_2\ A_2 \mid \ldots\ R_{t+1}\ S_{t+1}\ A_{t+1}$

Update $Q(S_0, A_0)$ → $\pi \leftarrow \varepsilon$-greedy

Update $Q(S_1, A_1)$ → $\pi \leftarrow \varepsilon$-greedy

Update $Q(S_t, A_t)$ → $\pi \leftarrow \varepsilon$-greedy.

- Guaranteed to converge to optimal action-value function as long as α is small enough, and GLIE conditions are met →

1. $\varepsilon_i > 0$ for all time-step i.

2. $\lim_{i \to \infty} \varepsilon_i = 0$.

- It will be true as long as we run episodes long enough →

$$\pi_*(s) = \arg\max_{a \in A(s)} q_*(s,a)$$

* Sarsa Algorithm →

Input: π, num_episodes, α

Output: $Q \approx q_\pi$ (if num_episodes is large enough).

Initialize Q arbitrarily (e.g. $Q(s,a) = 0$ for all $s \in S$, $a \in A(s)$).

including $Q(\text{terminal\_state}, \cdot) = 0$.

For $i = 1$ to num_episodes:

- $\varepsilon \leftarrow 1/i$
- Observe S_0
- Choose action A_0 from policy derived from Q (e.g. ε-greedy).
- $t \leftarrow 0$.
- Repeat
  - Take action A_t and observe S_{t+1}, R_{t+1}
  - Choose action A_{t+1} using policy derived from Q (eg. ε-greedy).
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$
  - $t \leftarrow t+1$
- until S_t is terminal.

end

return Q

- Q-learning (bootmax) →

S_0 A_0 R_1 S_1 | A_1 → ε-greedy policy. → Action decision.

$$Q(S_0, A_0) \leftarrow Q(S_0, A_0) + \gamma(R_1 + \gamma \max_{a \in A(s)} Q(S_1, a) - Q(S_0, A_0))$$

* For the update anmie greedy policy, it approximates to the optimal policy.

* Algorithm →

Input: π, num_episodes, α

Output: Q ($\approx q_\pi$ if num_episodes large enough).

Initialize Q arbitrarily.

For $i=1$ to num_episodes:

- $\varepsilon \leftarrow 1/i$
- Observe S_0
- $t \leftarrow 0$
- Repeat
  - Choose A_t using policy derived from Q (ε-greedy).
  - Take A_t and observe S_{t+1}, R_{t+1}
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
  - $t \leftarrow t+1$
- until S_t is terminal

end

return Q.

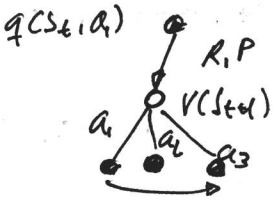
- Expected Sarsa ⟶

* Guaranteed to converge to the q_π under the same conditions as Sarsa.

Eliminates the variance from the random selection of A_{t+1}

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma \sum_{a \in A} \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)\right)$$

* Algorithm ⟶



Input: π, num_episodes, α

Output: Q ($\approx q_\pi$ if num_episodes large enough).

Initialize Q arbitrarily.

For $i = 1$ to num_episodes:

- $\varepsilon \leftarrow 1/i$
- Observe S_0
- $t \leftarrow 0$
- repeat
  - Choose action A_t using policy derived from Q (ε-greedy)
  - Take action A_t and observe R_{t+1}, S_{t+1}
  - $Q(S_t, A_t) \leftarrow$
    $$Q(S_t, A_t) + \alpha\left(R_{t+1} + \alpha \sum_{a \in A(S_{t+1})} \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)\right)$$
  - $t \leftarrow t+1$
- until S_{t+1} is terminal

end

return Q

- Analytically performance →

* All these algorithms converge if these conditions are met →

1. The value of ε decays according to GLIE.

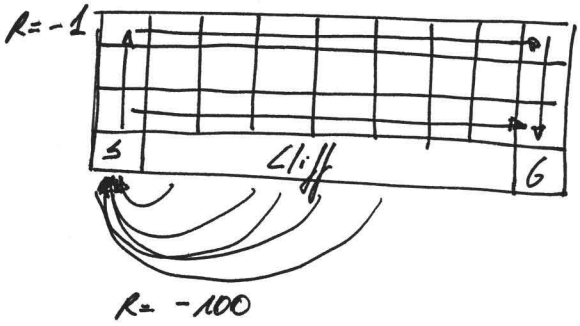
2. κ is sufficiently small.

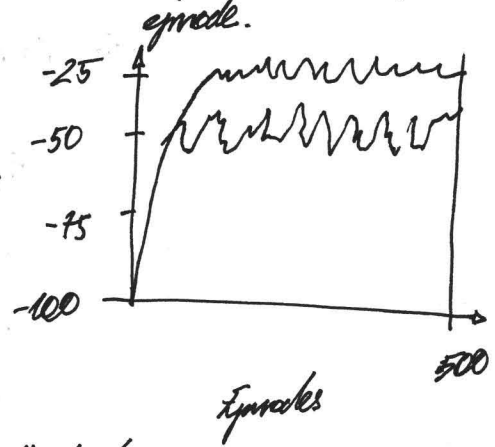
* Differences →

- Sarsa, Expected sarsa → on-policy TD. ⇒ Better on-line performance than off-policy.

- Sarsamax → off-policy

- Expected Sarsa better performance than Sarsa.





Results are from a single run but smoothed by averaging reward runs from 10 successive episodes.

- Sarsa and Q-learning are $\varepsilon = 0.1$ →

1. Q-learning achieve worse online performance but learns the optimal policy.

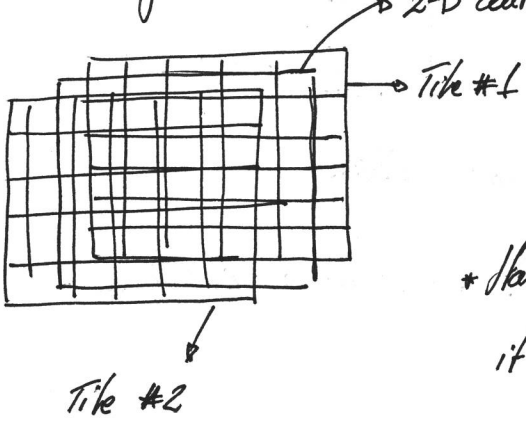
2. Sarsa achieves better online performance, but learns a sub optimal "safe" policy.

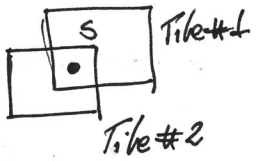
Reinforcement learning in continuous spaces

* Discretization →

- Approximate continuous function through uniform / non-uniform discretization.

- Tile coding →





* State is associated to each cell of the tile in which it falls

* Uses different tiles with certain offset. →
  Manual selection of cells, more offset.

- Algorithm → S, A, P, R, γ, m, n

```
for i=1 to m:
    Initialize tiling i with n/m tiles
    for j=1 to n/m:
        Initialize tile j with zero weight.

repeat
    s ← random state from S
    ΔV(s) ← max_a [ R(s,a) + γ V(P(s,a)) - V(s) ]

    for i=1 to m:
        w ← weight of active-tile (i,i)
        w ← w + (κ/m) ΔV(s)

until time expires
```

$$v(s) = \sum_{i=1}^{n} \delta_i(s)\, w_i \longrightarrow \text{weight for each tile.}$$

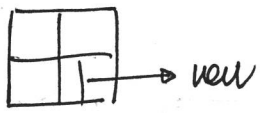
* Adaptative tile coding ⟶

  - Start with given number of cells and divide when appropriate

  - Example ⟶

    * split when we are not learning with our current representation.

    * Stop with number of splits or number of iterations.



* Function Approximation ⟶

  - Represent state-value and action-value functions ⟶

$$\hat{v}(s,w) \approx v_\pi(s)$$

$$\hat{q}(s,a,w) \approx q_\pi(s,a)$$

⟶ parameter w that shapes the function.

  - State-value Approximation ⟶

$$\hat{v}(s,w) = x(s)^T w = \sum_{j=1}^{n} x_j(s)\, w$$

$$x(s) = \begin{pmatrix} x_1(s) \\ \vdots \\ x_n(s) \end{pmatrix}$$

feature vector

* Value function ⟶ $\hat{v}(s,w) = x(s)^T w$

* Minimize error ⟶ $J(w) = \mathbb{E}_\pi\left[(v_\pi(s) - x(s)^T w)^2\right]$

* Error Gradient ⟶

$$\nabla_w J(w) = -2\,(v_\pi(s) - x(s)^T w)\, x(s)$$

* Update rule ⟶

$$\Delta w = -\alpha \frac{1}{2} \nabla_w J(w) = \alpha\,(v_\pi(s) - x(s)^T w)\, x(s)$$

- Action-value Approximation ⟶

$\hat{q}(s,a,w) = x(s)^T w_A$

$\downarrow$

$\hat{q}(s,a_n,w) = x(s)^T w_n$ ⟶ $(x_1(s,a), \dots, x_n(s,a)) \begin{pmatrix} w_{11} \dots w_{1m} \\ \vdots \quad\quad \vdots \\ w_{n1} \dots w_{nm} \end{pmatrix}$

$= (\hat{q}(s,a_1,w), \dots, \hat{q}(s,a_n,w))$

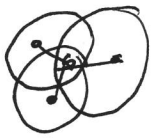
* Allows to control two actions at the same time

- Kernel features ⟶

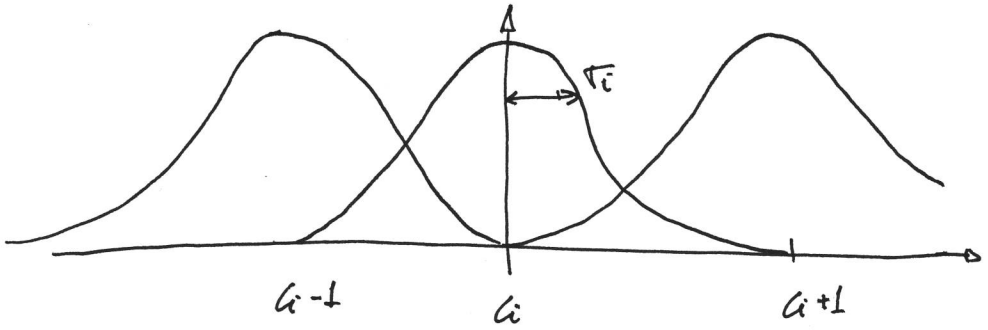
$$x(s) = \begin{pmatrix} x_1(s) \\ \vdots \\ x_n(s) \end{pmatrix} \longrightarrow \begin{matrix} x_1(s) = s \\ x_2(s) = s^2 \\ x_3(s) = s^3 \end{matrix} \Big\} \text{ Kernel functions.}$$

* Radial Basis Function (RBF) ⟶

- Associate each feature with a kernel function



$$\phi_i(s) \doteq \exp\left(-\frac{\|s - c_i\|^2}{2\sigma_i^2}\right)$$



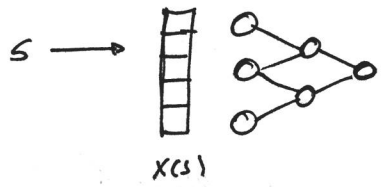
- Non-Linear function Approximation ⟶

$\hat{v}(s,w) = f(x(s)^T w)$ ⟵ DNN.

$\Delta w = \alpha(v_\pi(s) - \hat{v}(s,w)) \nabla_w \hat{v}(s,w)$

Deep Reinforcement Learning

* Neural Networks as Value functions →



$\rightarrow V_\pi(s,w) \equiv$ function approximator.

$s \rightarrow V_\pi(s) \rightarrow V_\pi(s) \in \mathbb{R} \equiv$ state value function under optimal policy.

- Backpropagation and gradient descent to update param w →

$$\text{error} \rightarrow (V_\pi(s) - \hat{v}(s,w))^2$$

$$\Delta w = \alpha (V_\pi(s) - \hat{v}(s,w)) \nabla_w \hat{v}(s,w)$$

$$\Delta w = \alpha (q_\pi(s,a) - \hat{q}_\pi(s,a,w)) \nabla_w \hat{q}_\pi(s,a,w)$$

- Now, question is how to approximate the optimal value functions.

* Monte Carlo learning →

$$V(s_t) \leftarrow V(s_t) + \alpha (G_t - V(s_t))$$

Actual return $\Rightarrow G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$

$$\Delta w = \alpha \left(\frac{V_\pi(s_t)}{G_t} - \hat{v}(s_t,w) \right) \nabla w$$

- MC is guaranteed to converge on a local minimum optima, in case of function approximation, it will converge to a global optima.

* MC function approximation →

Initialize w with random values.

Policy $\pi \leftarrow \varepsilon$-greedy $(\hat{q}(s,a,w))$

Repeat til converge:

Evaluation: Generate an episode $S_0, A_0, R_1, \ldots, S_T$ using π

For $t = 1$ to T

Every visit return. → $\Delta w = \alpha\left(G_t - \hat{q}(S_t, A_t, w)\right) \nabla_w \hat{q}(S_t, A_t, w)$

Improvement: $\pi \leftarrow \varepsilon$-Greedy $(\hat{q}(s,a,w))$

* Temporal Difference Learning →

$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma \underline{V(S_{t+1})} - V(S_t)\right)$$

↳ Estimated return.

$$\Delta w = \alpha\left[R_{t+1} + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)\right] \nabla_w \hat{v}(S_t, w)$$

TD-Error

- TD(0) Control with function approximation →

Initialize w randomly, $\pi \leftarrow \varepsilon$-Greedy $(\hat{q}(s,a,w))$

Repeat many episodes:

Initial State S

While S non terminal

$A \leftarrow$ State S using π.

$R, S' \leftarrow$ env(S, A)

$A' \leftarrow$ State S' using π

Update: $\Delta w = \alpha\left[R + \gamma \hat{q}(S', A', w) - \hat{q}(S, A, w)\right] \nabla_w \hat{q}(S, A, w)$

$S \leftarrow S'$, $A \leftarrow A'$

* Q-learning ⟶

- SARSA & Q-learning are TD approaches, they might not converge to a global optima with non linear function approximation.

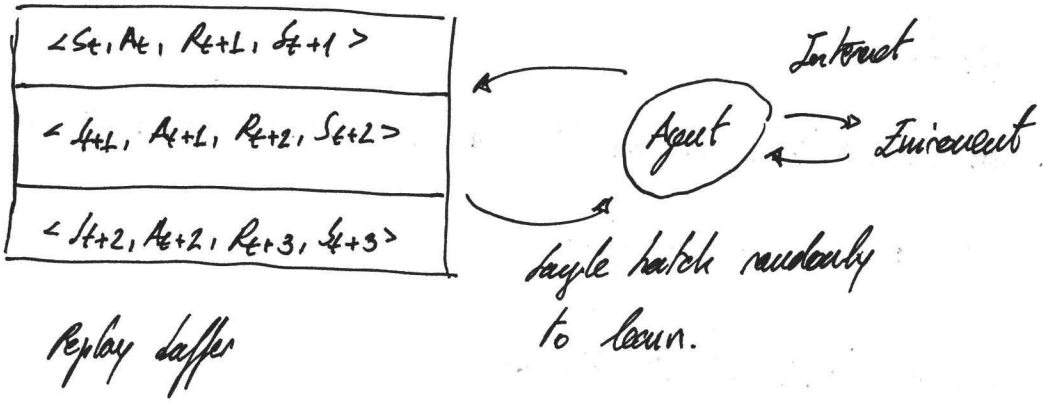
- Update step ⟶ $\Delta w = \alpha \left(R + \gamma \max_a \hat{q}(S', a, w) - \hat{q}(S, A, w) \right)$

- Differences ⟶

SARSA	Q learning
On-policy	Off-policy
Good online performance	Bad online performance
Q-Values affected by exploration	Q-Values unaffected by exploration.

- Off-policy ⟶ Decouples actions taken and learning value function.

* Experience Replay ⟶



- Every action to affect next state S_{t+1}, which means that each experience tuples can be highly correlated.

- Lastly at random from the buffer, breaks the correlations and prevents action values from diverging.

- Function approximation → may be able to approximate, evaluate unseen scenarios, while discretization of leaves 'holes' from previous learning cases.

- Experience replay also helps with rare state, actions →

  * If rare action, state is repeated then it will be used to learn another optimal action.

  * Keeping experience and learning later will help with it.

* Fixed Q-Targets →

  - Helps to address one type of correlation: consecutive experience tuples.

$$\Delta w = \alpha \Big[\underbrace{R + \gamma \max_a \hat{q}(S', a, w)}_{\text{TD target}} - \underbrace{\hat{q}(S, A, w)}_{\text{Current Value}} \Big] \nabla_w \hat{q}(S, A, w)$$

$$q_\pi(S, A)$$

  * Both are calculated based on w → Correlation between target and parameters we are changing.

  * Fix the parameter in the target →

    $w^- \equiv$ copy of w that we don't change during certain # of steps.

    decouples.

$$\Delta w = \alpha \Big[R + \gamma \max_a \hat{q}(S', a, w^-) - \hat{q}(S, A, w) \Big] \nabla_w \hat{q}(S, A, w)$$

* Algorithm ⟶

Initialize replay memory D with capacity N.

Initialize $\hat{q}$ with w randomly.

Initialize target action-value weights $w^- \leftarrow w$

For episode $e = 1$ to M:

Initial input from x_1 (frames)

Initial State $S \leftarrow \phi(x_1)$

For time_step $= 1$ to T:

Sample:

- $A \leftarrow$ State S using $\pi \leftarrow \varepsilon$-Greedy $(\hat{q}(S, A, w))$
- $R, x_{t+1} \leftarrow$ env (S, A)
- $S' \leftarrow \phi(\langle x_{t-2}, x_{t-1}, x_t, x_{t+1} \rangle)$
- Memory $D \leftarrow$ Experience tuple (S, A, R, S')
- $S \leftarrow S'$

Learn:

- Minibatch $\leftarrow (s_j, a_j, r_j, s_{j+1})$; $j \in$ range N.
- Set target $y_j = r_j + \max_a \hat{q}(s_{j+1}, a, w^-)$
- Update $\Delta w = \alpha \left[y_j - \hat{q}(s, a_j, w) \right] \nabla_w \hat{q}(s_j, a_j, w)$
- Every C steps, reset $w^- \leftarrow w$

* DQN Improvements →

- Overestimation of Q values →

$$\Delta W = \alpha \left[R + \gamma \max_a \hat{q}(s', a, w) - \hat{q}(s, A, w) \right] \nabla_w \hat{q}(s, A, w)$$

$$R + \gamma \hat{q}\left(s', \arg\max_a \hat{q}(s', a, w), w\right)$$

Q-Value — State — Action with max Q-Value.

* Notable, usually on first stages, because q values are still evolving and we might have enough samples to figure it out.

* It has been proven that overestimates values.

* Double Q-learning →

$$R + \gamma \hat{q}\left(s', \arg\max_a \hat{q}(s', a, w), w'\right)$$

(best action. / Evaluate action.)

- Two function approximators that must agree on the best action.

If they don't agree, q-value returned not that high.

- This prevents individual high rewards by chance that may have replications in the buffer.

- Recurrents →

Fixed Q Targets. ←

* Choose randomly one to update at each step, using the other for evaluation.

- Prioritized Experience Relay →

* TD Error → $\delta_t = R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, w) - \hat{q}(S_t, A_t)$

* Priority → $P_t = |\delta_t| + e$ → Memory entry $\langle S_t, A_t, R_{t+1}, S_{t+1}, P_t \rangle$

* Sampling probability → $P(i) = \frac{p_i^a}{\sum_k p_k^a}$

- A low δ_t doesn't mean that we meet the optimal, it may be that the estimation is off → $+e$

- This priority sampling may produce some overfit, if we don't weight this to introduce some uniform prob →

  * $a = 0$ → Uniform probability.
  * $a = 1$ → Priority probability.

* Modified update rule →

$$\Delta w = \alpha \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^b \delta_i \nabla_w \hat{q}(S_i, A_i, w)$$

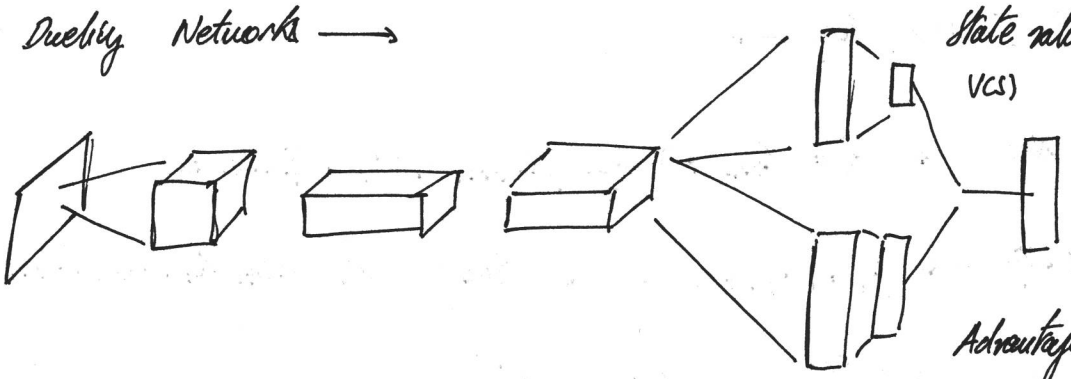
- Before it was an expectation from a sampling of all my experiences, now it must match our distribution.

- Importance sampling weight →

  $b \equiv$ hyperparameter. → $b = 0$ Uniform
  $b = 1$ Pure priority distribution.

  $N \equiv$ buffer size.

- Dueling Networks →



* Intuition →

Values of most states don't vary much across actions.

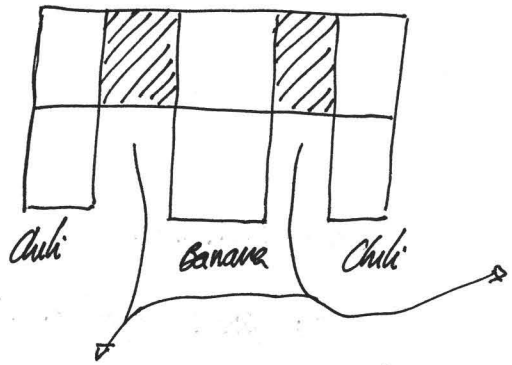
Policy Based Methods

- Reasons ⟶

Deterministic $\pi: S \longrightarrow a$; Stochastic $a \sim \pi(s,a) = P[a|s]$

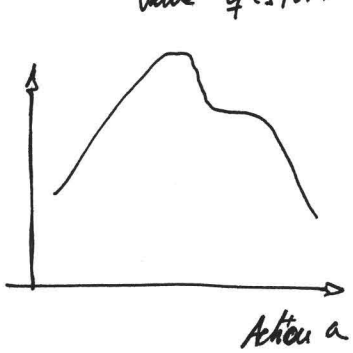
* Policy based methods can learn true stochastic policies ⟶
  - Randomness is derived from the environment, not as a ε-greedy policy which is an approximation.

* Aliased States ⟶
  - Two or more states that we perceive to be identical but are actually different.



* George and banana problem.
* State defined by walls and floor type.
* In the "11" cells actions to take are to be based on previous experiences, states are equal.
* Actions will be the same since features are equal
* This will make George to go back and forth to see on the pairs, it won't get out until ε-greedy provides way out ⟶ Inefficient.

* Continuous Action Spaces ⟶
  - It is now an optimization problem to find maximum.
  - If the function is highly action space dimensional this is not a trivial problem.



- Policy function approximation ⟶

$$a \sim \pi(s, a, \theta) = P[a|s, \theta]$$

* Linear function with softmax policy ⟶

  - For discrete set of actions.
  - We can sample across actions.

$$f(s, a, \theta) = x(s, a)^T \theta$$

$$a \sim \pi(s, a, \theta) = \frac{e^{f(s,a,\theta)}}{\sum_{a'} e^{f(s,a';\theta)}}$$

* Linear function with gaussian policy ⟶

$$f(s, a, \theta) = x(s, a)^T \theta$$

$$a \sim \pi(s, a, \theta) = N(\mu, \sigma^2)$$

$\mu = f(s, a, \theta)$

$\sigma^2 \equiv$ fixed or parameterized.

  - Continuous functions.
  - Sample action from distribution.

- Objective function ⟶

* Evaluate the expected value of the rewards under that policy.

$$J(\theta) = \mathbb{E}_\pi [R(\tau)] \; ; \; \tau \equiv \text{Trajectory, complete or partial episode.}$$

* State values ⟶

- Start state value ⟶ $J_1(\theta) = \mathbb{E}_\pi [G_1] = \mathbb{E}_\pi [V(s_1)]$

- Average state value ⟶ (Better values per occurance of state.)

$$J_{\bar{V}}(\theta) = \mathbb{E}_\pi [V(s)] = \sum_s d(s) V(s)$$

$$d(s) = \frac{N(s)}{\sum_{s'} N(s')} \; ; \text{ Markovian probability}$$

* Action values ⟶

$$J_{\bar{q}}(\theta) = \mathbb{E}_\pi [Q(s,a)] = \sum_s d(s) \sum_a \pi(s,a,\theta) Q(s,a)$$

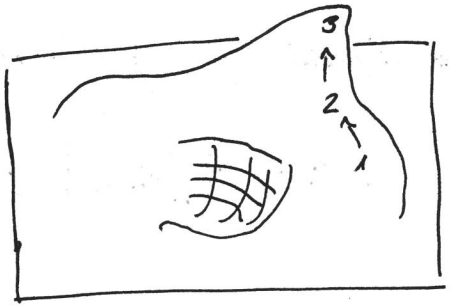
* Rewards ⟶

- Work without state, action value functions.

$$J_{\bar{r}}(\theta) = \mathbb{E}_\pi [r] = \sum_s d(s) \sum_a \pi(s,a,\theta) R_s^a$$

- Stochastic Policy Search ⟶

$J(\theta)$



* Start with arbitrary policy
* Add noise to policy at each step, choose the most promising.

* Adaptative noise ⟶ large noise at the beginning to avoid local maxima, decrease noise as you get closer to max.

- Policy Gradient ⟶

* Having a policy π_θ and objective function $J(\theta)$ ⟶ Compute gradient $\nabla_\theta J(\theta)$ and update $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

* If the function is not differentiable, not possible to find gradient directly.

* Gradient Estimation ⟶

For $k \in [1, n]$: ⟶ each dimension.

$$\frac{\partial J(\theta)}{\partial \theta_k} = \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

(ϵ: step; u_k ⟶ unit vector in k^{th} dimension.)

$$\nabla_\theta J(\theta) \triangleq \left\langle \frac{\partial J(\theta)}{\partial \theta_1}, \frac{\partial J(\theta)}{\partial \theta_2}, \dots, \frac{\partial J(\theta)}{\partial \theta_n} \right\rangle$$

Computationally inefficient.

* Computing gradient Analytically →

- Need access to the underlying policy.

$J(\theta) = \mathbb{E}_\pi[R(\tau)]$; $R(\tau) \equiv$ score function.

$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_\pi[R(\tau)] = \mathbb{E}_\pi[\nabla_\theta(\log \pi(s,a,\theta)) R(\tau)]$

Need to compute gradient from policy, not score function.

$\Delta\theta = \alpha \nabla_\theta(\log \pi(s,a,\theta)) R(\tau)$

- Derivation →

$\nabla_\theta \mathbb{E}_x[f(x)]$; $x \sim P[x|\theta]$

$\nabla_\theta \mathbb{E}_x[f(x)] = \nabla_\theta \sum_x P(x|\theta) f(x) = \sum_x \nabla_\theta P(x|\theta) f(x) =$

Likelyhood Ratio trick

$= \sum_x P(x|\theta) \frac{\nabla_\theta P(x|\theta)}{P(x|\theta)} f(x) = \sum_x P(x|\theta) \nabla_\theta(\log P(x|\theta)) f(x) =$

$= \mathbb{E}_x[\nabla_\theta(\log P(x|\theta)) f(x)]$

- Monte Carlo Policy Gradient →

Initialize θ Arbitrarily.

For each episode $\tau = S_0, A_0, R_1, S_1, \ldots, S_t$

For $t = 1 : T-1$:

$\Delta\theta = \alpha \nabla_\theta(\log \pi(S_t, A_t, \theta)) G_t$

$\theta = \theta + \Delta\theta$

← Algorithm Reinforce.

- Contrastive Policy Gradient →

- Constrained Policy Gradients →

* With policy gradient, we focus on the overall performance based on the reward, disregarding state, action → This could lead to two different policies A, B with similar rewards and really different behaviors.

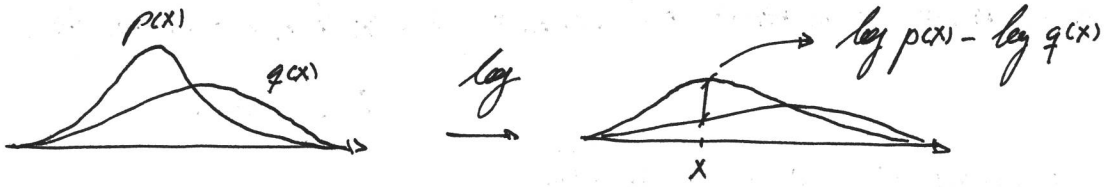
$$J_\theta = \mathbb{E}_\pi [R(\tau)] - \beta D[\pi(\cdot,\cdot,\theta) - \pi(\cdot,\cdot,\theta')]$$

* Difference between policies → $D[\pi(\cdot,\cdot,\theta) - \pi(\cdot,\cdot,\theta')] \leq \delta$

  - Stabilizes the algorithm
  - Prevents to have big changes in behavior

* KL - Divergence → Policy as probability function, how different?

$$D_{KL}(p\|q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx = \int_{-\infty}^{\infty} p(x)[\log p(x) - \log q(x)] dx$$



$$D_{KL}(p\|q) \neq D_{KL}(q\|p)$$

$$J(\theta) = \mathbb{E}_\pi [R(\tau)] - \beta D(\pi(\cdot,\cdot,\theta), \pi(\cdot,\cdot,\theta'))$$

$$J(\theta) = \mathbb{E}_\pi [R(\tau)] - \beta D_{KL}(\pi(\cdot,\cdot,\theta) \| \pi(\cdot,\cdot,\theta'))$$

Actor - Critic Methods

- Score function ⟶

  * We have a way to improve our policy, but we need a way to keep track how we are doing.

  * Non episodic environment, something that we can compute as it interacts with the environment.

$$\Delta\theta = \alpha \nabla_\theta (\log \pi(S_t, A_t, \theta))\, R(\tau)$$

$$R(\tau) = G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

↓

Policy Update

$$\Delta\theta = \alpha \nabla_\theta (\log \pi(S_t, A_t, \theta))\, Q(S_t, A_t)$$

Value Update

$$Q(S_t, A_t) \longleftarrow Q(S_t, A_t) + \beta \underbrace{(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))}_{\text{TD Difference.}}$$

  * $\beta \equiv$ learning rate for value updates.

  * This two can run in parallel and they don't need an entire episode to be updated.

- Two function approximation ⟶

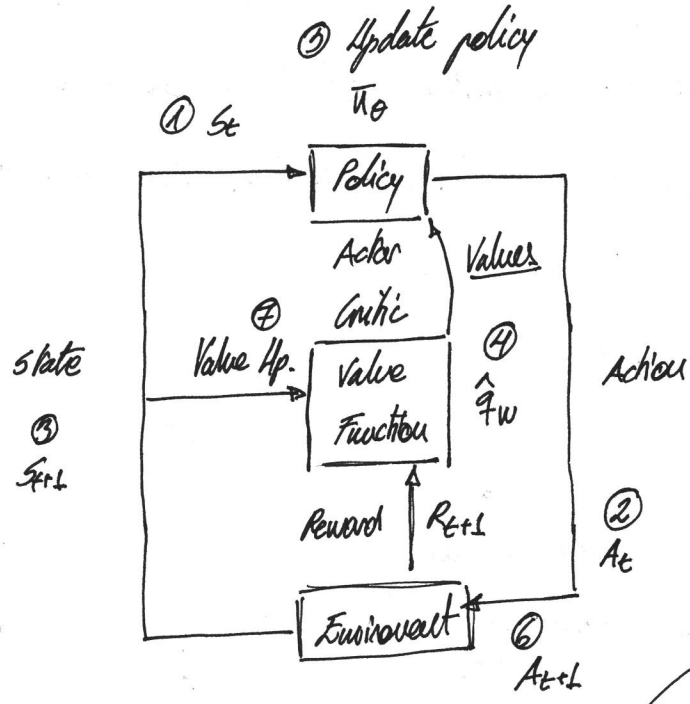
* Q-learning with function approximation

$$\Delta\theta = \alpha \nabla_\theta \left(\log \pi(S_t, A_t, \theta) \right) \hat{q}(S_t, A_t, w)$$ Policy

$$\Delta w = \beta \left(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, w) - \hat{q}(S_t, A_t, w) \right) \nabla_w \hat{q}(S_t, A_t, w)$$ Value

$\pi(S, a, \theta)$ ⟶ Actor ; probability of taking an action in state.

$\hat{q}(S, a, w)$ ⟶ Critic ; expected return in that state-action.



Deep Deterministic Policy Gradient.

⑤ Policy Update ⟶ $\Delta\theta = \alpha \nabla_\theta \left(\log \pi(S_t, A_t, \theta) \right) \hat{q}(S_t, A_t, w)$ ← Action gradient in implementation.

⑦ Value Update ⟶ $\Delta w = \beta \left[\underbrace{R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, w)}_{\text{Target}} - \hat{q}(S_t, A_t, w) \right] \nabla_w \hat{q}(S_t, A_t, w)$

- Advantage Function ⟶

* Policy gradient ⟶ $\Delta\theta = \alpha \nabla_\theta J(\theta)$

* Gradient ⟶

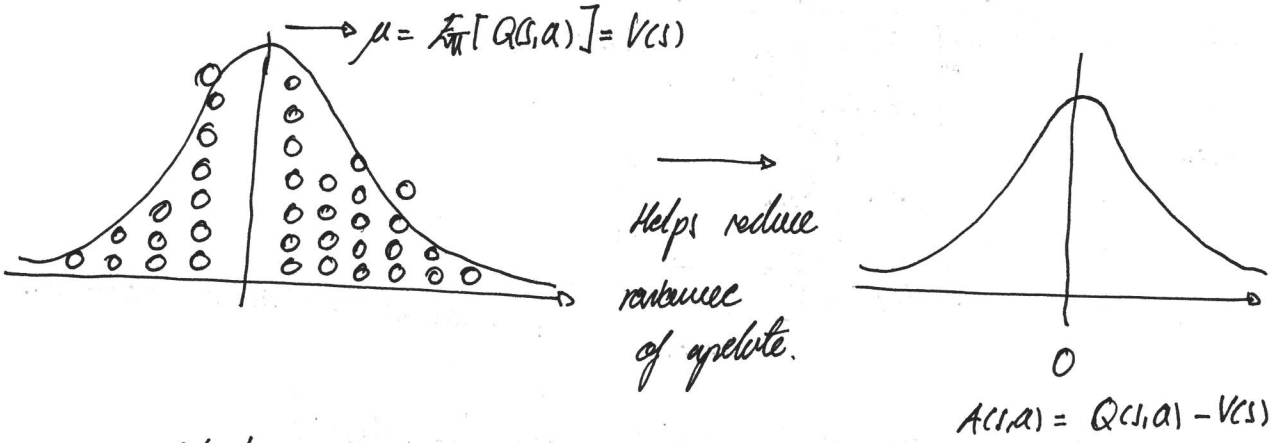
$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[\nabla_\theta(\log \pi(s,a,\theta))\, R(\tau)] = \mathbb{E}_\pi[\nabla_\theta(\log \pi(s,a,\theta)\, \hat{q}(s,a,w)]$$

Approximate the expectation by taking small steps.

$$\Delta\theta = \alpha \nabla_\theta(\log \pi(s,a,\theta))\, \hat{q}(s,a,w)$$

* Need to ensure small step ⟶ working with stochastic values. They can vary a lot.

* With the expectation, there is also a variance of the samples ⟶
Given by the action value function estimation, since each time step vary.



* Intuition ⟶
How much reward beyond the expected we get with the action.

- Actor-Critic with Advantage ⟶

* Gradient ⟶ $\nabla_\theta J(\theta) = \mathbb{E}_\pi\left[\nabla_\theta(\log \pi(s,a,\theta))\,(\hat{q}(s,a,w) - \hat{v}(s,a,w)\right]$

* Policy update ⟶

$$\Delta\theta = \alpha\,\nabla_\theta\left[\log \pi(s,a,\theta))\,(\hat{q}(s,a,w) - \hat{v}(s,w)\right]$$

* Use the TD error δ as estimation of the advantage function ⟶

$$\text{TD error } \delta = r + \gamma\hat{v}(s',w) - \hat{v}(s,w) \approx \hat{q}(s,a,w) - \hat{v}(s,w)$$

$$= R_{t+1} + \gamma\hat{V}(s_{t+1},w) - \hat{V}(s_t,w)$$