Reinforcement Learning
Elementary Solution Methods

Contents

1 Dynamic Programing

Dynamic programing (DP) is a mathematically well-developed theory. It though requires a complete and accurate model of the environment. It entails the basis for every other methods, who attempt to realize the as DP, with lesser computational costs.

We hereinafter assume that the state space and each state's space action is finite (i.e $|\mathcal{S}| < \infty$ and $\forall s \in \mathcal{S}$, $|\mathcal{A}(s)| < \infty$). The dynamic is supposed to be fully given by the set of transition probabilities $\mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}$ for all state and relative actions.

Remember that the optimal state value function V^* and action value function Q^* verify the Bellman optimality equations :

$$\begin{aligned} &\forall s \in \mathcal{S}, \quad V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^*(s')\right] \\ &\forall (s,a) \in \mathcal{S} \times \mathcal{A}(\mathcal{S}), \quad Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a' \in \mathcal{A}(s')} Q^*(s',a')\right] \end{aligned} \tag{1}$$

1.1 Policy evaluation

We hereinbefore describe how to compute the state value function for a policy π. This is called the **policy evaluation** problem, or the *prediction problem*.

We have that $\forall s \in \mathcal{S}$:

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}(s)} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s')\right] \tag{2}$$

If the model is known, this gives us $|\mathcal{S}|$ simultaneous linear equations with $|\mathcal{S}|$ unknowns. Even if it is analytically solvable, it might result it tedious computations. Therefore iterative methods are more suitable ways of solving this problem.

Let $V_0 : S \to \mathbb{R}$ an initial , arbitrary mapping for the state value function. We build the sequence $(V_k)_k$ of mapping using the Bellman equation for V^{π} as an update rule :

$$\forall s \in \mathcal{S}, \quad V_{k+1}(s) = \sum_{a \in \mathcal{A}(s)} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V_k(s')\right] \tag{3}$$

Indeed, V^{π} is a fixed-point problem for the Bellman equation, and therefore we might have chance to convert towards this fixed-point by using it as an iterative formula. It can actually be proved that under the same hypothesis guaranteeing the existence of V^{π} that :

$$V_k \underset{k \to \infty}{\longrightarrow} V^{\pi} \tag{4}$$

Such a procedure is known as an **iterative policy evaluation**.

Iterative Policy Evaluation Algorithm

```
   Input: Policy $\pi$ to be evaluated
1  Init $V(s) = 0, \forall s \in \mathcal{S}$. begin
2      repeat
3          $\Delta \leftarrow 0$
4          foreach $s \in \mathcal{S}$ do
5              $v \leftarrow V(s)$
6              $V(s) = \sum_{a \in \mathcal{A}(s)} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} [\mathcal{R}^{a}_{ss'} + \gamma V(s')]$
7              $\Delta = \max(\Delta, |v - V(s)|)$
8          end
9      until $\Delta < 0$;
10 end
   Output: $V \simeq V^{\pi}$ state value function for policy $\pi$
```

There exist two implementation of this algorithm : one that needs to keep trace of both the old and new iterations on the state-value function estimate, and another that does the actualization *in-place*. The latter is known to converge faster.

1.2 Policy improvement

We just saw how to compute the value function of a given policy. This is also going to help us find better ones. We consider the following problem : given V^{π}, would it be better or worse to change the policy ? A good way to find out is, at a given state $s \in \mathcal{S}$, to choose an action $a \neq \pi(s)$, switch to the next state and then follow π. The value of such a behavior is given by :

$$Q^{\pi}(s,a) = \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s')\right] \tag{5}$$

If such a value is greater than $V\pi(s)$, the new policy defined by *greedily* taking a in state s is better in overall than π.

The latest exemple is a special case of a general result called the **policy improvement** theorem. This theorems state the following :

Policy Improvement Theorem

Let π and π' be two policies such that $\forall s \in \mathcal{S}$:

$$Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s) \tag{6}$$

then π' is a better policy than π, i.e :

$$\forall s \in \mathcal{S}, \quad V^{\pi'}(s) = V\pi(s) \tag{7}$$

We now consider π' the *greedy policy* w.r.t to Q^{π} : $\forall s \in \mathcal{S}$:

$$\begin{aligned} \pi'(s) &= \underset{a \in \mathcal{A}(s)}{\text{arg max}} \{Q(s,a)\} \\ &= \underset{a \in \mathcal{A}(s)}{\text{arg max}} \left\{ \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s') \right] \right\} \end{aligned} \tag{8}$$

What π' achieves is to take the action that looks the best in the short term according to V^{π}. This lead to a policy that as good as or better as the original policy π.

1.3 Policy iteration

Once a policy π has been improved to yield a better policy π', we can then compute $V^{pi'}$ and repeat the improvement process :

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \dots \xrightarrow{I} \pi^* \tag{9}$$

For finite MDPs, such a procedure converges to an *optimal policy* and optimal value functions in a finite number of iterations. It is called a **policy iteration** procedure.

Policy iteration

```
1.Initialization
$V(s)$ and $\pi(s)$ arbitrary for all $s \in \mathcal{S}$

2. Policy Evalutation
repeat
    $\Delta \leftarrow 0$
    foreach $s \in \mathcal{S}$ do
        $v \leftarrow V(s)$
        $V(s) = \sum_{a \in \mathcal{A}(s)} \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V(s')]$
        $\Delta = \max(\Delta, |v - V(s)|)$
    end
until $\Delta < 0$;

3.Policy Improvement
policy_stable $\leftarrow$ true
foreach $s \in \mathcal{S}$ do
    $b \leftarrow \pi(s)$
    $\pi(s) = \underset{a \in \mathcal{A}(s)}{\text{argmax}} \{\sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s)]\}$
    If $b \neq \pi(s)$, policy_stable$\leftarrow$false
end
If policy_stable, then stop. Else go to 2.
```

1.3.1 Value iteration

There is a major drawback to the previous method. Indeed, each iteration requires a full value evaluation step, which can be challenging for large state space. However, they are many steps in the value evaluation routine that won't have any effect on the policy improvement. One can therefore wonder if he must wait for convergence, or just proceed to a few sweeps in value evaluation and then perform a policy improvement step.

In the context of policy iteration, one important special case is when the policy evaluation is stoped after only one full state space sweep. This simplified algorithm is called the value iteration procedure. It performs a simple backup procedure :

$$\forall s \in \mathcal{S}, \quad V_{k+1}(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s')\right] \tag{10}$$

For any arbitrary V_0, it can be shwon that $V_k \underset{k\to\infty}{\longrightarrow} V^*$ under the same hypothesis that ensure the existence and unicity of V^*. As one can notice, it actually uses the *Bellman optimality equation as an update rule* !

1.3.2 Asynchronous DP

Even is value iteration is more tractable than the policy iteration method, it still requires many sweeps in the state space, which can be computationally demanding.

The in-place asynchronous DP performs back-ups in any order whatsoever. This methods converges as long as it visits all the states. The most extreme form of this method is achieved when only one sweep is performed in state space before policy improvement. It namely allows to focus on a particular set of the state space, that is though relevant for optimality.

1.4 Generalized policy iteration

Policy iterations techniques, as different from one another as they may be, carry on the same iterative steps. One make the value function consistent with the policy (*value evaluation*) and the other make the policy greedy with respect to the current value function (*policy improvement*).

The generalized policy iterations (GPI) refers to the general idea of interaction between policy evaluation and policy improvement. Almost all reinforcement learning algorithms can be derived from a GPI.

The full GPI formulation and solving is of course much more tractable than the exaustive search. Given a problem with $|\mathcal{S}| = n$ and $\forall s \in \mathcal{S}$, $|\mathcal{A}(s)| = m$, an exhaustive search would imply comparing m^n different policies. DP giving out a polynomial complexity in m and n, it is therefore exponentially better than the exhaustive search !

2 Monte Carlo Methods

Unlike with dynamic programming, we no longer assume complete knowledge of the environment. Monte Carlo methods only require experience of environment interactions, be it from samples of sequences of state, actions and rewards or online simulated experience.

We define Monte Carlo methods only for *episodic states*, in order to have well-defined returns. Only at the end of an episode would the policy and its value function updated.

As with DP, we'll consider Monte Carlo methods for :

- policy evaluation
- policy improvement
- generalized policy iterations (GPI)

2.1 Monte Carlo policy evaluation

The basic idea behind Monte Carlo methods is to average the return observed after a visit to a state on many episodes, in order to approach the expected return (i.e what the value function is trying to express !).

2.1.1 Vocabulary

Let $s \in \mathcal{S}$, and we wish to evaluate $V^\pi(s)$. Each occurence of the state s is called a *visit*. We distinguish two different based MC methods :

- The *every-visit* MC method to estimate $V^\pi(s)$ as the average return following the visit of s.
- The *first-visit* MC method which averages only over the returns following the first visits to s.

Those are two very similar methods that have slightly different properties. Both converges to $V\pi$ as the number of visits builds up to $+\infty$. In the following, we'll focus on the first-visit method.

2.1.2 The FVMC algorithm

We hereinafter describe more precisely the first-visit Monte Carlo method for policy evaluation.

First-Visit Monte Carlo Method

```
1.Initialize :
    $\pi \leftarrow$ policy to be evaluated
    $V \leftarrow$ arbitrary state-value function
    $Return(s) \leftarrow$ empty list $\forall s \in \mathcal{S}$
2. Repeat (for large number of times)
    a) Generate an episode using $\pi$
    b) For each state $s$ in the episode :
        $R \leftarrow$ return following the first occurence of $s$
        Append $R$ to returns
        $V(s) = average(Return(s))$
```

2.1.3 Monte Carlo estimation of action values

When a model is not available, it is particularly helpful to estimate action values rather than the state values. Indeed, without a model (especially the transition probabilities $\mathcal{P}^a_{ss'}$), one must explicitly estimate the value of each action in order for the values to be useful in suggesting a policy.

We therefore wish to estimate $Q^{\pi}(s,a)$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$. The philosophy for estimating those values are te same as the every-visit and first-visit MC methods for state value evaluation.
However, many relevant state/actions may never be visited. Indeed, if π is deterministic, following π will allow to observe returns only from one of the actions in each state. The MC estimate of other actions will therefore not improve by experience.

This problem is a general problem in reinforcement learning and is designated as *maintaining exploration.* They are two different ways to deal with it :

- The exploring start method, where we specify the first step of an episode with a specific state-action pair, with each pair having a non-zero probability of being selected at the start. However, it is not a realistic method for real-life interactions.
- The best solution is therefore to consider policies that are stochastic with a non-zero probability for every action, in every state.

2.2 Monte Carlo Control

The term control denotes the approximation of optimal policies. We hereinafter consider a Monte Carlo version of the classical policy iteration. Like with DP, we perform alternative steps of policy evaluation and policy improvement (policy evaluation being performed as describe before).

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \dots \xrightarrow{I} \pi^* \tag{11}$$

We assume that episodes are generated with exploring start and that we observe an infinite number of episodes. Therefore the Monte-Carlo method computes Q^{π_k}, $\forall k$ exactly. As before, policy improvement is done by making the policy greedy w.r.t the current value function. With the available action-value function, no model ins needed for constructing the greedy policy :

$$\forall s \in \mathcal{S},\, \pi(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \{Q^{\pi}(s,a)\} \tag{12}$$

Hence policy improvement is performed by constructing π_{k+1} by making it greedy with respect to Q^{π_k}. We can show that the *policy improvement* theorem still checks out. Indeed, $\forall s \in \mathcal{S}$:

$$\begin{aligned} Q^{\pi_k}(s, \pi_{k+1}(s)) &= Q^{\pi_k}(s, \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \{Q^{\pi_k}(s,a)\}) \\ &= \max_{a \in \mathcal{A}(s)} Q^{\pi_k}(s,a) \\ &\geq Q^{\pi_k}(s,a) \quad \forall a \in \mathcal{A}(s) \\ &\geq Q^{\pi_k}(s, \pi_k(s)) \\ &\geq V^{\pi_k} \end{aligned} \tag{13}$$

which ensures that each π_{k+1} is uniformly better that π_k (unless it is equal to π_k, in which case both policies are optimal).

However, we've made two unlikely assumptions to achieve this result : exploring start and infinite number of episodes. Like we did in DP, the latest can be discarded, as we do not need full convergence of the value evaluation process to perform policy improvement. Like with some extreme case of the GPI, is would be natural for MCPI to alternate bewteen evaluation and improvement on an episode-to-episode basis. This leads to the *Monte Carlo with Exploring Starts* policy iteration algorithm. This algorithms ensures that if it converges, it must converges toward an optimal policy. However, convergences has not yet been formally proven, and remains an open question in the reinforcement learning community.

2.2.1 On Policy Monte Carlo Control

As said before, we need to find a way to avoid the unlikely assumption of exploring start. Therefore we need to find a way to ensure that all actions are selected infinitely often. The first approach in doing that is called on-policy control (it evaluates and improve directly the policy that is used to make the decisions).

For on-policy MC control, the policy is assumed to be **soft** :

$$\forall (s,a) \in \mathcal{S} \times \mathcal{A}(s), \quad \pi(s,a) > 0, \tag{14}$$

To improve the policy, we gradually shift the policy toward the soft-equivalent of greediness, which we denote as ε-greedy.
For the ε-greedy policy, the optimal value is selected with $1-\varepsilon$ probablity, and a random action is taken with ε probability. Such a policy is still soft, since :

$$\forall (s,a) \in \mathcal{S} \times \mathcal{A}(s), \quad \pi(s,a) \geq \frac{\varepsilon}{|\mathcal{A}(s)|} > 0 \tag{15}$$

We saw that without the exploring start hypothesis, we cannot improve the policy by making it greedy with respect to the action-value function. Fortunately, the GPI does not need for us to take it all the way to greediness. Actually, the ε-greedy policy, with respect to Q^π is an improvement over any ε-soft policy. Indeed, $\forall s \in \mathcal{S}$:

$$\begin{aligned} Q^\pi(s,\pi'(s)) &= \sum_{a\in\mathcal{A}(s)} \pi'(s,a)Q^\pi(s,a) \\ &= (1-\varepsilon)\max_a Q^\pi(s,a) + \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_{a\in\mathcal{A}(s)} Q^\pi(s,a) \\ &\geq \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_{a\in\mathcal{A}(s)} Q^\pi(s,a) + (1-\varepsilon)\sum_{a\in\mathcal{A}(s)} \frac{\pi(s,a)-\frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon}Q^\pi(s,a) \\ &\geq \sum_{a\in\mathcal{A}(s)} \pi(s,a)Q^\pi(s,a) = V^\pi(s) \end{aligned} \tag{16}$$

thus $\pi' \geq \pi$, with equality holding for π' and π being optimal among the ε-soft policies.

ε-soft On-Policy Monte Carlo Control Algorithm

```
1.Initialize for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$ :
    $Q(s,a) \leftarrow$ arbitrary
    $Return(s,a) \leftarrow$ empty list
    $\pi \leftarrow$ arbitrary $\varepsilon$-soft policy

2.Repeat Forever
    (a) Generate an episode using $\pi$
    (b) For each pair $(s,a)$ in that episode :
        $R \leftarrow$ return following the occurence of $(s,a)$
        Append $R$ to $Return(s,a)$
        $Q(s,a) \leftarrow$ average$(Return(s,a))$
        (c) For each $s$ in the episode : $a^* \leftarrow \underset{a}{\text{argmax}}\{Q(s,a)\}$
        Forall $a \in \mathcal{A}(s)$ :
            $\pi(s,a) \leftarrow \begin{cases} 1-\varepsilon+\frac{\varepsilon}{|\mathcal{A}(s)|} \text{ if } a = a^* \\ \varepsilon \text{ otherwise} \end{cases}$
```

2.2.2 Evaluation a policy while following another

Let's consider the following problem : we have episodes generated from a policy differenet that the one we wish to evaluate. How can we learn V^π and Q^π with episodes generated with π', i.e learn from experience "off" policy ?

Of course, we must require that every action taken under π is also taken under π' :

$$\forall (s,a) \in \mathcal{S} \times \mathcal{A}(s), \quad \pi(s,a) > 0 \Rightarrow \pi'(s,a) > 0 \tag{17}$$

Let us consider the i^{th} first visit to some state $s \in \mathcal{S}$. Let $p_i(s)$ and $p'_i(s)$ the probabilities of the sequence that followed that visit under π nad π'. Let $R_i(s)$ be the corresponding return from state s. To average these to obtain an unbiased estimate of V^π, we need only weight them by $\frac{p_i(s)}{p'_i(s)}$, i.e

$$V^\pi(s) \simeq \frac{\sum_{i=1}^{n_s} \frac{p_i(s)}{p'_i(s)} R_i(s)}{\sum_{i=1}^{n_s} \frac{p_i(s)}{p'_i(s)}} \tag{18}$$

where n_s denoted the number of returns obtained from s.

At first glance, one can think than the need for p_i and p'_i requires a model of the environment. This is true, but the fact that we only need the ratio of those two quantities bypasses the need of a model :

$$\begin{aligned} \forall s \in \mathcal{S}, \quad \frac{p_i(s)}{p'_i(s)} &= \frac{\prod_{k=1}^{T_i} \pi(s_k, a_k) \mathcal{P}_{s_k s_{k+1}}^{a_k}}{\prod_{k=1}^{T_i} \pi'(s_k, a_k) \mathcal{P}_{s_k s_{k+1}}^{a_k}} \\ &= \prod_{k=1}^{T_i} \frac{\pi(s_k, a_k)}{\pi'(s_k, a_k)} \end{aligned} \tag{19}$$

2.2.3 Off-Policy Monte Carlo Control

The **off-policy** MC control method uses two policies :

- A policy to generate behavior (*behavior policy*)
- A policy to be evaluated and improved (*evaluated policy*)

For exploring all possibilities, we recquire the behavior policy to be soft. Therefore, π' is maintenant an arbitrary soft policy.

An off-policy Monte Carlo Control algorithm

```
1.Initialize  for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$ :
    $Q(s,a) \leftarrow$ arbitrary
    $N(s,a) \leftarrow 0$
    $D(s,a) \leftarrow 0$
    $\pi \leftarrow$ arbitrary deterministic policy

2.Repeat Forever
    (a) Select a policy $\pi'$ and use it to generate an episode:

            $s_0, a_0, r_1 \dots, s_{T-1}, a_{T-1}, r_T, s_T$

    (b) $\tau \leftarrow$ latest time at which $a_\tau \neq \pi(s_\tau)$
    (c) For each pair $s$ appearing in the episode at time $\tau$ or later :
        $t \leftarrow$ time of the first occurence of $(s,a)$ such that $t \geq \tau$
        $w \leftarrow \prod_{k=t+1}^{T-1} \frac{1}{\pi'(s_k,a_k)}$
        $N(s,a) \leftarrow N(s,a) + wR_t$
        $D(s,a) \leftarrow D(s,a) + w$
        $Q(s,a) \leftarrow \frac{N(s,a)}{D(s,a)}$
    (d) For each $s$
        $\pi(s) \leftarrow \underset{a}{\text{argmax}} \{Q(s,a)\}$
```

2.3 Brief Summary

Monte Carlo methods learn value function and optimal policies from experience in the form of sample episodes. It has different advantages over the standard DP method :

- no model of the environment is needed
- easier to focus on small subset of the state

Maintaining sufficient exploration is an issue in MC method of control, since we need to keep exploring the whole state space to average returns. There exists two solution to that problem : using exploring starts or soft-policies.

On-policy methods are the name given to methods where the agent commits to always exploring and tries to find the best policy that keeps on exploring. On the other hand, *off-policy* methods explore but learns a deterministic optimal policy that may be unrelated to the policy followed.

Monte-Carlo methods are recent, and their convergence properties are unclear. Their effectiveness in practice has also little been tested.

3 Temporal Differences Methods

Temporal differences (TD) methods can be seen as a combination of DP and MC methods. Like MC, TD can learn from experience without a model of its environment. However, like DP, temporal differences methods update estimates based on other learned estimates : they are therefore said to *bootstrap*.

We'll consider here the problems a policy evaluation and control problem (finding an optimal policy).

3.1 Temporal differences policy evaluation

3.1.1 The prediction problem with TD

When MC methods wait until the end of an episode to determine how to increment a given $V(s_t)$, temporal differences methods wait only until the next time step. At time $(t+1)$, they immediately form a target and make a useful update using the observed reward and the estimate $V(s_{t+1})$.

The simplest temporal difference method, known as TD(0), gives out :

$$V(s_t) = V(s_t) + \alpha\big[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\big] \tag{20}$$

As one can see, when the target was R_t for Monte Carlo methods, it is $r_{t+1} + \gamma V(s_{t+1})$ for the TD method. Indeed, as we recall, $\forall s \in \mathcal{S}$:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi\left[R_t | s_t = s\right] \\ &= \mathbb{E}_\pi\left[r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s\right] \end{aligned} \tag{21}$$

When MC update rule is taken from the first line of the equality, TD update rule implements the second line. Equation (20) can be explain the following way :

$$\underbrace{V(s_t)}_{\text{new value}} = \underbrace{V(s_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \big[\underbrace{\overbrace{r_{t+1}}^{\text{instant reward}} + \overbrace{\gamma}^{\text{discount rate}} \overbrace{V(s_{t+1})}^{\text{estimate of future value}}}_{\text{learned value}} - V(s_t)\big] \tag{22}$$

TD therefore combines the sampling of MC and the bootstrap of DP. Actually, TD and MC updates are called as sample backups because they involve looking ahead to a sample successor state, using its value and the reward along the way to compute a back-up value (and therefore are different than the *full back-up* methods implemented in DP).

Temporal Difference Learning for the State Value Estimation

```
1.Initialize  $V(\cdot)$ arbitrarily, $\pi$ the policy to be evaluated

2. Repeat for every episode :
       Initialize $s$
       Repeat for every step in the episode :
           $a \leftarrow$ action given by $\pi$ for $s$
           Take action $a$, observe $r$, switch to $s'$
           $V(s) \leftarrow V(s) + \alpha\big[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\big]$
           $s \leftarrow s'$
   until $s \in \mathcal{S}^+$.
```

3.1.2 The advantages of TD prediction method

TD has many advantages : like MC, there is no need for a model for the environment, rewards and next step probability distributions. It is naturally implement *on-line*, in a fully incremental fashion (unlike MC methods, we don't have to wait for an episode to return to perform a value update, which can make the learning slow). Hence, TD methods are particularly efficient for very long episodes, and crucial for *continuing tasks*.

However, one may ask if there is still any guarantee of convergence ? The answer is yes. For any fixed policy, the TD algorithm has been proved to converge to V^{π} with probability 1 if the step size parameter decreases according to the usual stochastic approximation conditions :

$$\sum_k \alpha_k(a) = +\infty \quad \text{and} \quad \sum_k \alpha_k^2(a) < +\infty \tag{23}$$

where $\alpha_k(a)$ is the learning rate for the kth visit of the pair $(s, a) \in \mathcal{S} \times \mathcal{A}(s)$.

As to which of the MC or TD method converges faster to the correct value function, it is still an open question in reinforcement learning. However, TD methods have in practice been found to converge faster than MC methods.

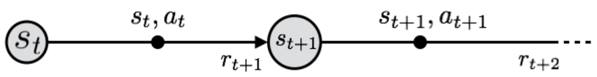
3.2 Temporal Difference Control

As we did before, we are going to follow the general pattern of GPI, only this time we'll use temporal difference learning methods for the policy evaluation parts.

As with MC methods, we face the need to trade-off between exploration and exploitation. Again, the available approaches fall into two main classes : off-policy and on-policy.

3.2.1 SARSA : On-Policy TD Control

We hereinafter present an on-policy control method. As we recall, we are presented with the following sequence : In the previous sections we consider transitions from state to state and learned the values of the



states. We now consider transition from state-action pair to state-action pair and learn the action-value function with the following update rule :

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha\big[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\big] \tag{24}$$

which is done after every transition from a non-terminal state (if $s_{t+1} \in \mathcal{S}^+$, then $Q(s_{t+1}, a_{t+1}) = 0$).

Hence, the update rule uses element of the quintuple event :

$$\big(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}\big) \tag{25}$$

i.e State Action Reward State Action, given it the name *SARSA*.

As in all on-policy methods, we continually estimate Q^{π} for the behavior policy π, and at the same time change π toward greediness, with respect to Q^{π}.

The General Sarsa Algorithm

```
1. Initialize $Q(s,a)$ arbitrarily $\forall (s,a) \in \mathcal{S} \times \mathcal{A}(s)$

2. Repeat for each episode :
       Initialize $s$
       Choose $a \in \mathcal{A}(s)$ using a soft policy derived from $Q$ (typically $\varepsilon$-greedy)
       Repeat for each step of the current episode :
           Take $a$, observe $r, s'$
           Choose $a'$ from $s'$ using policy derived from $Q$
            $Q(s,a) \longleftarrow Q(s,a) + \alpha\big[r + \gamma Q(s',a') - Q(s,a)\big]$
           $a \leftarrow a'$, $s \leftarrow s'$
   until $s \in \mathcal{S}^+$.
```

The convergences properties of $SARSA$ depend on the nature of the policy's dependency on Q. Indeed, $SARSA$ converges with probability 1 to an optimal policy in action value function as long as all the sate action pars are visited an infinite number of time and the policy converges in the limit to the greedy policy.

3.2.2 Q-learning : Off-Policy TD Control

The most important breakthrough in reinforcement learning was the development of an off-policy temporal difference algorithm known as Q-learning. It's update rule is given by :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a) - Q(s_t, a_t)\big] \tag{26}$$

The learned action-value function Q directly approximates Q^*, independently of the policy being followed. The policy still has an effect in that it determines which state-actions pairs are visited and updated. However, all that is required for convergence is that all pairs continue to be updated.

Along the hypothesis and a variant of the usual stochastic approximation condition on the step-size parameters, Q_t has been shown to converge with probability A to Q^*.

Q-Learning Algorithm

```
1. Initialize $Q(s,a)$ arbitrarily $\forall (s,a) \in \mathcal{S} \times \mathcal{A}(s)$

2. Repeat for each episode :
       Initialize $s$
       Repeat for each step of the current episode :
           Choose $a \in \mathcal{A}(s)$ using arbitrary policy
           Take $a$, observe $r, s'$
           $Q(s,a) \longleftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a' \in \mathcal{A}(s')} Q(s',a') - Q(s,a)\big]$
           $s \leftarrow s'$
   until $s \in \mathcal{S}^+$.
```

3.3 Brief Summary

Temporal differences methods are the most widely used methods in reinforcement learning. Indeed, they can be applied on line with a minimal amount of computation.

The special case of TD methods we displayed hereinbefore should rightly be called *one step, tabular, model-free* temporal difference methods. Indeed, they can be extended to multistep forms, using function approximation rather than tables (giving a link to artificial neural networks).