Reinforcement Learning **Lecture–5**

Model-Free Control

Anubhav Gupta

Given a policy, we now have the knowledge of how to efficiently estimate its value functions. We already know how the dynamic programming methods are used for prediction and control. These methods rely on the underlying MDP for planning.

However, for most of the practical problems, model of the environment is either unknown, or it is too big to use exactly (except by samples). Thus, model-free control methods are needed.

The general principle behind these control algorithms is generalized policy iteration (GPI). Note that DP control algorithms - policy iteration and value iteration were also based on the idea of GPI. In GPI, we maintains an approximate policy and an approximate value function, and these two repeatedly try to improve each other resulting in optimal policy and optimal value function.

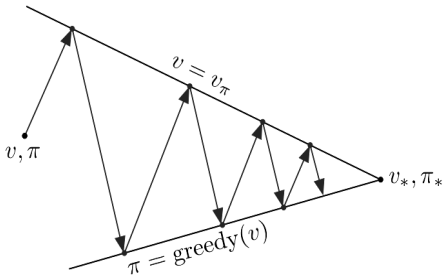

Figure 1: Generalized Policy Iteration

1 Monte Carlo Control

Let's start by extending the idea of policy iteration to Monte Carlo methods. Here, we will perform alternating steps of MC policy evaluation and greedy policy improvement, beginning with an arbitrary policy π_0. Let us make two assumptions for now:

- We observe infinite number of episodes
- Episodes are generated with exploding starts, i.e. all state-action pairs have non-zero probability to occur in some episodes

On thing to note here is that we must use action values q, instead of state values v, since state values, without MDP are not enough to improve the policy.

$$\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\, \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')$$

So, greedy policy is chosen based on action value functions.

$$\pi(s) \leftarrow \underset{a}{\text{argmax}}\, q(s, a)$$

If these conditions are met, MC policy iteration as discussed above converges to optimal policy and action value functions. However, we have made two strong assumptions. Let's relax both assumptions one by one and improve the algorithm.

Relaxing first assumption: If infinite number of episodes are not available, we give up trying to complete policy evaluation before returning to policy improvement. MC policy iteration algorithm alternate between evaluation and improvement on an episode-by-episode basis. This simple algorithm is called Monte Carlo ES (since we still have exploding starts assumption in place). Monte Carlo ES converges to optimal policy.

1.1 Monte Carlo Control without Exploding Starts

Let's now remove the assumption of exploding starts. In this case, if a greedy policy is used in the policy improvement step, then policy evaluation step will only ever evaluate a single action. This is the problem of exploration vs exploitation in reinforcement learning. The approaches to tackle this issue are divided into two categories - on-policy methods and off-policy methods.

In on-policy methods, instead of using a greedy policy over action values, we now use ϵ-greedy policy. An ϵ-greedy policy takes the greedy action with probability $1-\epsilon$ (exploit). Whereas it takes a random action with probability ϵ (exploration). This ensures that all the m action from any state are tried with non-zero probability.

$$\pi_{a|s} = \begin{cases} \epsilon/m + 1 - \epsilon, & \text{if } a = \text{argmax}_{a' \in \mathcal{A}} Q(s, a') \\ \epsilon/m, & \text{otherwise} \end{cases}$$

ϵ-greedy policies are example of ϵ-soft policies, defined as policies for which $\pi(a|s) \geq \frac{\epsilon}{|\mathcal{A}(s)|}$ for all state and action, for some ϵ.

Algorithm 1: On-policy (every-visit) MC control using ϵ-greedy policy for estimating $\pi \sim \pi_*$

Algorithm Parameters: small $\epsilon > 0$
Initialization: Initialize $Q(s,a)$ arbitrarily, $C(s,a) \leftarrow 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ϵ-greedy policy with respect to Q

for *each episode (until convergence)* **do**
  Generate an episode following $\pi : S_0, A_0, R_1, \cdots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  for *each time-step* $t = T-1, T-2, \cdots, 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + 1$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{C(S_t, A_t)}(G - Q(S_t, A_t)$
    for *all* $a \in \mathcal{A}$ **do**
      $\pi(a|S_t) \leftarrow \begin{cases} \epsilon/|\mathcal{A}(\mathcal{S}_t)| + 1 - \epsilon, & \text{if } a = \text{argmax}_{a' \in \mathcal{A}} Q(S_t, a') \\ \epsilon/|\mathcal{A}(\mathcal{S}_t)|, & \text{otherwise} \end{cases}$
    end
  end
end

A complete algorithm of on-policy MC control using ϵ-greedy policy is given by algorithm-1. We will now see that ϵ-greedy policy with respect to q_π is an improvement over any ϵ-soft policy π.

$$\begin{aligned} q_\pi(s, \pi'(s)) &= \sum_{a \in \mathcal{A}} \pi'(a|s) q_\pi(s,a) \\ &= \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s,a) + (1-\epsilon) \max_{a \in \mathcal{A}} q_\pi(s,a) \\ &\geq \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s,a) + (1-\epsilon) \frac{\pi(a|s) - \epsilon/m}{1-\epsilon} q_\pi(s,a) \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a) \\ &= v_\pi(s) \end{aligned}$$

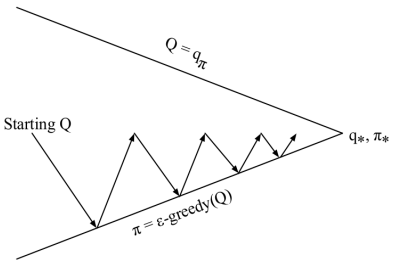

Figure 2: Monte Carlo Control

where $m = |\mathcal{A}(s)|$. The inequality above comes from the fact that max of a set of real numbers is at-least as big as any weighted average of those numbers. Now, from policy improvement theorem, $\pi' \geq \pi$ (i.e. $v_{\pi'}(s) \geq v_\pi(s), \forall s \in \mathcal{S}$).

Theorem. *For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to q_π is an improvement, i.e. $v_{\pi'}(s) \geq v_\pi(s)$.*

As stated in the algorithm, we alternate between MC policy evaluation and ϵ-greedy policy improvement in an episode-by-episode basis. Every iteration, an episode is generated using current policy π. Then this policy is evaluated, which is used again for policy improvement of all the state-action pairs present in the episode. This is also shown graphically in figure-2.

An important observation here is that the MC control method discussed so far never converges to a deterministic optimal policy. This is because the resulting policy is always ϵ-greedy, which always has some randomness

within it. One way to get around this issue is using GLIE (Greedy in the Limit with Infinite Exploration) MC control.

Definition. *A policy is said to be GLIE (Greedy in the Limit with Infinite Exploration) if*

- *All state-action pairs are explored infinitely many times*

$$\lim_{k\to\infty} N_k(s,a) = \infty$$

- *The policy converges to a greedy policy*

$$\lim_{k\to\infty} \pi_k(a|s) = \mathbb{1}(a = \underset{a'\in\mathbb{A}}{\operatorname{argmax}}\, Q_k(s,a'))$$

So, ϵ-greedy policy discussed above is GLIE if ϵ reduces to 0 as number of steps/episodes increase. One such decrement is given by $\epsilon \leftarrow 1/k$. In algorithm-1, if ϵ is decremented in this way before every policy improvement step, the resulting algorithm is known as **GLIE Monte-Carlo** control.

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

2 On-Policy TD Control: Sarsa

Temporal difference learning methods have several obvious advantages over Monte-Carlo methods. TD methods naturally work in continuous environments and have efficient online implementations. TD estimates generally have much lower variance than their MC counterparts. So, let's turn our attention now to temporal difference methods for control.

Algorithm 2: Sarsa (On-policy TD Control) for estimating Q $\sim q_*$

Algorithm Parameters: step size $\alpha \in (0,1]$, small $\epsilon > 0$
Initialization: Initialize $Q(s,a)$ arbitrarily for all state-action pairs. $Q(terminal,\cdot) = 0$
Initialize π to be ϵ-greedy policy with respect to Q

for *each episode* **do**
  Initialize S
  Choose A from S using policy π
  for *each timestep t of episode* **do**
    Take action A, observe R, S'
    Choose A' from S' using policy π
    $Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A))$
    for *all* $a \in \mathcal{A}$ **do**
      $\pi(a|S) \leftarrow \begin{cases} \epsilon/|\mathcal{A}(\mathcal{S})| + 1 - \epsilon, & \text{if } a = \operatorname{argmax}_{a'\in\mathcal{A}} Q(S,a') \\ \epsilon/|\mathcal{A}(\mathcal{S})|, & \text{otherwise} \end{cases}$
    end
    $S \leftarrow S'$
    $A \leftarrow A'$
    If S is terminal, proceed to next episode
  end
end

On-policy TD control methods also use the idea of GPI, but by using TD methods for polity evaluation. Just like MC method, ϵ-greedy policy improvement is used. The resulting algorithm is called **Sarsa**. Sarsa first learns action-value function $q_\pi(s,a)$ for the current policy π for all state-action pairs (s,a). This is done using the TD(0) prediction method. Sarsa considers transitions from state-action pair to state-action pair and learn the values of all state-action pairs. TD(0) update for action values is given as:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t))$$

This update is done after every transition from a non-terminal state S_t. The update rule uses events $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, that make up a transition from one state-pair to the next. This quintuple gives rise to the name Sarsa. After estimating q_π, at each step, we update the policy towards ϵ-greedy policy w.r.t. the value function q_π. The complete pseudocode of Sarsa is given in algorithm-2.

Theorem. *Sarsa converges to an optimal action-value function* $Q(s,a) \to q_*(s,a)$ *given following two conditions are met:*

- *All intermediate policies* $\pi_t(a|s)$ *are GLIE*
- *Step sizes* α_t *satisfy Robbins-Munro conditions*

$$\sum_{t=1}^{\infty} \alpha_t = \infty, \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

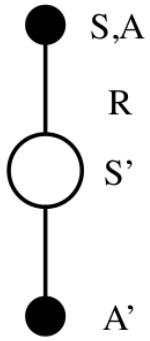

Figure 3: Sarsa Backup Diagram

3 n-Step Sarsa

Let us now see how n-step methods can be combined with Sarsa to produce on-policy control method, known as n-step Sarsa. The main idea behind n-step Sarsa is to learn action values instead of state values, and then use ϵ-gree policy with respect to these. It is also shown in the backup diagrams of n-step Sarsa (fig 4) where only difference from n-step methods is that all of them end in actions rather than states.
Lets define the n-step Q-returns in terms of estimated action values as follows:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n}), \ n \geq 1, 0 \leq t \leq T - n.$$

Then, n-step Sarsa updates $Q(s,a)$ towards n-step Q-return:

$$Q_{t+n}(S_t, A_t) \leftarrow Q_{t+n-1}(S_t, A_t) + \alpha(G_t^{(n)} - Q_{t+n-1}(S_t, A_t)), \ 0 \leq t \leq T$$

while the values of all other states remain unchanged. I.e. $Q_{t+n}(s,a) = Q_{t+n-1}(s,a)$, for all s, a such that $s \neq S_t, a \neq A_t$. Pseudocode for n-step Sarsa algorithm is given by algorithm-3.

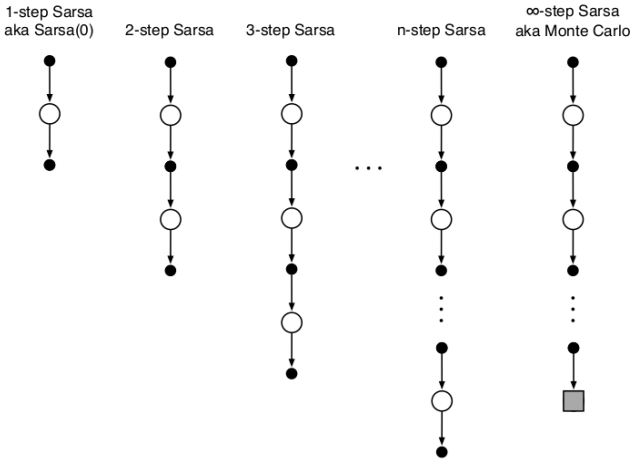

Figure 4: Backup diagrams for n-step Sarsa

Algorithm 3: n-step Sarsa for estimating $Q \sim q_\pi$ or q_*

Algorithm Parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$ and a positive integer n
Initialization: Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ϵ-greedy policy with respect to Q, or to a fixed given policy

for *each episode* **do**
  Initialize and store $S_0 \neq$ terminal
  Select an action $A_0 \sim \pi(\cdot|S_0)$
  $T \leftarrow \infty$
  for *each time-step* $t = 0, 1, 2, \cdots$ *until* $t' = T - 1$ **do**
    if $t < T$ **then**
      Take action A_t
      Observe and store R_{t+1} and S_{t+1}
      if S_{t+1} *is terminal* **then**
        $T \leftarrow t + 1$
      else
        Select an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
      end
    end
    $t' \leftarrow t - n + 1$ (time whose state's estimate is being updated)
    if $t' \geq 0$ **then**
      $G \leftarrow \sum_{i=t'+1}^{min(t'+n,T)} \gamma^{i-t'-1} R_i$
      If $t' + n < T$, then $G \leftarrow G + \gamma^n Q(S_{t'+n}, A_{t'+n})$
      $Q(S_{t'}, A_{t'}) \leftarrow Q(S_{t'}, A_{t'}) + \alpha[G - Q(S_{t'}, A_{t'})]$
      ensure that $\pi(\cdot|S_{t'})$ is ϵ-greedy w.r.t. Q
    end
  end
end

4 Sarsa(λ)

As discussed in prediction problem, the n-step returns can be combined by taking their weighted average, and define the λ-return:

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

with weight parameter $\lambda \in [0, 1]$. The backup diagram of Sarsa(λ) (figure-5) illustrates the weights given to Q-returns of different time-steps.

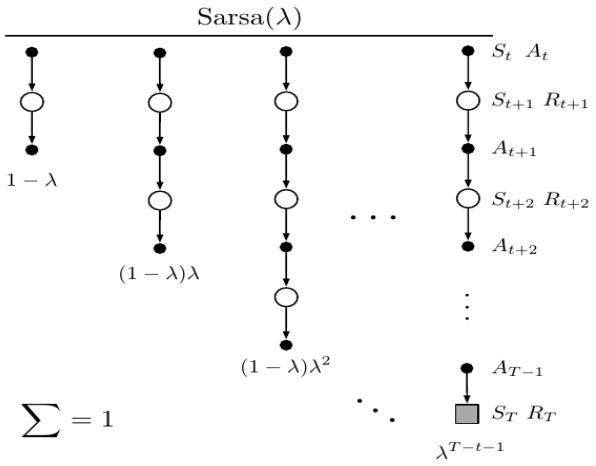

Figure 5: Backup diagrams for Sarsa(λ)

Just like TD(λ) in prediction, Sarsa(λ) also has two different views - **forward view**, which is an offline view, and **backward view** which provides the online implementation of Sarsa(λ) by the use of eligibility traces. Forward view Sarsa(λ) updates action values towards λ return:

$$Q_{t+n}(S_t, A_t) \leftarrow Q_{t+n-1}(S_t, A_t) + \alpha(G_t^{\lambda} - Q_{t+n-1}(S_t, A_t)),\ 0 \leq t \leq T$$

while the values of all other states remain unchanged.

Backward view provides an online implementation of Sarsa(λ) through the use of eligibility traces. Sarsa(λ) has one eligibility trace defined for each state-action pair.

$$\begin{aligned} E_0(s, a) &= 0 \\ E_t(s, a) &= \gamma\lambda E_{t-1}(s, a) + \mathbb{1}(S_t = s, A_t = a) \end{aligned}$$

Then, using this eligibility trace, whenever a non-zero reward is available value functions for all relevant state-action pairs can be updated in proportion to their eligibility traces $E_t(s, a)$ and TD-error δ_t.

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \\ Q(s, a) &\leftarrow Q(s, a) + \alpha\delta_t E_t(s, a) \end{aligned}$$

A pseudocode for complete Sarsa(λ) is given by algorithm-4.

Algorithm 4: Sarsa(λ) for estimating $Q \sim q_\pi$ or q_*

Algorithm Parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialization: Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ϵ-greedy policy with respect to Q

for *each episode* **do**
  Initialize and store $S \neq$ terminal
  Select an action $A \sim \pi(\cdot|S)$
  $E(s, a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$ (Eligibility Trace)
  for *each step* $t = 0, 1, \cdots$ *until* S *is terminal state* **do**
    Take action A, Observe reward R and next state S'
    Select an action $A' \sim \pi(\cdot|S')$
    $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
    $E(S, A) \leftarrow E(S, A) + 1$
    for *all* $s \in \mathcal{S}, a \in \mathcal{A}(s)$ **do**
      $Q(s, a) \leftarrow Q(S, A) + \alpha\delta E(s, a)$
      $E(s, a) \leftarrow \gamma\lambda E(s, a)$
    end
    $S \leftarrow S'; A \leftarrow A'$
    ensure that π is ϵ-greedy w.r.t. Q
  end
end

5 Off-Policy Methods and Q-Learning

Let us now focus attention towards more general control methods, known as off-policy methods. In off-policy methods the policy being used to generate behavior, called the behavior policy, may be different being evaluated and improved, called the target policy. By using two different policies, the off-policy methods provide a way to explore (by using stochastic behavior policy) while also moving towards an optimal deterministic policy (target policy).

5.1 Off-policy Monte Carlo Control

Monte Carlo control methods use the idea of weighted importance sampling and generalized policy iteration to estimate π_* and q_*. The target policy π is chosen to be greedy w.r.t. action value function, while the behavior policy μ is chosen to be ϵ-soft to keep exploring more actions. The pseudocode for off-policy MC is given by algorithm-5.

Algorithm 5: Off-policy MC control for estimating $\pi \sim \pi_*$

Initialization: Initialize $Q(s,a)$ arbitrarily, $C(s,a) \leftarrow 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be a greedy policy with respect to Q. i.e. $\pi(s) \leftarrow \text{argmax}_a\, Q(s,a)$

for *each episode (until convergence)* **do**
  $\mu \leftarrow$ any soft policy
  Generate an episode following $\mu : S_0, A_0, R_1, \cdots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  for *each time-step* $t = T-1, T-2, \cdots, 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}(G - Q(S_t, A_t)$
    $\pi(S_t) \leftarrow \text{argmax}_a\, Q(S_t, a)$ (with ties broken consistently)
    If $A_t \neq \pi(S_t)$ then proceed to next episode
    $W \leftarrow W \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}$
  end
end

5.2 Q-Learning

Q-learning is one of the most popular off-policy TD control algorithm in reinforcement learning. Recall that in order to learn an optimal action value function (and hence an optimal policy) for off-policy control, it is required that the behavior policy continue to explore all state-action pairs in long run. Q-learning ensures that using an ϵ-soft behavior policy and greedy target policy. However it differs from other off-policy learning methods in that the learned action value function directly approximates $q*$, irrespective of the policy being followed.

Q-leaning doesn't use importance sampling. An ϵ-greedy behavior policy is used to sample next action, but alternative successor states considered are based on the target policy being optimized. So, Q-learning allows both behavior and target policy to improve over time-steps. The target of Q-learning is given by:

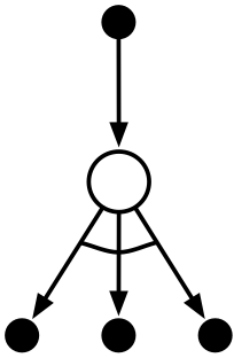

Figure 6: Q-Learning Backup Diagram

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$$

Target policy is greedy w.r.t. Q and behavior policy is chosen to be ϵ-greedy w.r.t. Q.

Algorithm 6: Q-learning (Off-policy TD control) for estimating $\pi \sim \pi_*$

Algorithm Parameters: step size $\alpha \in (0,1]$, small $\epsilon > 0$
Initialization: Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}, Q(terminal, \cdot) = 0$

for *each episode (until convergence)* **do**
  Initialize S
  for *each step of episode* **do**
    Choose A from S using ϵ-greedy policy derived from Q
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma \max_a Q(S', a) - Q(S,A))$
    $S \leftarrow S'$
    If S is terminal, proceed to next episode
  end
end

Theorem. *Q-learning control algorithm converges to the optimal action-value function,* $Q(s,a) \rightarrow q_\pi(s,a)$

6 Relationship Between DP and TD Methods

Table-1 provides a relationship between dynamic programming methods and temporal different learning methods such as Sarsa and Q-leaning.

	Full Backup (DP)	Sample Backup (TD)
Bellman Expectation Equation for $v_\pi(s)$	Iterative Policy Evaluation $V(s) \leftarrow \mathbb{E}\left[R + \gamma V(S')|s\right]$	TD Learning $V(S) \leftarrow V(S) + \alpha(R + \gamma V(S') - V(S))$
Bellman Expectation Equation for $q_\pi(s,a)$	Q-Policy Iteration $Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma Q(S',A')|s,a\right]$	Sarse $Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A))$
Bellman Optimality Equation for $q_*(s,a)$	Q-Value Iteration $Q(s,a) \leftarrow \mathbb{E}\left[R + \max_{a' \in \mathcal{A}} \gamma Q(S',a')|s,a\right]$	Q-Learning $Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma \max_{a' \in \mathcal{A}} Q(S',A') - Q(S,A))$

Table 1: Relationship Between DP and TD

References

[1] Reinforcement Learning: An Introduction (2nd Edition), by Richar Sutton, Andrew Barto.

[2] David Silver's Reinforcement Learning course available at
`https://www.davidsilver.uk/teaching/`