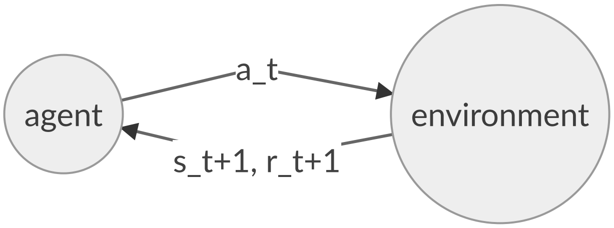
RL Cheatsheet

Basic Concept



Action

$$a \in \mathbb{R}^{d_a}$$

Under the RL framework, action is the only thing that the agent has control over. The agent can choose to act randomly, disregarding the environment reward and state, or to act greedily and maximize the reward from next step. Action is how the agent responds to the environment.
Action could be a real value or a vector. The set of all actions the agent can make are called *action space*.

State

$$s \in \mathbb{R}^{d_s}$$

State is what completely characterizes the current environment. It lies entirely out of control of the agent.
Like action, state could be a real value or a vector. The set of all possible states the environment can be in are called *state space*.

Reward

$$r \in \mathbb{R}$$

Reward is the key feedback from environment to agent. It is what drives the agent to learn and to change its future action. Think about reward as training data in a supervised learning framework.
Unlike action and state, reward is always a single value.

Basic Relationship

$$a_t \sim \pi(s_t)$$

What does the agent's action depend on? The agent acts based on its observation from the environment, which is the state. Reward changes how the agent acts in future, but does not affect how the agent makes current action.

The function π is called *action policy*. It characterizes how the agent reacts to environment. We use $\sim$ instead of $=$ here because the action policy returns a probability distribution. For example, under ϵ-greedy policy, the agent chooses a random action ϵ percentage of the time.

RL is framed as a dynamic process. We refer to concept at a specific time step with subscript t.

$$s_{t+1} \sim f(s_t, a_t)$$

The next state the environment will be in naturally depends on the current state s_t. In addition, it depends on agent's action a_t with the environment. The environment could behave with certain randomness rather than deterministically, so $f(s_t, a_t)$ is a probability distribution.

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

Reward captures how happy/unhappy the environment is with the change. To capture such change, we need all previous state s_t, current state s_{t+1} and previous action a_t by agent.

Because we can model the current state using $s_{t+1} \sim f(s_t, a_t)$, we say the reward is a function of just s_t and s_{t+1}.

Return

Trajectory

$$(s_0, a_0, s_1, a_1, ..., s_T)$$

Trajectory is a sequence of states and actions that fully characterizes the entire RL process. A terminal state s_T exists only for an episodic process; otherwise the trajectory goes on forever. Notice that reward is not part of the trajectory. Think about reward as an arbitrarily imposed signal to change how the agent chooses its action, but it does not contain any information about the trajectory itself.

Return

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

Return is the accumulative reward over an entire trajectory, with a discount factor γ. Without γ, return is simply a sum of all rewards the agent collects. The discount factor γ serves to emphasize the short term reward over reward in the future. At the same time, it's also mathematically convenient to have it so that the return doesn't go to infinity.

Value function

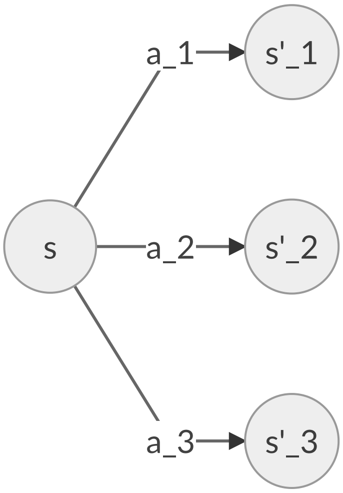
$$V^\pi(s) = \mathbf{E}_{\tau \sim \pi}[R(\tau) | s_0 = s]$$

We are often interested in the return $R(\tau)$ from a given starting state. We call this expected value *value function*. As the *value function* concerns the entire trajectory starting from s, it is necessarily dependent on the *action policy*, which governs how the agent responds to environment states and thus the entire trajectory. We use $\tau \sim \pi$ in the expected value to indicate the trajectory is determined by the *action policy*.

Action value function

$$Q^{\pi}(s,a) = \mathbf{E}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a]$$

A similar concept to *value function*, *action value function* models the expected return starting from a given state and the agent making a particular action. Given a starting state s, *value function* models when the agent makes its first action based on the action policy $a \sim \pi(s)$, while *action value function* models when the first action is exactly as given.



Consider the above state transition from s to s', where the agent can choose an action from a_1, a_2, a_3. *Action value function* is conditioned on one chosen action. *Value function*, on the other hand, models the expected value of all possible *action value function* as long as the action is sampled from agent's *action policy*.

$$V^{\pi}(s) = \mathbf{E}_{a\sim\pi(s)}[Q^{\pi}(s,a)]$$

Optimal Return

Reward/return is what stimulates the agent to learn. In other words, the agent tries to act such that the overall return is maximized. We are interested in the maximal return achievable from a given starting state, with a given action in the first time step.

Optimal value function

$$V^{*}(s) = \max_{\pi} \mathbf{E}_{\tau\sim\pi}[R(\tau)|s_0 = s]$$

Optimal value function is the *value function* when the best *action policy* is used. Here, the term *best* is defined as the one that gives the highest *value function*. We use $*$ to indicate such an optimal *action policy* in place of π.

Optimal action value function

$$Q^*(s,a) = \max_{\pi} \mathbf{E}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a]$$

Similarly, *optimal action value function* is the *action value function* when the best *action policy* is used. Because the first action is explicitly given as a, the optimal action policy is only used starting from the second action.

Note that unlike definitions for *value function* and *action value function*, the above two optimal functions are independent of the actual action policy. They simply care about the optimal return achievable from a starting state (and action), regardless of the policy being used. In the Q-learning algorithm, Q stands for the *optimal* action value function. With the modeled optimal Q function, we can conveniently obtain our optimal action at a given state.

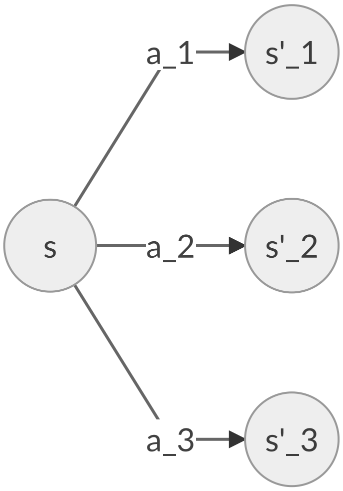
Optimal Action

$$a^*(s) = \arg\max_{a} Q^*(s,a)$$

If we have the optimal action value function Q^* at state s, then our optimal action a^* is just the action that yields the highest Q^* value. Note that this optimal action is non-greedy: it doesn’t simply try to maximize the reward from next state, but optimizes the total return over the entire trajectory, taking into account all future steps.

One could say that the above optimal action is one possible action policy for the agent.

Bellman Equation



Bellman equation, value function

$$V^{\pi}(s) = \mathbf{E}_{a\sim\pi(s),s'\sim f(s,a)}[r(s,a) + \gamma V^{\pi}(s')]$$

γ is the discounting factor, π is our action policy, $f(s,a)$ models the environment and s' is the next state.

To calculate expected return of current state $V^{\pi}(s)$, we see that it should be a sum of reward from current step and expected return from the next state with discount factor γ. Reward from current step depends on the action taken, modeled by $a \sim \pi(s)$, and expected return from next state is simply $V^{\pi}(s')$, where s' represents the next state. To get s', recall that our environment models the next state as $s' \sim f(s,a)$.

Bellman equation, action value function

$$Q^{\pi}(s,a) = r(s,a) + \gamma\mathbf{E}_{a'\sim\pi(s'),s'\sim f(s,a)}[Q^{\pi}(s',a')]$$

a' is the next action sampled from action policy $\pi(s')$.

Expected return of current state-action pair $Q^{\pi}(s,a)$ is similarly calculated as the sum of current reward $r(s,a)$ and value function from next state-action pair $Q^{\pi}(s',a')$. Since the action for the current step is given as a, we don't need to choose it based on policy π. However in this case, since we are using both state and action from the next step, we will need to first model the next state, $s' \sim f(s,a)$, and then samples the next action as $a' \sim \pi(s')$.

Bellman equation characterizes the relationship between value function from current state and that from next state. By chaining value functions of connecting states, Bellman equation allows to model each state's expected return based on next state's, eventually stopping at the terminal state s_T. This process is reminiscent of a dynamic programming algorithm.

Optimal Bellman Equation

Optimal Bellman equations are Bellman equations when the optimal action is taken. In other words, we simply need to choose the best action at a given state to reach the optimal Bellman equations.

Optimal Bellman equation, value function

$$V^*(s) = \max_a \mathbf{E}_{s' \sim f(s,a)}[r(s,a) + \gamma V^*(s')]$$

Note that instead of sampling the action a from policy $\pi(s)$, optimal Bellman equation specifies that the best action is chosen at current state s. The action that achieves the optimal Bellman equation is also our optimal action.

Optimal Bellman equation, action value function

$$Q^*(s,a) = r(s,a) + \gamma \max_{a'} \mathbf{E}_{s' \sim f(s,a)}[Q^*(s',a')]$$

Here, the best action a' from next state s' is required.

Just like we are more interested in the optimal action value function $Q^*(s,a)$ than an arbitrary action value function with a given policy $Q^\pi(s,a)$, for Bellman equations we are also primarily interested in the optimal scenarios. Optimal Bellman equations describe the highest possible return from any given state/state-action pair. Most importantly, it makes the fascinating observation that, as long as the agent chooses the greedy action at either current (for value function) or next (for action value function) state, with respect to the optimal action value function Q^*, the agent will achieve the globally optimal return.

Temporal Difference Learning

Temporal difference (TD) learning is a method to improve the current state estimate using estimates of future states. The underlying assumption is that by having some knowledge about the state transition, which is the reward r from the current to next state, we can have a better estimation of the current state return.

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha(r + \gamma V^\pi(s') - V^\pi(s))$$

Here, s is the current state and s' is the next state.

TD Estimate

We refer to the term $V^\pi(s)$ as the *TD Estimate*. This is the estimated value function of current state.

TD Target

We refer to the term $r + \gamma V^\pi(s')$ as the *TD Target*. This is the target value function that we want to optimize *TD Estimate* towards to. *TD Target* is calculated based on value estimate of next state and current reward. By incorporating the additional information r, *TD Target* is a more accurate estimate than *TD Estimate*. Drawing an analogy to supervised learning, $r + \gamma V^\pi(s')$ is assumed to be the *label*, and $V^\pi(s)$ is the *prediction*.

TD learning could be applied for both value function $V^\pi(s)$ and value action function $Q^\pi(s, a)$. In the case for $Q^\pi(s, a)$, TD learning goes as following.

$$Q^\pi(s, a) \leftarrow Q^\pi(s, a) + \alpha(r + \gamma Q^\pi(s', a') - Q^\pi(s, a))$$

Q-learning

Q-learning is the result of combining TD learning with optimal Bellman equation. With the philosophy of TD learning, Q-learning builds a better estimate of current state-action pair by using reward information and optimal estimate for the next state-action pair.

$$Q^*(s,a) \leftarrow Q^*(s,a) + \alpha(r + \gamma \max_{a'} Q^*(s',a') - Q^*(s,a))$$

Note that the term $r + \gamma \max_{a'} Q^*(s',a')$ is exactly what optimal Bellman equation gives us as the proxy for $Q^*(s,a)$. Under TD learning, this proxy gives a better estimate of current state-action pair, so we want to update $Q^*(s,a)$ towards it. In other words, we want to minimize the difference between current estimate $Q^*(s,a)$ and *TD target* $r + \gamma \max_{a'} Q^*(s',a')$.

$$L(\theta) = (r + \gamma \max_{a'} Q^*_\theta(s',a') - Q^*_\theta(s,a))^2$$

As we frequently use a neural network for the optimal action value function $Q^*(s',a')$, θ here represents the neural network parameters. The goal is to minimize the loss $L(\theta)$ with respect to θ so that $Q^*_\theta(s,a)$ approaches $r + \gamma \max_{a'} Q^*_\theta(s',a')$.