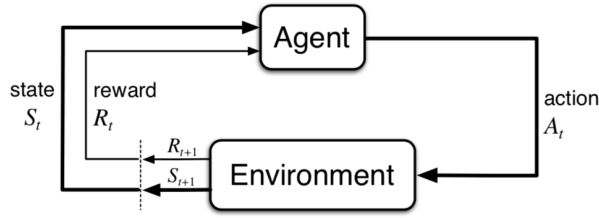


# Reinforcement Learning Cheat Sheet

## Agent-Environment Interface



The Agent at each step  $t$  receives a representation of the environment's *state*,  $S_t \in S$  and it selects an action  $A_t \in A(s)$ . Then, as a consequence of its action the agent receives a *reward*,  $R_{t+1} \in R \in \mathbb{R}$ .

## Policy

A *policy* is a mapping from a state to an action

$$\pi_t(s|a) \quad (1)$$

That is the probability of select an action  $A_t = a$  if  $S_t = s$ .

## Reward

The total *reward* is expressed as:

$$G_t = \sum_{k=0}^H \gamma^k r_{t+k+1} \quad (2)$$

Where  $\gamma$  is the *discount factor* and  $H$  is the *horizon*, that can be infinite.

## Markov Decision Process

A **Markov Decision Process**, MPD, is a 5-tuple  $(S, A, P, R, \gamma)$  where:

finite set of states:

$s \in S$

finite set of actions:

$a \in A$

state transition probabilities:

$p(s'|s, a) = \Pr\{S_{t+1} = s' | S_t = s, A_t = a\}$

expected reward for state-action-nexstate:

$r(s', s, a) = \mathbb{E}[R_{t+1} | S_{t+1} = s', S_t = s, A_t = a]$

(3)

## Value Function

Value function describes *how good* is to be in a specific state  $s$  under a certain policy  $\pi$ . For MDP:

$$V_\pi(s) = \mathbb{E}[G_t | S_t = s] \quad (4)$$

Informally, is the expected return (expected cumulative discounted reward) when starting from  $s$  and following  $\pi$

## Optimal

$$v_*(s) = \max_{\pi} v^{\pi}(s) \quad (5)$$

## Action-Value (Q) Function

We can also denoted the expected reward for state, action pairs.

$$q_{\pi}(s, a) = \mathbb{E}_{\pi} [G_t | S_t = s, A_t = a] \quad (6)$$

## Optimal

The optimal value-action function:

$$q_*(s, a) = \max_{\pi} q^{\pi}(s, a) \quad (7)$$

Clearly, using this new notation we can redefine  $v^*$ , equation 5, using  $q^*(s, a)$ , equation 7:

$$v_*(s) = \max_{a \in A(s)} q_{\pi^*}(s, a) \quad (8)$$

Intuitively, the above equation express the fact that the value of a state under the optimal policy **must be equal** to the expected return from the best action from that state.

## Bellman Equation

An important recursive property emerges for both Value (4) and Q (6) functions if we expand them.

## Value Function

$$\begin{aligned}
 v_{\pi}(s) &= \mathbb{E}_{\pi} [G_t | S_t = s] \\
 &= \mathbb{E}_{\pi} \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right] \\
 &= \mathbb{E}_{\pi} \left[ R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_t = s \right] \\
 &= \underbrace{\sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)}_{\text{Sum of all probabilities } \forall \text{ possible } r} \left[ r + \gamma \underbrace{\mathbb{E}_{\pi} \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_{t+1} = s' \right]}_{\text{Expected reward from } s_{t+1}} \right] \\
 &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) [r + \gamma v_{\pi}(s')]
 \end{aligned} \quad (9)$$

Similarly, we can do the same for the Q function:

$$\begin{aligned}
 q_{\pi}(s, a) &= \mathbb{E}_{\pi} [G_t | S_t = s, A_t = a] \\
 &= \mathbb{E}_{\pi} \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right] \\
 &= \mathbb{E}_{\pi} \left[ R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_t = s, A_t = a \right] \\
 &= \sum_{s', r} p(s', r|s, a) \left[ r + \gamma \mathbb{E}_{\pi} \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_{t+1} = s' \right] \right] \\
 &= \sum_{s', r} p(s', r|s, a) [r + \gamma V_{\pi}(s')]
 \end{aligned} \quad (10)$$

## Dynamic Programming

Taking advantages of the subproblem structure of the V and Q function we can find the optimal policy by just *planning*

## Policy Iteration

We can now find the optimal policy

1. Initialisation

$V(s) \in \mathbb{R}$ , (e.g  $V(s) = 0$ ) and  $\pi(s) \in A$  for all  $s \in S$ ,  $\Delta \leftarrow 0$

2. Policy Evaluation

**while**  $\Delta \geq \theta$  (a small positive number) **do**

**foreach**  $s \in S$  **do**

$v \leftarrow V(s)$

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

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

**end**

**end**

3. Policy Improvement

*policy-stable*  $\leftarrow$  *true*

**foreach**  $s \in S$  **do**

$old-action \leftarrow \pi(s)$

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

*policy-stable*  $\leftarrow$  *old-action*  $= \pi(s)$

**end**

**if** *policy-stable* **return**  $V \approx v_*$  and  $\pi \approx \pi_*$ , **else go to 2**

**Algorithm 1:** Policy Iteration

## Value Iteration

We can avoid to wait until  $V(s)$  has converged and instead do policy improvement and truncated policy evaluation step in one operation

```

Initialise  $V(s) \in \mathbb{R}$ , e.g.  $V(s) = 0$ 
 $\Delta \leftarrow 0$ 
while  $\Delta \geq \theta$  (a small positive number) do
    foreach  $s \in S$  do
         $v \leftarrow V(s)$ 
         $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$ 
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
    end
end
output: Deterministic policy  $\pi \approx \pi_*$  such that
 $\pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$ 

```

**Algorithm 2:** Value Iteration

## Monte Carlo Methods

Monte Carlo (MC) is a *Model Free* method, It does not require complete knowledge of the environment. It is based on **averaging sample returns** for each state-action pair. The following algorithm gives the basic implementation

```

Initialise for all  $s \in S, a \in A(s)$  :
     $Q(s, a) \leftarrow$  arbitrary
     $\pi(s) \leftarrow$  arbitrary
     $Returns(s, a) \leftarrow$  empty list
while forever do
    Choose  $S_0 \in S$  and  $A_0 \in A(S_0)$ , all pairs have
    probability  $> 0$ 
    Generate an episode starting at  $S_0, A_0$  following  $\pi$ 
    foreach pair  $s, a$  appearing in the episode do
         $G \leftarrow$  return following the first occurrence of  $s, a$ 
        Append  $G$  to  $Returns(s, a)$ 
         $Q(s, a) \leftarrow average(Returns(s, a))$ 
    end
    foreach  $s$  in the episode do
         $\pi(s) \leftarrow \arg\max_a Q(s, a)$ 
    end
end

```

**Algorithm 3:** Monte Carlo first-visit

For non-stationary problems, the Monte Carlo estimate for, e.g.,  $V$  is:

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)] \quad (11)$$

Where  $\alpha$  is the learning rate, how much we want to forget about past experiences.

## Sarsa

Sarsa (State-action-reward-state-action) is a on-policy TD control. The update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

### $n$ -step Sarsa

Define the  $n$ -step Q-Return

$$q_t^{(n)} = R_{t+1} + \gamma R_t + 2 + \dots + \gamma^{n-1} R_{t+n} + \gamma^n Q(s_{t+n})$$

$n$ -step Sarsa update  $Q(S, a)$  towards the  $n$ -step Q-return

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [q_t^{(n)} - Q(s_t, a_t)]$$

### Forward View Sarsa( $\lambda$ )

$$q_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} q_t^{(n)}$$

Forward-view Sarsa( $\lambda$ ):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [q_t^\lambda - Q(s_t, a_t)]$$

```

Initialise  $Q(s, a)$  arbitrarily and
 $Q(terminal - state, ) = 0$ 
foreach episode  $\in episodes$  do
    Choose  $a$  from  $s$  using policy derived from  $Q$  (e.g.,
     $\epsilon$ -greedy)
    while  $s$  is not terminal do
        Take action  $a$ , observer  $r, s'$ 
        Choose  $a'$  from  $s'$  using policy derived from  $Q$ 
        (e.g.,  $\epsilon$ -greedy)
         $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma Q(s', a') - Q(s, a)]$ 
         $s \leftarrow s'$ 
         $a \leftarrow a'$ 
    end
end

```

**Algorithm 4:** Sarsa( $\lambda$ )

## Temporal Difference - Q Learning

Temporal Difference (TD) methods learn directly from raw experience without a model of the environment's dynamics. TD substitutes the expected discounted reward  $G_t$  from the episode with an estimation:

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \quad (12)$$

The following algorithm gives a generic implementation.

```

Initialise  $Q(s, a)$  arbitrarily and
 $Q(terminal - state, ) = 0$ 
foreach episode  $\in episodes$  do
    while  $s$  is not terminal do
        Choose  $a$  from  $s$  using policy derived from  $Q$ 
        (e.g.,  $\epsilon$ -greedy)
        Take action  $a$ , observer  $r, s'$ 
         $Q(s, a) \leftarrow$ 
         $Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ 
         $s \leftarrow s'$ 
    end
end

```

**Algorithm 5:** Q Learning

## Deep Q Learning

Created by *DeepMind*, Deep Q Learning, DQL, substitutes the  $Q$  function with a deep neural network called *Q-network*. It also keep track of some observation in a *memory* in order to use them to train the network.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[ \underbrace{(r + \gamma \max_a Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i))^2}_{\text{target}} \underbrace{- Q(s, a; \theta_i)}_{\text{prediction}} \right] \quad (13)$$

Where  $\theta$  are the weights of the network and  $U(D)$  is the experience replay history.

```

Initialise replay memory  $D$  with capacity  $N$ 
Initialise  $Q(s, a)$  arbitrarily
foreach episode  $\in episodes$  do
    while  $s$  is not terminal do
        With probability  $\epsilon$  select a random action
         $a \in A(s)$ 
        otherwise select  $a = \max_a Q(s, a; \theta)$ 
        Take action  $a$ , observer  $r, s'$ 
        Store transition  $(s, a, r, s')$  in  $D$ 
        Sample random minibatch of transitions
         $(s_j, a_j, r_j, s'_j)$  from  $D$ 
        Set  $y_j \leftarrow$ 
         $\begin{cases} r_j & \text{for terminal } s'_j \\ r_j + \gamma \max_a Q(s', a'; \theta) & \text{for non-terminal } s'_j \end{cases}$ 
        Perform gradient descent step on
         $(y_j - Q(s_j, a_j; \Theta))^2$ 
         $s \leftarrow s'$ 
    end
end

```

**Algorithm 6:** Deep Q Learning

Copyright © 2018 Francesco Saverio Zupichini  
<https://github.com/FrancescoSaverioZupichini/Reinforcement-Learning-Cheat-Sheet>

## Double Deep Q Learning