- "world" / environment where our agent acts, is characterized by its state S_t at time t, states evolve in discrete time steps $S_t \to S_{t+1}$
- behavior of the "world" is fully characterized by the transition probability given current state $S_t = s$ and action $a_t = a$, what's the probability to get to state $S_{t+1} = s'$ and receive reward $R_{t+1} = r$?

$$p(s', r \mid s, a)$$

important: this is stationary, doesn't depend in which time step we reached s (Markov property)

- behavior of the agent is determined by its policy $\pi(a \mid s)$: probability to choose amongst possible action a (hopefully good ones) in state s
- if world and/or agent behave <u>deterministically</u>, $p(s', r \mid s, a)$ and/or $\pi(a|s)$ reduce to delta-distributions
- value function of a state s for given policy π expected ~~reward~~ total reward when the game starts in state s and agent follows the policy π:

$$V_\pi(s) = \mathbb{E}_{p,\pi}\left[\sum_{t'=t+1}^{\infty} \gamma^{t'-t-1} R_{t'} \mid S_t = s\right]$$

- self-consistence of values leads to <u>Bellmann equations</u>

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

- the optimal value function chooses best action: $\boxed{V^*(s) = \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V^*(s')\right]}$

- determine optimal value by <u>value iteration</u>:

  ⓪ initial guess $V^{(0)}(s)$ (e.g. $V^{(0)}(s) = 0$)

  ① for $\tau = 1, 2, \ldots$ (or until convergence)

  $$\forall s: \quad V^{(\tau)}(s) = \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V^{(\tau-1)}(s')\right]$$

  converges to a fixed point (global optimum ?)

- once we know the optimal value of each state, we can derive optimal policy:

$$\boxed{\pi^*(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V^*(s')\right]}$$

<u>Model-free RL methods</u>

- problem so far: to determine $V^*(s)$ and $\pi^*(s)$, we need to know the transition probabilities $p(s', r \mid s, a)$
- usually, we don't know them $\Rightarrow$ <u>model-based RL</u> $\overset{?}{=}$ first learn $p(s', r \mid s, a)$ ("world model") and derive $V^*(s)$, $\pi^*(s)$ by value iteration

  <u>adv:</u> understand the "world" <u>disadv:</u> very difficult
- <u>model-free RL</u> learns good policy without fully understanding the "world"

  <u>adv</u>: easier <u>disadv.</u>: black-box solutions that are hard to interpret
- meta-algorithm: <u>repeat many times</u>: (a) play according to current policy

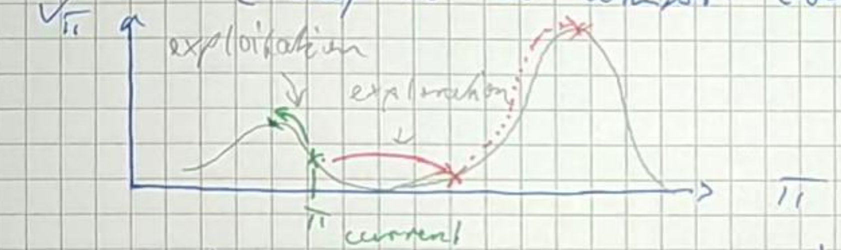
  (b) collect reward data

  (c) use data to improve policy

⇒ simultaneous learning of policy and (implicitly) world behavior

<u>exploration – exploitation trade-off</u>

- exploration: learn more about the world
- exploitation: fine-tune the current policy and maximize reward
  (stay in the comfort zone of your current understanding)



- common solution: <u>ε-greedy policy</u>: hyperparameter ε (~~annected~~ annealed during training)
  - with probability $(1-\varepsilon)$: execute the best action according to your current policy   exploit
  - — " — ε: execute random action   explore

  → guarantees that after infinitely many trials, every combination of state s and action a is tried infinitely often ≙ have seen all world

<u>Monte-Carlo (MC) value estimation</u>

- choose action according to ε-greedy policy and estimate values by averaging actual rewards: - playing the game once ≙ "episode" ⇒ episode index $\tau = 1, 2, \ldots$
  - doing a move in game τ: index $t = 1, 2, \ldots T_\tau$

    T_τ: duration of episode τ

return of step t in game τ = total reward of the rest of the game τ: $G_{\tau t} = \sum_{t'=t+1}^{T_\tau} \gamma^{t'-t-1} \underbrace{R_{\tau t'}}_{\text{actual rewards}}$

- determine value $V_\pi(s)$ as the average over all games where we encountered state s

$$V_\pi(s) \approx \frac{1}{N_s} \sum_{\tau, t:\ s_{\tau t} = s} G_{\tau t}$$

N_s: how often we saw state s

can be computed incrementally: update $V_\pi(s)$ whenever we come to state s:

$$N_s \leftarrow N_s + 1$$

new value, old value

$$V_\pi(s) \leftarrow V_\pi(s) + \frac{1}{N_s} \underbrace{\left(G_{\tau t} - V_\pi(s) \right)}_{\text{correction of current guess } V_\pi(s)}$$

generalize to arbitrary learning rate α

$$\boxed{V_\pi(s) \leftarrow V_\pi(s) + \alpha \left(G_{\tau t} - V_\pi(s) \right)}$$

advantage: this also works when the policy π is not constant over training (we need this, because π should improve!)

disadvantage: slow

Temporal Difference (TD) value estimation

- computing $G_{\tau t}$ is inconvenient, because one must wait to the end of episode τ before $G_{\tau t}$ is determined, immediate updates after each move are better

$\Rightarrow$ approximation: $$G_{\tau t} \approx \underbrace{R_{\tau, t+1}}_{\text{actual reward}} + \gamma \underbrace{V_\pi(s_{\tau, t+1})}_{\text{current guess for value of the rest}}$$

$$\boxed{V_\pi(s_{\tau t}) \leftarrow V_\pi(s_{\tau t}) + \alpha \left[R_{\tau, t+1} + \gamma V_\pi(s_{\tau, t+1}) - V_\pi(s_t) \right]}$$

- more accurate variant: <u>n-step TD value estimation</u>

approximate $G_{\tau t}$ with real rewards for the next n moves (plain TD: $n=1$)

$$G_{\tau t} \approx R_{\tau,t+1} + \gamma R_{\tau,t+2} + \ldots + \gamma^{n-1} R_{\tau,t+n} + \gamma^n V_\pi(s_{\tau,t+n+1})$$

update rule
$$\boxed{V_\pi(s_{\tau,t}) \leftarrow V_\pi(s_{\tau,t}) + \alpha \left[\sum_{t'=t+1}^{t+n} \gamma^{t'-t-1} R_{\tau,t'} + \gamma^n V_\pi(s_{\tau,t+n+1}) - V_\pi(s_{\tau,t}) \right]}$$

main advantage: actually observed return $R_{\tau,t}$ is used in n updates

$\Rightarrow$ make better use of our data

- once we have determined the value of the current policy, we can use this to improve the policy
$$\pi'(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r \mid s,a) \left[r + \gamma V_\pi(s) \right]$$

<u>but</u>: we need to know the transition probabilities

<u>Q</u>: can we also do this in a model-free setting?

Q - learning

- define "action-value function" $Q_{\pi}(s,a)$ ("Q-function")
- remember Bellman eq. for value function: $v_{\pi}(s) = \sum_a \pi(a|s) \underbrace{\sum_{s',r} p(s',r|s,a)\left[r + \gamma v_{\pi}(s')\right]}_{= Q_{\pi}(s,a)}$

$$\boxed{Q_{\pi}(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_{\pi}(s')\right]}$$

$$v_{\pi}(s) = \sum_a \pi(a|s)\, Q_{\pi}(s,a)$$

interpretation: value of state s, when we continue with action a, and then follow policy π for the rest of the game (= ignore policy in first move)

$\Rightarrow$ optimal policy chooses the action that maximizes action-value

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

$$Q^*(s) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \underbrace{v^*(s')}_{v_{\pi^*}(s') = \max_a Q^*(s',a)}\right]$$

Bellman equations for Q-functions

$$\boxed{Q^*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_a Q^*(s',a)\right]}$$

same holds for sub-optimal policies

$\Rightarrow$ once we know $Q_{\pi}(s,a)$ for old policy, we get improved policy without knowing transition prob.

(guaranteed: π' is not worse than π)

$$\boxed{\forall s: \ \pi'(s) = \arg\max_a Q_{\pi}(s,a)}$$

Q-learning algorithm

(0) initialize $Q^{(0)}(s,a)$ arbitrary (good guess ⇒ faster convergence)

but make sure that ~~$Q^{(0)}(s,a) = 0$~~ $Q^{(0)}(s,a) = 0$ for all terminal states

initial policy $\pi^{(0)}(s) = \arg\max_a Q^{(0)}(s,a)$

(1) for episodes $\tau = 1, 2, \dots$ (until convergence)

(a) play episode according to ε-greedy policy:

$$a_{\tau,t} \sim \begin{cases} \arg\max_a Q^{(\tau-1)}(s_{\tau t}, a) & \text{with prob. } 1-\varepsilon \\ \text{uniform}(a) & \text{with prob } \varepsilon \end{cases}$$

[alternative exploration - exploitation trade-off: softmax policy

$$a_{\tau,t} \sim \frac{e^{Q^{(\tau-1)}(s_{\tau,t}, a)/\varrho}}{\sum_{a'} e^{Q^{(\tau-1)}(s_{\tau,t}, a')/\varrho}}$$

softmax function for vector v

$$\frac{\exp(v_i/\varrho)}{\sum_{i'} \exp(v_{i'}/\varrho)}$$

role of temperature ϱ: $\varrho \to 0$: always choose $a = \arg\max_{a'} Q(s,a')$

$\varrho \to \infty$: $a \sim \text{uniform}(a')$

anneal ϱ as τ increases]

⇒ game outcome $s_{\tau,0}, a_{\tau 0}, r_{\tau 1}, s_{\tau 1}, a_{\tau 1}, r_{\tau 2}, \dots r_{\tau T_\tau}$

(b) update Q-function: for $t = 1, \dots T_\tau$

$$\boxed{Q^{(\tau+1)}(s_{\tau,t-1}, a_{\tau,t-1}) \leftarrow Q^{(\tau)}(s_{\tau,t-1}, a_{\tau,t-1}) + \alpha\left[r_{\tau t} + \gamma V^{(\tau)}(s_{\tau,t}) - Q^{(\tau)}(s_{\tau,t-1}, a_{\tau,t-1})\right]}$$

two variants according to how we define the value $V^{(\tau)}(s_{\tau,t})$ of the rest of the game:

- $V^{(\tau)}(s_{\tau t}) = \max_a Q^{(\tau)}(s_{\tau,t}, a)$ "Q-learning"
- $V^{(\tau)}(s_{\tau t}) = Q^{(\tau)}(s_{\tau t}, a_{\tau t})$ "SARSA algorithm"

(2) update policy: $\boxed{\forall s: \hat{\pi}(s) = \arg\max_a Q^{(\tau_{end})}(s, a)}$ find policy

[variant: n-step Q-learning

$$Q(s_{\tau,t-1}, a_{\tau,t-1}) \leftarrow Q(s_{\tau,t-1}, a_{\tau,t-1}) + \alpha\left[\sum_{t'=t}^{t+n-1} \gamma^{t'-t} r_{t'} + \gamma^n V(s_{\tau,t+n}) - Q(s_{\tau,t-1}, a_{\tau,t-1})\right]$$

advantages of Q-learning: - can update policy without knowing $p(s', r \mid s, a)$
"model-free method"

- supports <u>off-policy learning</u>: we can mix game outcomes obtained with different policies in updating Q
(at the beginning of training, policy is bad, but we can still use game outcomes in the updates later on)
$\Rightarrow$ learning with <u>experience buffer</u>