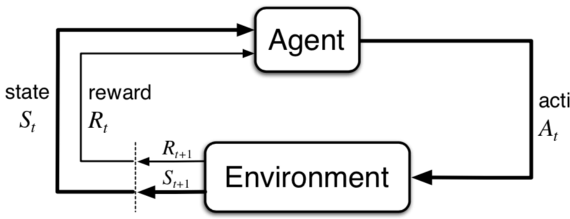
Reinforcement Learning Cheat Sheet

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

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

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

Reward

The total *reward* is expressed as:

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

Where γ 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, γ) where:

$$\begin{aligned}
&\text{finite set of states:}\\
&s \in S\\
&\text{finite set of actions:}\\
&a \in A\\
&\text{state transition probabilities:}\\
&p(s'|s,a) = Pr\{S_{t+1} = s' | S_t = s, A_t = a\}\\
&\text{expected reward for state-action-nexstate:}\\
&r(s', s, a) = \mathbb{E}[R_{t+1} | S_{t+1} = s', S_t = s, A_t = a]
\end{aligned} \tag{3}$$

Value Function

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

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

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

Optimal

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

Action-Value (Q) Function

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

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

Optimal

The optimal value-action function:

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

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

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

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 booth Value 4 and Q 6 functions if the expand them.

Value Function

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

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

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

Contraction Mapping

Definition

Let (X, d) be a metric space and $f : X \to X$. We say that f is a *contraction* if there is a real number $k \in [0, 1)$ such that

$$d(f(x), f(y)) \leq kd(x, y)$$

for all x and y in X, where the term k is called a *Lipschitz coefficent* for f.

Contraction Mapping theorem

Let (X, d) be a complete metric space and let $f : X \to X$ be a contraction. Then there is one and only one fixed point x^* such that

$$f(x^*) = x^*$$

Moreover, if x is any point in X and $f^n(x)$ is inductively defined by $f^2(x) = f(f(x))$, $f^3(x) = f(f^2(x))$, ..., $f^n(x) = f(f^{n1}(x))$, then $f^n(x) \to x^*$ as $n \to \infty$. This theorem guarantees a unique optimal solution for the dynamic programming algorithms detailed below.

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 $\Delta < \theta$ *(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*
while *not policy-stable* **do**
  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* $\neq \pi(s)$
  end
end

Algorithm 1: Policy Iteration

Value Iteration

We can avoid to wait until $V(s)$ has converged and instead to 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 $\Delta < \theta$ *(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 no-stationary problems, the Monte Carlo estimate for, e.g, V is:

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

Where α is the learning rate, how much we want to forget about pass experiences.

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:

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

The following algorithm gives a generic implementation.

Initialise $Q(s, a)$ arbitrarily
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 4: Q Learning

Sarsa

Sarsa (State-action-reward-state-action) is to control what Temporal Difference is to policy evaluation.

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

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, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
foreach *episode* $\in$ *episodes* **do**
  Initialise *eligibility traces*: $E(s, a) = 0$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
  Initialise S, A
  while s *is not terminal* **do**
    take action A, observe R, S'
    choose A' from S' using policy derived from Q (e.g. ϵ-greedy)
    $delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
    $E(S, A) \leftarrow E(S, A) + 1$
    foreach $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$ **do**
      $Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$
      $E(s, a) \leftarrow \gamma \lambda E(s, a)$
    end
    $S \leftarrow S'$, $A \leftarrow A'$
  end
end

Algorithm 5: Sarsa(λ)

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.

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

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 $y_i \leftarrow$
    $\begin{cases} r_j & \text{for terminal } s'_j \\ r_j + \gamma \max_a Q(s', a'; \theta) & \text{for non-terminal } s'_j \end{cases}$
    Perform gradient descent step on $(y_j - Q(s_j, a_j; \Theta))^2$
    $s \leftarrow s'$
  end
end

Algorithm 6: Deep Q Learning