Lecture 6: Temporal-Difference Learning

Tianpei Xie

Aug 5th., 2022

Contents

1 Introduction

If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-dierence (TD) learning [Sutton and Barto, 2018]. ***Temporal-Difference (TD) Learning*** is a combination of Monte Carlo ideas and dynamic programming (DP) ideas. Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environments dynamics. Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap). The relationship between TD, DP, and Monte Carlo methods is a recurring theme in the theory of reinforcement learning.

Note that TD, DP, and Monte Carlo *control* all use the same Generalized Policy Iteration (GPI) methods to improve policy. The differences in the methods are primarily differences in their approaches to the *prediction* problem.

Another thing to note is that all of methods we discussed above is Tabular method, since we do not assume any functional form for the value function but just record it into a table.

2 Temporal-Difference Prediction

Recall that in Monte Carlo Methods, the value function is estimated by sample episodes. We can write it into an *incremental update*. Let $V(s)$ be the (empirical) estimate of state-value $v_\pi(s)$,

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi\left[G_t | S_t = s\right] \\
&\approx \mathbb{E}_{\text{emp}}\left[G_t | S_t = s\right] := V_{N(s)}(s) \\
&= \frac{\sum_{t=1}^{N(s)} G_t}{N(s)} \\
&= \frac{1}{N(s)}\left[G_{N(s)} + (N(s)-1)\frac{\sum_{t=1}^{N(s)-1} G_t}{N(s)-1}\right] \\
&= \frac{\sum_{t=1}^{N(s)-1} G_t}{N(s)-1} + \frac{1}{N(s)}\left[G_{N(s)} - \frac{\sum_{t=1}^{N(s)-1} G_t}{N(s)-1}\right] \\
V_{N(s)}(s) &= V_{N(s)-1}(s) + \alpha_{N(s)}\left[G_{N(s)} - V_{N(s)-1}(s)\right]
\end{aligned}$$

where $N(s) = |T(s)|$ and $T(s) = \{S_t = s\}_{t=1}^{\infty}$ or the number of epsiodes for first-visit MC. The above incremental update can be summarized as

$$\begin{gathered}
V(S_t) \leftarrow V(S_t) + \alpha_t\left(G_t - V(S_t)\right) \qquad (1) \\
\textbf{new} \text{ estimate} \leftarrow \textbf{old} \text{ estimate} + \text{step\_size}\,(\textbf{target} - \textbf{old} \text{ estimate})
\end{gathered}$$

This formula defines the value function using two terms: the **old estimate**, and the **difference** between the target and the old estimate. We call this algorithm *constant-α Monte Carlo.*

The Temporal-Difference (TD) learning extends the formula in (1) by replacing the new estimate/rewards with the new estimates *in next step* and the rewards; $G_t = R_{t+1} + \gamma G_{t+1}$. Then TD replaces the estimate of cumulative rewards at $t+1$, G_{t+1}, with the value function $V(S_{t+1})$ at

```
Tabular TD(0) for estimating $v_\pi$

Input: the policy $\pi$ to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
   Initialize $S$
   Loop for each step of episode:
      $A \leftarrow$ action given by $\pi$ for $S$
      Take action $A$, observe $R$, $S'$
      $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
      $S \leftarrow S'$
   until $S$ is terminal
```

Figure 1: TD(0) algorithm

$t+1$

$$
\begin{aligned}
V(S_t) &\leftarrow V(S_t) + \alpha_t\left(G_t - V(S_t)\right)\\
&\approx V(S_t) + \alpha_t\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right). \qquad (2)
\end{aligned}
$$

This algorithm is called ***one-step TD*** or ***TD(0)***. It is a special case of *multi-step TD or TD(λ)*. The approximation of target $\underline{G_t \approx R_{t+1} + \gamma V(S_{t+1})}$ is the key for TD learning. It comes from Bellman equation combining with Monte Carlo estimation:

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi\left[G_t | S_t = s\right] \approx \mathbb{E}_{\text{emp}}\left[G_t | S_t = s\right]\\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} | S_t = s\right]\\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma\, v_\pi(S_{t+1}) | S_t = s\right], \text{ (Bellman equation)}\\
&\approx \mathbb{E}_{\text{emp}}\left[R_{t+1} + \gamma\, v_\pi(S_{t+1}) | S_t = s\right], \text{ (Monte Carlo estimate)}\\
\text{thus } G_t &\approx R_{t+1} + \gamma v_\pi(S_{t+1})\\
&\approx R_{t+1} + \gamma V(S_{t+1}). \qquad (3)
\end{aligned}
$$

The last approximation is to replace v_π with its estimate V. Whereas Monte Carlo methods must wait until the *end of the episode* to determine the increment to $V(S_t)$ (only then is G_t known), TD methods need to wait only until the *next time step*. The quantity $\underline{R_{t+1} + \gamma V(S_{t+1})}$ is referred as the ***target*** (instead of G_t for Monte Carlo methods.) and $\underline{\delta_t := R_{t+1} + \gamma V(S_{t+1}) - V(S_t)}$ as the Temporal-Difference error or, ***TD error***. Figure 1 shows the TD(0) algorithm

Let us compare TD learning to Dynamic Programming (DP) and Monte Carlo (MC). Because TD(0) bases its update in part on an existing estimate, we say that it is a *bootstrapping* method, like DP. Check the value function and Bellman equation as

$$
\text{DP: } v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma\, v_\pi(S_{t+1}) | S_t = s\right] \qquad (4)
$$

$$
\text{MC: } v_\pi(s) = \mathbb{E}_\pi\left[G_t | S_t = s\right] \qquad (5)
$$

DP estimate uses an estimate of the Bellman equation (4) as target. Although the expectation can be computed exactly, the value at state S_{t+1} is an estimate since we do not know S_{t+1} at time t. The MC uses an estimate of (5) as target. The Monte Carlo target is an estimate because the expected value is not known; a *sample return* is used in place of the real expected return. The TD

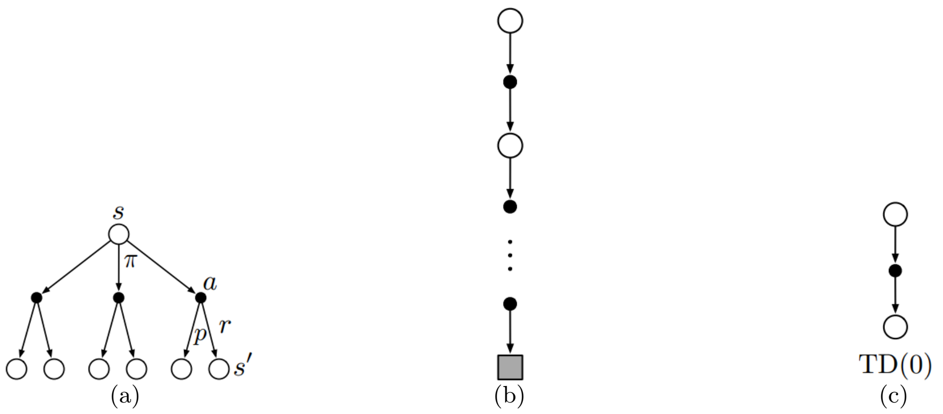

Figure 2: Backup diagram for (a) Dynamic Programming (DP); (b) Monte Carlo (MC) methods; (c) Temporal-Difference (TD(0)) learning. Note that both DP and TD used boostrapping to compute state-value based on previous estimate of state-value function. MC only updates at the end of episode.

target is an estimate for *both reasons*: it *samples the expected values* in (4) and it uses the current estimate V instead of the true v_π. Thus, TD methods combine the **sampling** of Monte Carlo with the **bootstrapping** of DP.

Figure 2 compare the update of DP, MC and TD in backup diagram. We see that both DP and TD replies on bootstrapping to compute estimate based on existing estimate. On the other hand, both MC and TD updates as *sample updates* because they involve looking ahead to a *sample successor state* (or state-action pair), using the value of the successor and the reward along the way to compute a backed-up value, and then updating the value of the original state (or state-action pair) accordingly. This is different from *expected update* from DP, which uses the complete knowledge of dynamics p.

The TD-error δ_t measuring the difference between the *estimated value* of S_t and the *better estimate*/target $R_{t+1} + \gamma V(S_{t+1})$. δ_t is the error of estimation in $V(S_t)$, available at time $t+1$. δ_t will not be available until $t+1$.

$$\delta_t := R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{6}$$

We can actually rewrite the error for Monte Carlo estimate using δ_t

$$\begin{aligned}
G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \\
&= \delta_t + \gamma \left(G_{t+1} - V(S_{t+1})\right) \\
&= \delta_t + \gamma \delta_{t+1} + \gamma^2 \left(G_{t+2} - V(S_{t+2})\right) \\
&\ldots \\
&= \sum_{k=t}^{T-1} \gamma^{t-k} \delta_k
\end{aligned}$$

If S_{t+1} is terminal, then the TD-error is $\delta_t := R_{t+1} - V(S_t)$ since $V(\text{terminal}) = 0$.

2.1 Advantages of TD Prediction Methods

The advantages of temporal difference method TD(0) over dynamic programming and Monte Carlo methods can be summarized as below:

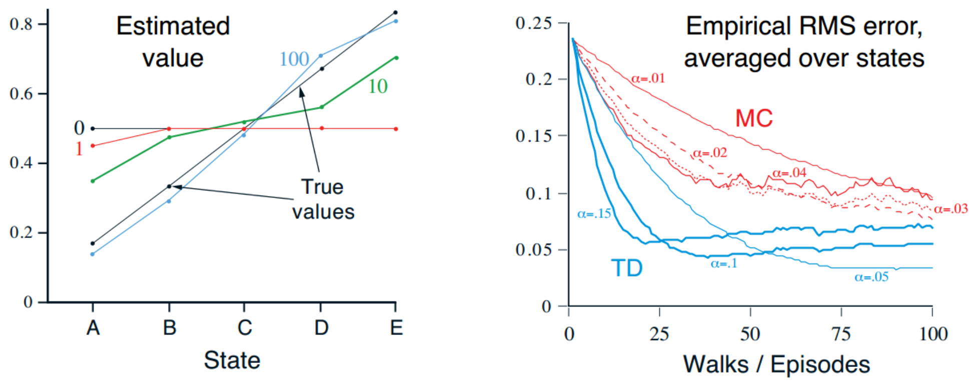

Figure 3: An example from [Sutton and Barto, 2018] based on random walk Markov decision process. It can be seen that TD converge faster than MC since it updates value estimate faster at each time step.

- Both Monte Carlo, TD(0) are ***sample-based learning algorithm***. They do not require complete knowledge of *dynamics* p. DP, however, requires a *model*.
- TD(0) learning naturally is an ***online learning algorithm***, which updates the value estimate at end of each time step. On the other hand, Monte Carlo algorithm only updates at *the end of episode* so that it can observe the return value. Because of this, TD(0) updates information based on observed rewards faster than MC. MC is very *slow* esp. when one episode is long. Some Monte Carlo methods must ignore or discount episodes on which experimental actions are taken, which can greatly slow learning.
- TD(0) learning also has ***convergence guarantee***. For any fixed policy π, TD(0) has been proved to converge to v_π, in the *mean* for a constant step-size parameter if it is sufficiently small, and *with probability* 1 if the step-size parameter decreases according to the usual stochastic approximation conditions. In that sense, both TD and Monte Carlo methods converge *asymptotically* to the correct predictions.
- Both DP and TD(0) use ***bootstrapping***. They learn aguess from a guess, i.e. they update the value estimate of a state based on value estimate of its successor state plus the rewards after it. In other words, This helps to *reduce the variance* of estimate compared to MC methods.

We can also compare with unsupervised learning and supervised learning:

- Like MC and DP, TD prediction has a *target*. On the other hand, all of these methods are ***unsupervised*** since no human label is needed. Just need to wait for the target observation. In that sense, it is more *scalable*, and *model-free*. TD learning is a method for ***learning to predict***. In particular, it learns a prediction based on *another, later, learned* prediction. **TD error** is the difference between two predictions $R_{t+1} + \gamma V(S_{t+1})$ and $V(S_t)$. In this sense, it is very similar to **supervised learning**. Like *back-propagation the error* in supervised learning, TD learning also predict the value backwards from the terminal state.
- TD learning is relevent only when we consider ***multi-step*** *prediction*. The thing predicted (the discounted cumulative rewards conditional on some behavior) is multi-step in the future. This is very different from supervised learning, which makes a one-step prediction only.

```
Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Loop for each step of episode:
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

Figure 4: Sarsa, on-policy TD control

3 Temporal-Difference Control

3.1 Sarsa: On-policy TD Control

TD prediction methods can be used for the control problem. Note that in Generalized Policy Iteration (GPI), the policy improvement is obtained by choosing greedy/ϵ-greedy policy with respect to the action-value function. ***Sarsa algorithm*** only substitute the TD methods for the evaluation or prediction part in the GPI.

Since we need to estimate the action-value function using TD learning, we recall the Bellman equation for action-value function q and state-value function v as below.

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma\, v_\pi(S_{t+1}) | S_t = s\right] \\ q_\pi(s, a) &= \mathbb{E}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a\right] \end{aligned} \tag{7}$$

Similar to how TD prediction is derived from the equation of state-value function, Sarsa derive the TD prediction from the equation of action-value function.

$$\begin{aligned} V(S_t) &\leftarrow V(S_t) + \alpha_t\left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]. \\ Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha_t\left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]. \end{aligned} \tag{8}$$

As in (8), we see that the estimate of state-value function Q at time t depends on the state-value function estimate at $t+1$, rewards at $t+1$. The name of *Sarsa* comes from the fact that the update is controlled by the tuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$. (unlike the TD prediction, here we need to sample a new action based on the new state and policy.)

Sarsa algorithm is on-policy control, since the policy used to generate actions is the same policy that is updated. 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 form of the Sarsa control algorithm is given in the box on the next page. Like on-policy MC control, we have to balance the exploration-exploitation by choosing π as ϵ-soft. Figure 4 describes the Sarsa algorithm.

The convergence properties of the Sarsa algorithm depend on the nature of the policys dependence on Q. Using ϵ-soft/ϵ-greedy policy, Sarsa converges with probability 1 to an optimal policy and

```
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

Figure 5: Q-learning, off-policy TD control

action-value function as long as all stateaction pairs are visited an infinite number of times and the policy converges in the limit to the greedy policy.

3.2 Q-learning: Off-policy TD Control

One of the early breakthroughs in reinforcement learning was the development of an off-policy TD control algorithm known as ***Q-learning***.

Instead of Bellman equation for q, we now consider the Bellman optimality equation for q_*

$$
\begin{aligned}
q_\pi(s, a) &= \mathbb{E}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a\right] \\
q_*(s, a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a' \in \mathcal{A}(s)} q_*(S_{t+1}, a') \Big| S_t = s,\ A_t = a\right]
\end{aligned} \tag{9}
$$

The update rule for Q-learning is defined by

$$
Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t \left[R_{t+1} + \gamma \max_{a' \in \mathcal{A}(S_{t+1})} Q(S_{t+1}, a') - Q(S_t, A_t)\right]. \tag{10}
$$

As indicated in (9), the learned action-value function, Q, directly approximates q_*, the optimal action-value function, independent of the policy being followed. While Sarsa is a sample-based *policy iteration algorithm*, the *Q-learning* is a sample-based ***value-iteration algorithm***.

Q-learning is an ***off-policy control*** since the target policy learned does not depend on the policy used to generate new action A_{t+1}. In fact, $\max_{a' \in \mathcal{A}(S_{t+1})} Q(S_{t+1}, a')$ implies that the *target policy* is a deterministic greedy policy $\pi'(s) = \text{argmax}_{a'} Q(S_{t+1}, a')$, whereas the behavior policy π is stochastic like ϵ-greedy.

Q approximates the optimal action value function independent of the behavior policy. The behavior policy still has an effect in that it determines which state-action pairs are visited and updated. However, all that is required for *correct convergence* is that all pairs continue to be updated. This is a *minimal requirement* in the sense that any method guaranteed to find optimal behavior in the general case must require it. Under this assumption and a variant of the usual stochastic approximation conditions on the sequence of step-size parameters, Q has been shown to **converge with probability 1** to q_*. Figure 10 describes the Q-learning.

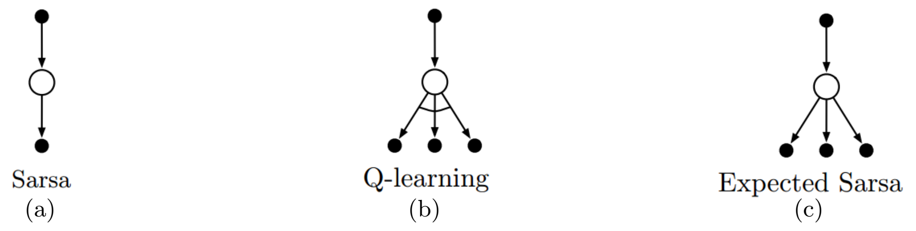

Figure 6: Backup diagram for (a) Sarsa; (b) Q-learning; (c) Expected Sarsa.

3.3 Expected Sarsa: Off-policy TD Control

Expected Sarsa is an algorithm derived from the same Bellman equation (11) as the Sarsa. However, instead of sample the next action A_{t+1}, we can directly computed the exact expected value since we know policy $\pi(a'|\cdot), \forall a'$. Compare to Q-learning, we can replace the maximization in (10) with the expectation.

From the Bellman equation for action-valued function,

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1}) q_\pi(S_{t+1}, a') \Big| S_t = s, A_t = a\right] \end{aligned} \tag{11}$$

Expected Sarsa has the update rule as

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t \left[R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1}) Q(S_{t+1}, a') - Q(S_t, A_t)\right]. \tag{12}$$

Expected Sarsa is an **off-policy control** since the update does not depend on the behavior policy that generates the new action since the expectation is computed for *all* actions. Given the next state, S_{t+1}, this algorithm moves ***deterministically*** in the same direction as Sarsa moves in ***expectation***,

Expected Sarsa is more complex computationally than Sarsa but, in return, it *eliminates the variance* due to the random selection of A_{t+1}. Given the same amount of experience we might expect it to perform slightly better than Sarsa, and indeed it generally does. By evaluating all actions instead of the maximal action, the expected Sarsa would choose actions more safely to avoid potential high loss when choosing the optimal action deterministically.

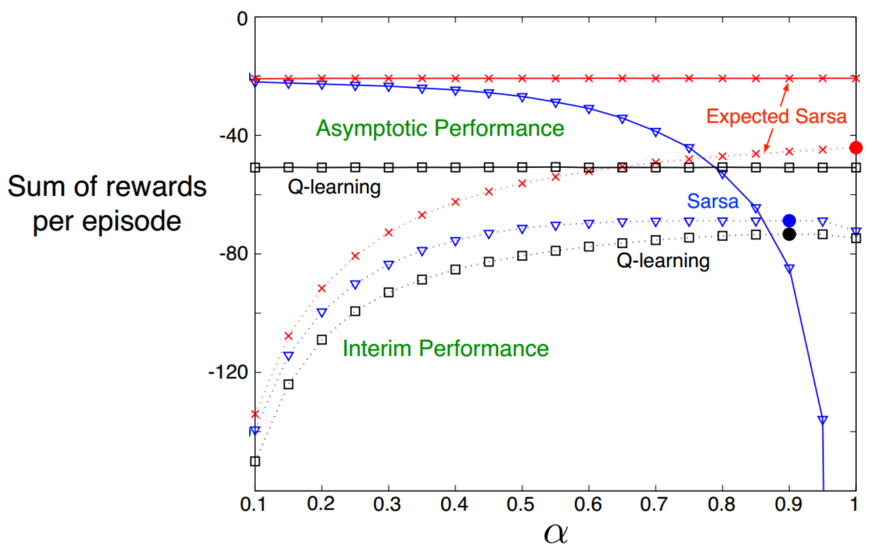

Figure 7: Experiment performance between Sarsa, Q-learning, Expected Sarsa. Note that Sarsa is impacted by the choice of α but Q-learning and Expected Sarsa are not.

4 Summary

We can summarize the update rule for Sarsa, Q-learning and expected Sarsa as a simple update rule, using $\hat{G}_t$ as the approximate return at t:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t \left[\hat{G}_t - Q(S_t, A_t)\right].$$

$$\begin{aligned} \textbf{Sarsa} \quad & \hat{G}_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) \\ \textbf{Q-Learning} \quad & \hat{G}_t = R_{t+1} + \gamma \max_{a' \in \mathcal{A}(S_{t+1})} Q(S_{t+1}, a') \\ \textbf{Expected Sarsa} \quad & \hat{G}_t = R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1}) Q(S_{t+1}, a') \end{aligned}$$

The **key** for TD Control compared to Monte Carlo Control is that $\hat{G}_t$ is *partially* ***computed*** instead of ***observed*** from real experience only. Note that TD Control need as much information from real experience as MC Control but is learning much faster. As compared to functional approximation, TD Control is also ***cheap*** in terms of requirements on computational resources.

References

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.