1 RL in finite domains

So far we have seen the very basics of MDPs and some dynamic programming method to predict V^* and π^*. In this chapter, we will focus on RL methodologies which aim to estimate as well V^* and π^*, but in unknown environment where the MDP structure is not known. RL theory provides us a vast catalogue of algorithm that can be identified by the following characteristics

- **Model-free vs Model-based** Model free approaches try to directly estimate V^*, without computing the MDP model. On the other hand, model-based approach estimate the MDP model, and then they apply classic dynamic programming methods to estimate the state-value function.
- **On-policy vs Off-policy** On policy means that we estimate some metric, like state-value function, of the policy that we are using to collect the data. Off-policy methodologies try to estimate a policy that is different from the one generating the data.
- **Online vs Offline** An online algorithm update the policy or value functions every time we get new data, interleaving data collection and learning phase. Offline algorithms separate the data collection phase from the learning phase.
- **Tabular vs Function approximation** Tabular approaches stores directly the values of the value functions. This is feasible only on small problems. Function approximation doesn't store directly the values of the value function, but it approximate it with some function. Doing so, it can even describe infinite value function.
- **Value-based vs Policy-based vs Actor-Critic** Value based approaches try to estimate the value function. Policy-based search in policy space to find the optimal policy. Actor-critic combines these two approaches.

In the following sections we will focus on model-free algorithms. We will see both prediction and control case.

1.1 Model-free prediction

With Model–free prediction we want to estimate the value function of an unknown MRP[1]. This is clearly different from what we have seen in the previous chapter, because we don't know the model of the MDP. We only know which actions are available and which states makes up the MDP

[1] MDP + policy

1.1.1 Monte-Carlo reinforcement learning (TD(1))

MC is a model-free approach which don't need the knowledge of the MDP transitions/rewards. This method learn directly from episodes of experience. It must be used in episodic MDPs[2] because the learning phase is based on the entire trajectory[3], from the starting state to the goal state. MC can't be used in MDPs that don't terminate. MC can be used for both prediction and control. For prediction we have

- **Input** Episodes $\{s_1, a_1, r_1, \dots\ s_T\}$, generated by following policy π in given MDP
- **Output** Value function V^π

First, let's recall what the Monte-Carlo approach consist of. MC is a class of methods that rely on repeated random sampling to obtain numerical results. Let X be a random variable with mean $\mu = E[X]$ and variance $\sigma^2 = Var[X]$. Let $x_i \sim X$, $i = 1, \dots, n$ be n i.i.d. realization of X.
The empirical mean of X is

$$\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^{N} x_i$$

We also know that the empirical mean is an unbiased estimator for which

$$E[\hat{\mu}_n] = \mu, \quad Var[\hat{\mu}_n] = \frac{Var[X]}{n}$$

As we have said, our goal is to compute V^π. We know that based on the return v_t

$$V^\pi(s) = E[v_t | s_t = s]$$
$$v_t = r_{t+1} + \gamma r_{t+2} + \dots + \gamma^{T-1} r_{t+T}$$

Monte Carlo policy evaluation uses empirical mean return instead of expected return to estimate V^π. In practice, we perform various episodes and we collect the returns for every state. Then, we use the empirical mean of the returns collected in s to estimate $V^\pi(s)$. The averaging of the returns can be performed in two ways

- **First visit** Average returns only for the first time s is visited.
- **Every visit** Average returns for every time s is visited.

[2]An episodic MDP means that we have an absorbing/goal state and when we reach it, the problem starts all over again from the starting state

[3]A trajectory is a sequential combination of states, actions, and rewards $s_0, a_0, r_0, s_1, a_1, r_1, \dots$

Example - MC prediction Suppose to have the following episodes with $\gamma = 1$

$$\begin{tikzcd} S_1 \arrow[r, "+1"'] & S_3 \arrow[r, "-1"'] & S_4 \arrow[r, "+2"'] & S_1 \arrow[r, "-2"'] & S_5 \end{tikzcd}$$

$$\begin{tikzcd} S_2 \arrow[r, "-1"'] & S_3 \arrow[r, "+1"'] & S_1 \arrow[r, "-2"'] & S_4 \arrow[r, "+2"'] & S_5 \end{tikzcd}$$

We want to estimate the value-state function of s_1 using first visit

$$\begin{aligned} \hat{V}_1^\pi(s_1) &= 1 - 1 + 2 - 2 = 0 \\ \hat{V}_2^\pi(s_1) &= -2 + 2 = 0 \\ \hat{V}^\pi(s_1) &= \frac{0+0}{2} = 0 \end{aligned}$$

Using every visit, we consider all the occurrences of s_1

$$\begin{aligned} \hat{V}_{11}^\pi(s_1) &= 1 - 1 + 2 - 2 = 0 \\ \hat{V}_{12}^\pi(s_1) &= -2 \\ \hat{V}_{21}^\pi(s_1) &= -2 + 2 = 0 \\ \hat{V}^\pi(s_1) &= \frac{(0) + (-2) + (0)}{3} = -\frac{2}{3} \end{aligned}$$

This two methods have different properties. First visit is unbiased. We know for the bias-variance tradeoff that, if we have low bias, we have high variance. So first visit is suited for problems with many many training samples. First visit will produce noisy estimation of the state-value function even with a decent amount of samples. This limits a lot the learning speed, because we need to calculate a lot of episodes. Every visit is consistent[4]. It has more bias and less variance compared to first visit. This can be explained by the fact that every visit considers some transitions of an episode multiple times, introducing bias. But, from the same number of episodes, it can extract more training samples reducing the variance. From a computational point of view, it would be nice if we could calculate incrementally the empirical mean of the return. If we observe a new episode, we can update the mean without recalculating it from scratch.

[4]An estimator is consistent if $\lim_{n\to\infty} \hat{x}_n = E[x]$. If we have infinite samples the estimator converges to the true value

$$
\begin{aligned}
\hat{\mu}_k &= \frac{1}{k}\sum_{j=1}^{k} x_j \\
&= \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right) \\
&= \frac{1}{k}\Big(x_k + (k-1)\hat{\mu}_{k-1}\Big) \\
&= \hat{\mu}_{k-1} + \frac{1}{k}(x_k - \hat{\mu}_{k-1}) \qquad (1)
\end{aligned}
$$

In the MC case, using (1) we have

$$V(s_t) \leftarrow V(s_t) + \frac{1}{N(s_t)}(v_t - V(s_t)) \qquad (2)$$

where $N(s_t)$ is the number of times we have calculated the return for s_t. This updates can be extended to non-stationary problems. In this cases is useful to forget old episodes.

$$V(s_t) \leftarrow V(s_t) + \alpha(v_t - V(s_t)) \qquad (3)$$

where α is a learning rate between zero and one. This has the result to weight more the new samples, because the weighting of $(v_t - V(s_t))$ doesn't converge to zero as in the case of $\frac{1}{N(s_t)}$, but remain constant.

Note - Characteristic recap

- Must work on episodic MDPs. To learn we use the entire episode
- Unlike dynamic programming, MC can evaluate at each episode only one choice at each state
- MC doesn't bootstrap. This term will be clear later
- Time required to estimate one state does not depend on the total number of states, but on the variance of the samples

1.1.2 Temporal difference learning (TD(0))

Before explaining this new algorithm, it's worth recalling a very important learning rate property.

Proposition 1.1. *Let X be a random variable in $[0,1]$ with mean $\mu = E[X]$. Let $x_i \sim X$, $i = 1, \dots, n$ be n i.i.d. realizations of X. Consider the following exponential estimator*

$$\mu_i = (1 - \alpha_i)\mu_{i-1} + \alpha x_i$$

with $\mu_i = x_1$ and α_i's are learning rates.
If $\sum_i \alpha_i = \infty$ and $\sum_i \alpha_i^2 < \infty$, then $\hat{\mu}_n \overset{a.s.}{\to} \mu$. The estimator is consistent.

For example $\frac{1}{i}$ satisfies the above condition, and so produces a consistent estimator. The very good thing about the estimator above is that, manipulating the learning rate, we can interpolate between the previous estimator value and the new sample. With $\alpha_i = 0$ we are ignoring new sample and we keep unaltered our estimate. With $\alpha_i = 1$ we forget the previous estimate and we fully rely on the new sample. With α_i between zero and one we can decide how much we want to rely on the past or on the new sample. Keep in mind this concept, because it will be used for the temporal difference approach. TD are a model-free methods which learn directly from the episodes experience. Differently from MC, it can use incomplete episodes to make guesses about the value functions. To do this, we have to estimate the return from a given state. For MC we have

$$V(s_t) \leftarrow V(s_t) + \alpha(\mathbf{v_t} - V(s_t))$$

For TD we simply substitute v_t with $r_{t+1} + \gamma V(s_{t+1})$

$$V(s_t) \leftarrow V(s_t) + \alpha(\mathbf{r_{t+1}} + \gamma \mathbf{V}(\mathbf{s_{t+1}}) - V(s_t))$$

Doing so, we can use partial rollout of an episodes, because we don't need the entire episode to estimate r_t. This practice is called bootstrapping. $r_{t+1} + \gamma V(s_{t+1})$ is called **TD target**. $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ is called **TD error**.

Note - MC & TD rewriting We can observe that the MC and TD value function updates can be rewritten as we have seen in proposition 1.1. For example for MC

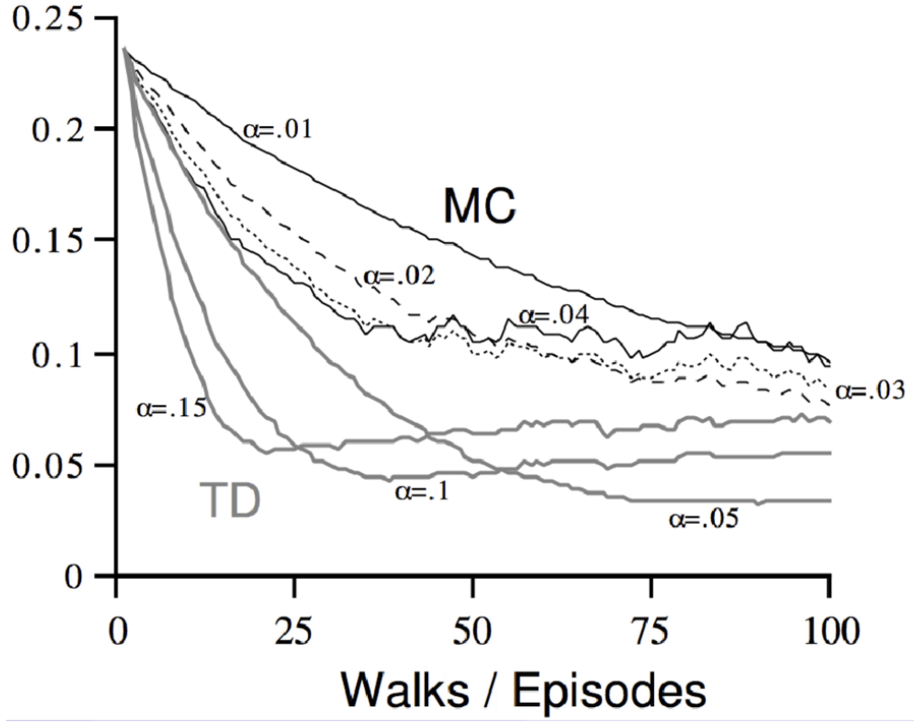
$$\begin{aligned} V(s_t) &\leftarrow V(s_t) + \alpha(v_t - V(s_t)) \\ &\leftarrow V(s_t) + \alpha v_t - \alpha V(s_t) \\ &\leftarrow (1-\alpha)V(s_t) + \alpha v_t \end{aligned}$$

As we have seen before, α balances the importance of new data versus old estimations.

We are sure that TD is a fairly biased, but consistent estimator. The high bias generates from bootstrapping, because at the beginning of the learning phase we rely on random, and so biased, estimates of the value-functions to estimate the return v_t.

Note - MC vs TD comparison

- **Episode termination** TD can learn online after every step. MC must wait until end of episode before return is known. TD can be use in continuing(non–terminating) environments. MC only works for episodic(terminating) environments
- **Bias-Variance** TD has low variance, some bias. MC has high variance, zero bias. TD is more used when the problem is Markovian. MC returns in non-Markovian problems.

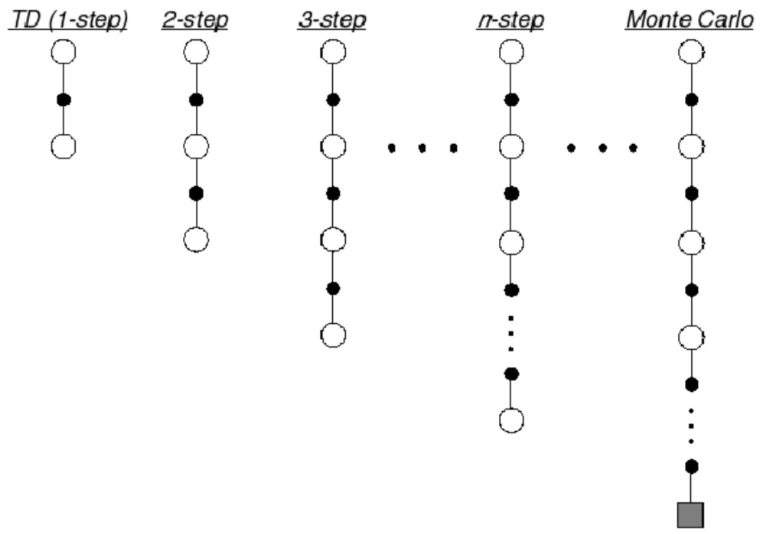


RMS error averaged over states

From the figure above we can observe some interesting properties. TD converges faster and has better results. The learning rate influences the convergence speed. Higher values will make the estimator converge faster, but the error will be higher. In TD this is due to the introduction of bias when we rely too much on new samples. When α is too large we are estimating the value function of a state, with another biased estimate of another state. This can clearly introduce some problems. MC suffers as well as TD with too high learning rates. The largest problem arises because of the large variance. We can see how with $\alpha = 0.04$, MC becomes very noisy.

1.1.3 TD(λ)

We have seen how MC and TD have opposite characteristics in terms of bias-variance. It would be nice if we could choose which properties our algorithm should have. To do so, we can use TD(λ). With the hyperparameter $\lambda \in [0, 1]$, we can swing between TD with $\lambda = 0$ and MC with $\lambda = 1$.



Circles are states and dots are actions

Before introducing formally the TD(λ) equation we have to define the n-step returns that we see in the figure above. For $n = 1, 2, \ldots, \infty$ the n-step return is

$$v_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}) \tag{4}$$

The n-step return is composed by n immediate rewards, plus the estimate of the state-value function in the upcoming state. If we replace this estimator in the TD learning we get

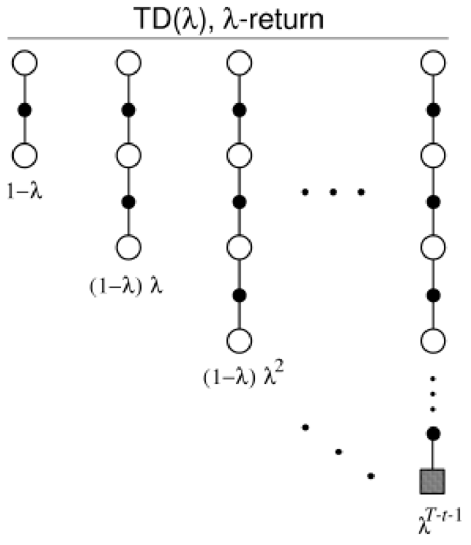
$$V(s_t) \leftarrow V(s_t) + \alpha(v_t^{(n)} - V(s_t))$$

This isn't the TD(λ) formula, but it gives us an intuition about the procedure. This approach has a problem. To calculate the n-step return we must perform n steps. We can't update our estimate of the state-value function at each step. To solve it, we can average the n-step returns after each step, to combine information from all steps. In practice, we perform a weighted average called λ-return v_t^λ. Using weight $(1-\lambda)\lambda^{n-1}$,

$$v_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} v_t^{(n)} \tag{5}$$

We can use this new estimator in the TD learning formula

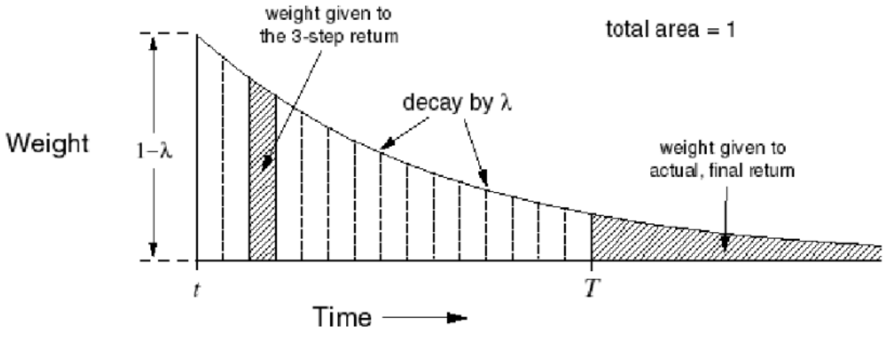
$$V(s_t) \leftarrow V(s_t) + \alpha(v_t^\lambda - V(s_t)) \tag{6}$$



This approach is called **Forward-view** TD(λ). We can see that for $\lambda = 0$, only the 1-step return have a weight different from zero. For $\lambda = 1$, only the complete episode estimate have a weight different from zero. For values in between, we can give more importance to returns with fewer steps or give more weight to n-step returns with more steps. We can manage the bias-variance tradeoff by changing λ.

Note - Weights As one can imagine, the weight formulation is very important. One of the most important property is that the weight must sum to 1. In our case, the sum of the weight is

$$
\begin{aligned}
WeightSum &= (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1} \\
&= \underbrace{\sum_{n=1}^{\infty}(1-\lambda)\lambda^{n}}_{\text{geometric series}} = 1
\end{aligned}
$$

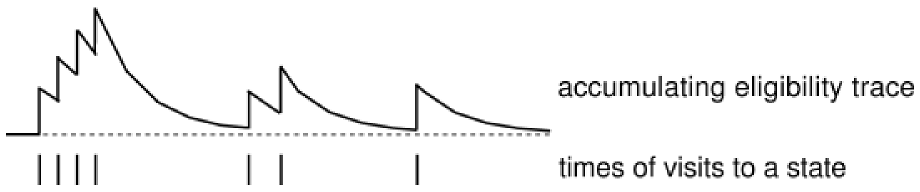


To recap, we want to estimate the state-value function of a state. Starting from the given state, at each step, we perform an action, stretching the trajectory length. Every step, we calculate the n-step return. To estimate the starting state value function, we average over the n-step returns obtained at each step, weighted with the hyperparameter λ. With this approach(forward-view), we still need to perform every step until we reach the complete episodes, because v_t^λ still need the n-step return associated to the complete episode. To solve this problem we can use the so called **Backward-view** TD(λ).
The forward-view provide us some the theory necessary to approach the problem. Backward-view gives us the mechanism. Our objective is to generate an algorithm capable of updating our estimate of the value function at each step. Our first step is to define the **eligibility traces**. The eligibility traces are used to solve the credit assignment problem, which consist in associating the merit of a given reward to a state. Given a trajectory, which states have influenced the most the reward? The eligibility traces are one of the many heuristic for solving the problem,

- **Frequency heuristics** We assign credits to the most frequent state
- **Recency heuristics** We assign credit to the most recent states
- **Eligibility traces** We combine the frequency and recency heuristic[5]

$$e_{t+1}(s) = \underbrace{\gamma\lambda e_t(s)}_{\text{recency term}} + \underbrace{1(s = s_t)}_{\text{frequency term}} \tag{7}$$

The eligibility trace is calculated in every state, and is update at each step. The frequency term makes the trace decay over time because γ(discount factor) and λ(TD hyperparameter) are less than one. The recency term adds one to the trace every time the state is visited.



We have seen how the eligibility trace $e_t(s)$ measure the contribution of state s to the return. We can rewrite the state-value function including this term. For TD we had

$$\begin{aligned} V(s_t) &\leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) && \text{TD} \\ &\leftarrow V(s_t) + \alpha(\delta_t) \\ V(s_t) &\leftarrow V(s_t) + \alpha(\delta_t e_t(s)) && TD(\lambda) Backward-view \end{aligned}$$

TD(λ) uses the eligibility traces to give more importance to the most recently or more frequently visited states. The states with high eligibility trace will rely more on newly seen data, by giving a boost to the learning rate.

[5]+1($s = s_t$) means that we add one only if the current state s is equal to s_t

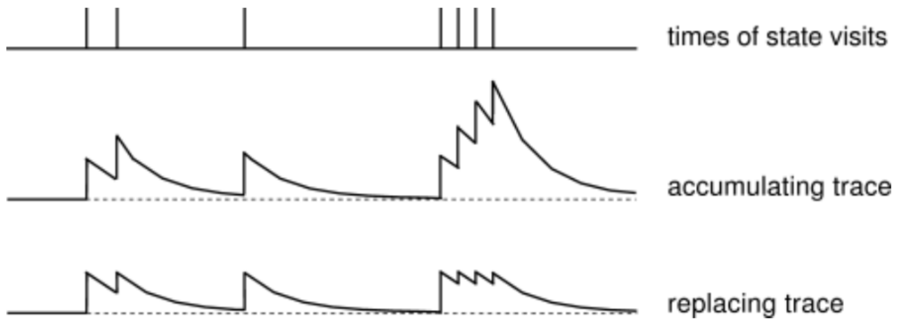
Here we have the algorithm for backward-view TD(λ)

Algorithm 1: backward-view TD(λ)

```
Output    : $V^\pi$
Input     : S, A, $\pi$, $\gamma$, $\lambda$
Initialize: V(s) arbitrarily
for all episodes do
    $e(s) \leftarrow 0, \forall s \in S$
    $s \leftarrow$ starting state
    repeat
        $a \leftarrow$ action given by $\pi$ for $s$
        Take action $a$, observe $r$, and next state $s'$
        $\delta \leftarrow r + \gamma V(s') - V(s)$
        $e(s) \leftarrow e(s) + 1$
        for all $s \in S$ do
            $V(s) \leftarrow V(s) + \alpha\delta e(s)$
            $e(s) \leftarrow \gamma\lambda e(s)$
        end
        $s \leftarrow s'$
    until $s$ is terminal;
end
```

Using accumulating traces, frequently visited states can have eligibilities greater than one. This can be a problem for convergence. **Replacing traces**, instead of adding 1 when you visit a state, set that trace to 1.

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s), & \text{if } s \neq s_t \\ 1, & \text{if } s = s_t \end{cases} \tag{8}$$



1.2 Model-free control

So far we have seen how to estimate the value function of a given policy. Now our objective is to find the policy π^*, that maximize the value function. We already done this in the previous chapter, under the assumption that we know the transition model of our problem. This is no longer true, so we need to modify our algorithms to accommodate this new situation.

1.2.1 On-Policy-Monte-Carlo control

This approach is based on policy iteration. As a reminder, policy iteration is divided in two steps, which are iterated until convergence. The first is policy evaluation. In this step we estimate the value function of our policy. Once we have an estimate, we can perform the second step, policy improvement. From the value function estimate, we can calculate the relative greedy policy, which we are sure is better then the previous policy. We iterate this two step until two consecutive policy evaluation are the same. The relative policy is the optimal one.
The policy evaluation step uses the transition model of the MDP. To solve this problem, we can use one of the model-free prediction algorithm we have seen in the previous section. For example we can use MC. We are done right? Not so fast. Now we can evaluate a policy model-free, but the policy improvement still needs the transition model, because it need to calculate the greedy policy.

$$\pi'(s) = \underset{a\in A}{argmax}\left\{ R(s,a) + \gamma \sum_{s'\in S} \mathbf{P}(\mathbf{s'}|\mathbf{s},\mathbf{a})V(s') \right\}$$

We can do policy improvement model-free by using the action-value function

$$\pi'(s) = \underset{a\in A}{argmax}\left\{ Q(s,a) \right\} \tag{9}$$

In this way, we don't need anymore the transition model, because the greedy policy can be calculated by choosing the action that maximizes the action-value function in a state. In the policy evaluation step, we are no longer estimating V, but we have to calculate the action-value function $Q = Q^{\pi}$. We estimate the value of each state-action pair. Obviously the number of state-action pair are way more than the number of states, so we need to estimate more values than before. To compensate, we need to get more samples. Are we good? Not really unfortunately. Suppose to start our policy iteration with the random policy. We perform the policy evaluation step, and then we calculate the new greedy policy. Usually the greedy policy is deterministic. It means that in every state, we will always choose only one action. This is a problem, because to estimate Q we need to explore all the state-action pair, but being deterministic we take always the same action in a given state. It happens that some state-action pairs will be never explored, and so we don't have a clue on the utility of the pair. The major consequence of this is that we can't use the greedy policy improvement as we have seen before, and furthermore we can't use deterministic policies using policy

iteration to estimate Q. Here we can start to see one of the most important concept in RL. The **exploration vs exploitation dilemma**. This concept will be explained better in the next chapter, for now we say that when learning the value function of an MDP, we need to find the right balance between exploring the state-action pairs and exploiting the knowledge collected so far.
One of the easiest way to produce non-deterministic policies during the policy improvement step, is to use ϵ **greedy exploration**. The policy will be calculated as follow

$$\pi(s,a) = \begin{cases} \frac{\epsilon}{m} + 1 - \epsilon & \text{if } a^* = \underset{a \in A}{argmax}\{Q(s,a)\} \\ \frac{\epsilon}{m} & \text{otherwise} \end{cases} \tag{10}$$

Simplest idea for ensuring continual exploration. All the m actions have non-zero probability of being chosen. The greedy action have a large probability of being chosen. ϵ manages the exploration-exploitation dilemma. Larger values of ϵ will generate policies which explore more, because it gives more probability to the non-greedy actions. The very good thing about ϵ-greedy policy is its theoretical guarantees

Theorem 1.1 (ϵ-greedy policy improvement)**.** *For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to Q^π is an improvement*

$$\begin{aligned} Q^\pi(s,\pi'(s)) &= \sum_{a \in A} \pi'(a|s) Q^\pi(s,a) \\ &= \frac{\epsilon}{m} \sum_{a \in A} \big(Q^\pi(s,a)\big) + (1-\epsilon) \max_{a \in A} Q^\pi(s,a) \\ &\geq \frac{\epsilon}{m} \sum_{a \in A} \big(Q^\pi(s,a)\big) + (1-\epsilon) \sum_{a \in A} \frac{\pi(a|s) - \frac{1}{\epsilon}}{1-\epsilon} Q^\pi(s,a) \\ &= \sum_{a \in A} \pi(a|s) Q^\pi(s,a) = V^\pi(s) \end{aligned}$$

Therefore from policy improvement theorem, $V^{\pi'}(s) \geq V^\pi(s)$

This is great news, the ϵ-greedy policy with respect to a policy is for sure better. To recap we have,

- **Policy evaluation** MC policy evaluation, $Q = Q^\pi$
- **Policy improvement** ϵ-greedy policy improvement

As we have seen for DP policy iteration we don't need to calculate the true value function in the policy evaluation step, but we can stop when our approximation is close enough. In this case we have generalized model-free policy iteration

- **Policy evaluation** MC policy evaluation, $Q \approx Q^\pi$

To completely define on-policy MC control, we still need some definitions.

Definition 1.1 (GLIE)**.** *We say that an exploration strategy is Greedy in the Limit of Infinite Exploration(GLIE) if*

- *All state-action pair are explored infinitely many times*
$$\lim_{k\to\infty} N_k(s,a) = \infty, \quad \forall(s,a)$$
- *The policy converges on a greedy policy*
$$\lim_{k\to\infty} \pi_k(a|s) = \mathbf{1}(a = \underset{a'\in a}{argmax}\{Q_k(s',a')\})$$

In GLIE MC control we have. Given sample k^{th} episode using $\pi : \{s_1, a_1, r_1, \dots, s_T\} \sim \pi$, for each state and action in the episode we have

$$N(s_t,a_t) \leftarrow N(s_t,a_t) + 1$$
$$Q(s_t,a_T) \leftarrow Q(s_t,a_t) + \frac{1}{N(s_T,a_t)}(v_t - Q(s_t,a_t))$$

The policy improvement step based on the new action-value function will use

$$\epsilon \leftarrow \frac{1}{k}$$
$$\pi \leftarrow \epsilon - greedy(Q)$$

Theorem 1.2 (GLIE Monte-Carlo)**.** *GLIE Monte-Carlo converges to the optimal action-value function, $Q(s,a) \to Q^*(s,a)$*

Now we have a way to find the optimal policy with a model-free approach.
This method have many hyperparameters which are related to some time scales. We can recap their role and their relationship in order to make things clearer. There are three time scales,

- **Behavioural** related to the discount factor γ. $\frac{1}{1-\gamma}$
- **Sampling** related to the learning rate α for the estimation of Q
- **Exploration** related to ϵ, for the ϵ-greedy strategy

To have good result during the learning phase, these three hyperparameters have to respect this relationship

$$1-\gamma \ll \alpha \ll \epsilon$$

As a starting point we can use $1-\gamma \approx \alpha \approx \epsilon$. Then we decrease ϵ faster than α. We do so because if the learning rate is too small, while ϵ is still large, the exploration will not be counted. When updating our value function, α will weight more the old samples. This in practice, would ignore the newly explored states and actions. Practically, given M trials we can set $\alpha \sim 1 - \frac{m}{M}$ and $\epsilon \sim (1-\frac{m}{M})^2$. γ should remains constant, because it is a property of the problem we are trying to solve, and not of the learning algorithm we are using. But in practice, γ is initialized to low values, because the problem becomes easier. Then it is gradually moved towards its correct value. This approach is called curriculum learning.

1.2.2 On-policy Temporal-Difference control

As we did before, we can use temporal difference to solve the policy evaluation step. This approach have, as in the prediction case, a lot of advantages compared to MC. It has lower variance, it's online and it can work with incomplete sequences. The most simple way to adapt the control problem using TD is by applying temporal difference to Q in the policy evaluation step. Then, we can use ϵ-greedy policy improvement. These steps are done every time we take a new action.
This on-policy control is called **SARSA**[6]. As in the MC case we need to evaluate the action-value function

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

At each time-step we perform

- **Policy evaluation** SARSA, $Q \approx Q^\pi$
- **Policy improvement** ϵ-greedy improvement

Algorithm 2: SARSA On-Policy control

```
Output    : $Q^*$
Input     : S, A, $\pi_0$, $\gamma$, $\epsilon$
Initialize: $Q(s,a)$ arbitrarily
            $\pi \leftarrow \pi_0$
do
    $s \leftarrow$ starting state
    $a \leftarrow$ action given by $\pi$ for $s$
    repeat
        Take action a, observe r, s'
        $a' \leftarrow$action given by $\pi$ for $s'$
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$
        $a \leftarrow a'$
        $\pi \leftarrow \epsilon$-greedy($\pi$)
    until $s$ is terminal;
while convergence;
```

To ensure that the SARSA algorithm converges to the optimal solution, we have to check if some conditions are respected

[6]SARSA stands for: State Action Reward State Action

Theorem 1.3. *SARSA converges to the optimal action-value function, $Q(s,a) \to Q^*(s,a)$, under the following conditions*

- *GLIE exploration strategy(sequence of policies $\pi_t(s,a)$)*
- *Robbins-Monro sequence of step-sizes α_t*

$$\sum_{t=1}^{\infty} \frac{1}{\alpha_t} = \infty$$
$$\sum_{t=1}^{\infty} \frac{1}{\alpha_t^2} \leq \infty$$

We can see that for SARSA we are using TD(0), because we are bootstrapping after a single step. As we did before for the prediction problem, we can define a procedure which can swing across MC and TD with a parameter λ. This procedure is called SARSA(λ)

- **Forward view** update action–value $Q(s,a)$ to λ-return v_t^λ
- **Backward view** use eligibility traces for each state-action pairs.

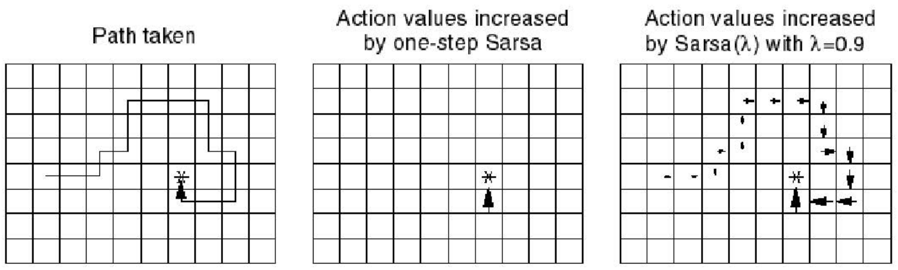
$$e_t(s,a) = \gamma\lambda e_{t-1}(s,a) + \mathbf{1}((s_t,a_t) = (s,a))$$

Algorithm 3: Backward-view SARSA(λ)

```
Output    : $Q^*$
Input     : S, A, $\pi_0$, $\gamma$, $\lambda$, $\epsilon$
Initialize: Q(s,a) arbitrarily
            $\pi \leftarrow \pi_0$
do
    $e(s,a) \leftarrow 0, \forall s,a$
    $s \leftarrow$ starting state
    $a \leftarrow$ action given by $\pi$ for $s$
    repeat
        Take action $a$, observe $r$, and next state $s'$
        $a' \leftarrow$ action given by $\pi$ for $s'$
        $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$
        $e(s,a) \leftarrow e(s,a) + 1$
        for all $s,a$ do
            $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
            $e(s,a) \leftarrow \gamma\lambda e(s,a)$
        end
        $s \leftarrow s'$
        $a \leftarrow a'$
        $\pi \leftarrow \epsilon$-greedy($\pi$)
    until s is terminal;
while convergence;
```

The preferred method for what we have seen before is the backward view.

Example - SARSA(λ) Imagine to have the gridworld in the figure below


Path taken
Action values increased by one-step Sarsa
Action values increased by Sarsa(λ) with λ=0.9

From left to right: 1.Path taken 2.SARSA(0) 3.SARSA(0.9)

Suppose to perform a complete episode of our MDP and we have estimated the action-value function with SARSA(0) and SARSA(0.9). The first corresponds to the TD approach. In fact, we update only the state-action pair right before the goal state, which is the only one returning a non-zero reward. All the other pairs are not updated because the goal return echoes only to the previous pair on the trajectory. On the other hand, With $\lambda = 0.9$, the goal return is echoed through all the trajectory. We can also observe that the update decay over the trajectory, as we would expect.

1.2.3 Off-Policy learning

Off-policy methodologies try to estimate a policy that is different from the one generating the data. The estimated policy is called **target policy** $\pi(a|s)$, while the data generating one is called **behavior policy** $\bar{\pi}(a|s)$. This could seem counter intuitive, but it can bring a lot of advantages to our learning phase. For example, splitting the two policy can give us an advantage when learning from observing humans or other agents, or reuse experience generated from old policies($\pi_1, \ldots, \pi_{t-1}$). How can we estimate a policy from another one? We can use the concept of **importance sampling**. This method is used in general to estimate the expectation of a different distribution w.r.t. the distribution used to draw samples. Imagine to have a generic function $f(x)$. We sample $f(x)$ using a distribution p[7]. To denote that the samples x are drawn from p we use $x \sim p$. The expected value of $f(x)$ can be written as $E_{x\sim p}[f(x)]$. Our objective is to find $E_{x\sim p}[f(x)]$ by sampling x with another distribution q. We can manipulate the expected value to achieve this.

$$\begin{aligned} E_{x\sim p}[f(x)] &= \int p(x)f(x)dx \\ &= \int q(x)\left(\frac{p(x)}{q(x)}f(x)\right)dx \end{aligned}$$

[7]Note that $f(x)$ is simply a generic function, for which we want to find the expected value. We want to estimate the expected value through sampling. To sample from the input space(x), we must use a probability distribution, in our case p

$$= E_{x\sim q}\left[\frac{p(x)}{q(x)} f(x)\right]$$

We can see in the second line of the procedure, that we can consider as a new generic function $\frac{p(x)}{q(x)} f(x)$. Seeing that, we have an integral of a function, multiplied by a distribution q. This is by definition an expected value, with the samples drawn from q. In this way we can estimate the expected value of $f(x)$, by sampling from another distribution. The ratio $\frac{p(x)}{q(x)}$ is also called **importance weight** $W(x)$. We can guess that this ratio weights the importance of the given sample. When p gives a big importance to a region, and q doesn't, we must amplify the sample, if we have taken it from q. In p we sample frequently a given region. In q we sample rarely, but when applying importance sampling every sample from that region have a large weight to compensate. In fact when $p(x)$ is large and $q(x)$ is small $W(x)$ becomes bigger. Importance sampling is like doing a weighted average, where the samples are weighted according to their importance.

In our RL problem, we can use it at our advantage. If we consider as our generic function the return v_t, and as sampling distributions the target policy π and behavior policy $\bar{\pi}$, we can apply importance sampling[8]. Assume to use a MC approach, so we estimate the return with the whole trajectory. Using $\bar{\pi}$, we obtain at the end of an episode the return v_t, Then, we weight using importance sampling the return obtained obtaining v_t^{μ}.

$$v_t^{\mu} = \frac{\pi(a_t|s_t)\pi(a_{t+1}|s_{t+1})}{\bar{\pi}(a_t|s_t)\bar{\pi}(a_{t+1}|s_{t+1})} \cdots \frac{\pi(a_T|s_T)}{\bar{\pi}(a_T|s_T)} v_t \tag{11}$$

Once we obtained the re-weighted return, we can update the action-value function using v_t^{μ}

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(v_t^{\mu} - Q(s_t, a_t)) \tag{12}$$

The behavior policy must be chosen very wisely. It has to be very close to the target policy, otherwise the estimation variance grows dramatically. In particular we must use a $\bar{\pi}$ that is zero where π is zero.

We can also use importance sampling with SARSA. In this case we get much better results compared to MC. We use TD targets generated from $\bar{\pi}$ to evaluate π. In practice with importance sampling, we weight the TD target $r + \gamma Q(s', a')$ according to the similarity between the two policies. As we have seen before, with SARSA we perform a step and then we evaluate the action-value function. Doing so, we only need a single importance sampling correction at each step

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_{t+1} + \gamma \frac{\pi(a_{t+1}|s_{t+1})}{\bar{\pi}(a_{t+1}|s_{t+1})} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right) \tag{13}$$

With SARSA we have much lower variance than Monte–Carlo importance sampling and policies only need to be similar over a single step.

[8] $f(x) \to v_t$, $p(x) \to \pi$, $q(x) \to \bar{\pi}$

Off–Policy Control with Q–learning Q-learning is the most used RL algorithm. It's very similar to SARSA,

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a' \in A}\{Q(s',a')\} - Q(s,a)) \tag{14}$$

We can see how the only difference is the usage of the max operator in the TD target. This little difference makes a lot of difference. First of all, the algorithm becomes off-policy, because the next action a' in the TD target is decided using the greedy policy in the current state, and not using the policy π. Most importantly, we can demonstrate that Q-learning converges to the optimal Q^* even when the GLIE condition is not respected. This means that we don't have to stop exploring in order to converge to the optimal solution. The algorithm is so solid that, even if we use always the random policy the algorithm will converge to the optimal solution. As SARSA can be seen as the model-free version of policy iteration, Q-learning can be seen as the model-free version of value iteration. In fact, we can see how Q-learning uses the optimal Bellman operator over the single step. Note that, to choose the actual action a in a given state we use the policy π derived from the ϵ-greedy improvement. To estimate the next action a' we use the greedy policy of π.

Algorithm 4: Q-learning

```
Output    : $Q^*$
Input     : S, A, $\pi_0$, $\gamma$, $\epsilon$
Initialize: Q(s,a) arbitrarily
            $\pi \leftarrow \pi_0$
do
    $s \leftarrow$ starting state
    repeat
        $a \leftarrow$ action given by $\pi$ for $s$
        Take action $a$, observe $r$, and next state $s'$
        $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a' \in A}\{Q(s',a')\} - Q(s,a))$
        $s \leftarrow s'$
        $\pi \leftarrow \epsilon$-greedy($\pi$)
    until $s$ is terminal;
while convergence;
```