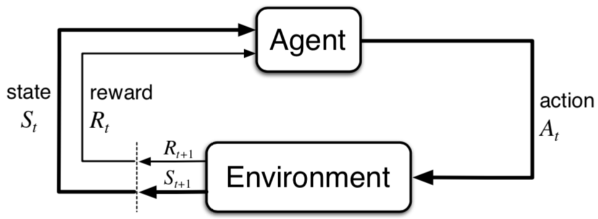
Reinforcement Learning Cheat Sheet

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

$$\pi(s|a) \tag{1}$$

That is the probability of select an action $A_t = a$ if $S_t = s$. also policy could be a deterministic function

$$\pi(s) = a \tag{2}$$

Return

The *Return* is defined as:

$$G_t \doteq \sum_{k=0}^{H} \gamma^k r_{t+k+1} \tag{3}$$

Where γ is the *discount factor* and H is the *horizon*, that can be infinite (of course, mathematically, it should be infinite. because otherwise policy will also depend on time).

Markov Decision Process

A **Markov Decision Process**, MDP, is a 5-tuple (S, A, P, R, γ) where:

$$\begin{aligned}
&\text{finite set of states:}\\
&s \in S\\
&\text{finite set of actions:}\\
&a \in A \text{ or } A(s)\\
&\text{state transition probabilities:}\\
&P(s'|s,a) = Pr\{S_{t+1} = s'|S_t = s, A_t = a\}\\
&\text{reward function:}\\
&R(s) = \mathbb{E}[R_{t+1}|S_t = s]\\
&\text{where two latter equations can be written as one:}\\
&P(s',r|s,a) = Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}\\
&\text{discount factor:}\\
&\gamma \in [0,1)
\end{aligned} \tag{4}$$

Markov Property

to be able to use MDP as a model of a problem, the condition

$$\begin{aligned}
&Pr\{S_{t+1}, R_{t+1}|s_t, a_t, r_t, s_{t-1}, a_{t-1}, \dots, r_0, s_0, a_0\} =\\
&Pr\{S_{t+1}, R_{t+1}|s_t, a_t\}
\end{aligned} \tag{5}$$

should be true.

Value Function

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

$$V_\pi(s) = \mathbb{E}[G_t|S_t = s] \tag{6}$$

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

Optimal

$$V_*(s) = \max_\pi V_\pi(s) \tag{7}$$

Optimal Policy

Optimal policy is the policy that maximize value function (i.e. return) for each state, denoted by π^*:

$$\forall s, \pi \qquad V_{\pi^*}(s) \geq V_\pi(s) \tag{8}$$

Action-Value (Q) Function

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

$$q_\pi(s,a) = \mathbb{E}_\pi\left[G_t|S_t = s, A_t = a\right] \tag{9}$$

Optimal

The optimal value-action function:

$$q_*(s,a) = \max_\pi q^\pi(s,a) \tag{10}$$

Clearly, using this new notation we can redefine V^*, equation 7, using $q^*(s,a)$, equation 10:

$$V_*(s) = \max_{a \in A(s)} q_{\pi*}(s,a) \tag{11}$$

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 (6) and Q (9) functions if we expand them.

Value Function

$$\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi\left[G_t|S_t = s\right]\\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right]\\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2}|S_t = s\right]\\
&= \underbrace{\sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a)}_{\text{Sum of all probabilities } \forall \text{ possible } r}\\
&\quad \left[r + \gamma \underbrace{\mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2}|S_{t+1} = s'\right]}_{\text{Expected reward from } s_{t+1}}\right]\\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a)\left[r + \gamma V_\pi(s')\right]
\end{aligned} \tag{12}$$

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

$$\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t|S_t = s, A_t = a\right]\\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right]\\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2}|S_t = s, A_t = a\right]\\
&= \sum_{s',r} p(s',r|s,a)\left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2}|S_{t+1} = s'\right]\right]\\
&= \sum_{s',r} p(s',r|s,a)\left[r + \gamma V_\pi(s')\right]
\end{aligned} \tag{13}$$

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**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end
end
3. Policy Improvement
policy-stable $\leftarrow$ *true*
foreach $s \in S$ **do**
    old-action $\leftarrow \pi(s)$
    $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$
    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 $V(s) \in \mathbb{R}, \text{e.g} V(s) = 0$
$\Delta \leftarrow 0$
while $\Delta \geq \theta$ *(a small positive number)* **do**
  foreach $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end
end
ouput: Deterministic policy $\pi \approx \pi_*$ such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s')\right]$

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 π
  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**
    $\pi(s) \leftarrow \operatorname{argmax}_a Q(s, a)$
  end
end

Algorithm 3: Monte Carlo first-visit

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

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t)\right] \tag{14}$$

Where α 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:

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

n-step Sarsa

Define the n-step Q-Return

$$q^{(n)} = R_{t+1} + \gamma Rt + 2 + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n})$$

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

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

Forward View Sarsa(λ)

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

Forward-view Sarsa(λ):

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

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., ϵ-greedy)
  while *s is not terminal* **do**
    Take action a, observer r, s'
    Choose a' from s' using policy derived from Q (e.g., ϵ-greedy)
    $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$
    $a \leftarrow a'$
  end
end

Algorithm 4: Sarsa(λ)

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:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] \tag{15}$$

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., ϵ-greedy)
    Take action a, observer r, s'
    $Q(s, a) \leftarrow$
    $Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$
  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.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(\underbrace{r + \gamma \max_a Q(s', a'; \theta_{i-1})}_{\text{target}} - \underbrace{Q(s, a; \theta_i)}_{\text{prediction}} \right)^2 \right] \tag{16}$$

Where θ 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 ϵ 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

Forked from Francesco Saverio Zuppichini and improved by Alireza Nobakht
https://github.com/alirezanobakht13/Reinforcement-Learning-Cheat-Sheet

Double Deep Q Learning