

# Reinforcement Learning Notes

Avinash Reddy

January 2023



# Chapter 1

## Introduction

### Introduction

Reinforcement Learning is also known as

- optimal control
- Approximate Dynamic programming
- Neuro-Dynamic Programming

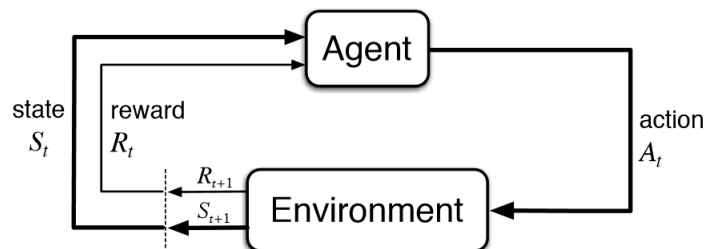
Reinforcement Learning is an area of machine learning inspired by the behavioural psychology concerned with how software agents ought to take actions in an environment so as to maximise some notion of cumulative reward.

Animal psychology

- negative rewards - pain and hunger
- positive reinforcements - pleasure and food
- reinforcements used to train animals

Applying the similar philosophy to software agents

### 1.1 Definition



Reinforcement Learning Application Areas :

- Game Playing
- Operations Research
- Elevator Scheduling
- controls
- spoken dialogue systems
- data center energy consumption
- self managing networks
- autonomous vehicles
- computational finance

## Chapter 2

# Multi-Armed Bandits

We are not interested in this chapter. Or else we will visit at end of our journey. Just introduced to match the chapters number with Sutton and Barto book chapters.



## Chapter 3

# Markov Decision Process

### Definition

- State  $s \in S$
- Action  $a \in A$
- reward  $r \in \mathbb{R}$
- Transition model  $Pr(s_{t+1}|s_t, a_t)$
- Reward model  $Pr(r|s_{t+1}, s_t, a_t)$
- Discount Factor  $\gamma \in [0, 1]$
- Horizon  $T$

Goal is to find a policy  $\pi(a|s)$  that maximises the expectation of discounted return.

### How RL differs from MDP solutions

- No Transition Model
- No Reward Model

$$J(\pi) = \mathbb{E}_{\pi} \left[ \sum_{t=0}^T \gamma^t r_t \right]$$

However, we still solve the MDP problem using RL by interacting with the environment by learning the transition and reward models or directly learning the policy.

### Types of RL algorithms

- Model Based- if we try to learn the transition and reward models
- Model Free - here, we don't learn any model dynamics. No transition and reward models. Below are the types of model free RL algorithms
- Value Based- if we try to learn the value function  $V(s)$  of the state or value function of state-action pair  $Q(s, a)$ .
- Policy Based- if we try to learn the policy  $\pi(a|s) - \pi(s, a)$  directly.
- Policy Gradient- if we try to learn the policy  $\pi(a|s) - \pi(s, a)$  directly using gradient ascent.
- Actor Critic - contains both policy  $\pi(a|s) - \pi(s, a)$  and value function  $V(s) - Q(s, a)$ .





## Chapter 4

# Dynamic Programming

we compute the state value function  $V^\pi(s)$  for the policy  $\pi$ . This is called the *policy evaluation* problem. We can write the Bellman equation for the state value function as

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_t = s \right] \\ &= \mathbb{E}_\pi [G_t \mid s_t = s] \\ &= \mathbb{E}_\pi [R_{t+1} + \gamma G_{t+1} \mid s_t = s] \\ &= \mathbb{E}_\pi [R_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s] \\ &= \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) [r + \gamma V^\pi(s')] \end{aligned} \tag{4.1}$$

If environment dynamics are known, then  $V^\pi(s)$  is a simultaneous linear equations in  $|S|$  unknowns. Iterative solutions are most suitable. Assume  $v_0$  as the initial approximation for the state value function. Then update rule is given by the bellman equation

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_k(s')] \tag{4.2}$$

This iterative approach converges to  $V^\pi(s)$ . The policy evaluation problem is solved when  $v_k(s) = V^\pi(s)$  for all  $s \in S$ .

### Iterative Policy Evaluation Algorithm

#### State-Action Value Function

The state-action value function  $Q^\pi(s, a)$  is defined as the expected return starting from state  $s$ , taking action  $a$ , and then following policy  $\pi$  thereafter. We can write the Bellman equation for the state-action value function as

**Algorithm 1** Iterative Policy Evaluation

- 
- 1: Input policy  $\pi$ , initial approximation  $v_0(s) \forall s \in S$ ,
  - 2: Set discount factor  $\gamma$ ,  $k = 0$ ,  $V(\text{terminal}) = 0$
  - 3:  $v_k(s) \leftarrow v_0(s)$  for all  $s \in S$
  - 4: **while**  $|v_k - v_{k-1}| < \theta$  **do**
  - 5:    $v_{k+1}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a | s) \sum_{s', r} p(s', r | s, a) [r + \gamma v_k(s')]$  for all  $s \in S$
  - 6:    $k \leftarrow k + 1$
  - 7: **end while**
  - 8: **return**  $v_k(s)$  for all  $s \in S$
- 

$$\begin{aligned}
Q^\pi(s, a) &= \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_t = s, a_t = a \right] \\
&= \mathbb{E}_\pi [G_t \mid s_t = s, a_t = a] \\
&= \mathbb{E}_\pi [R_{t+1} + \gamma G_{t+1} \mid s_t = s, a_t = a] \\
&= \mathbb{E}_\pi [R_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a] \\
&= \sum_{s', r} p(s', r \mid s, a) \left[ r + \gamma \sum_{a'} \pi(a' \mid s') Q^\pi(s', a') \right] \\
&= \mathbb{E}_\pi [R_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a] \\
&= \sum_{s', r} p(s', r \mid s, a) [r + \gamma V^\pi(s')]
\end{aligned} \tag{4.3}$$

*policy improvement theorem* helps us in updating the policy once we found out the value function of a policy by policy evaluation. The theorem states that if we have a policy  $\pi$  and a value function  $V^\pi(s)$ , then there exists a policy  $\pi'$  such that  $V^{\pi'}(s) \geq V^\pi(s)$  for all  $s \in S$ . This means that we can always improve the value of a policy by following a greedy policy with respect to the value function. The greedy policy with respect to the value function is defined as

$$\pi'(s) = \arg \max_{a \in \mathcal{A}} Q^\pi(s, a) \tag{4.4}$$

which leads to  $V^{\pi'}(s) \geq V^\pi(s)$  for all  $s \in S$ .

**Policy Improvement Algorithm**

Assuming a deterministic policy  $\pi(s) = a$

**Algorithm 2** Policy Improvement

- 
- 1: Input policy  $\pi$ , value function  $v(s)$  for all  $s \in S$ ,
  - 2: Set discount factor  $\gamma$
  - 3: Evaluate the policy  $\pi$  to get  $v(s)$  for all  $s \in S$  using Algorithm 1 Policy Evaluation.
  - 4: Improve the policy by following a greedy policy with respect to the value function.  $\pi \leftarrow \arg \max_{a \in \mathcal{A}} Q^\pi(s, a)$  for all  $s \in S$
-

However, policy improvement algorithm involves policy evaluation. So, we need to repeat the policy evaluation and policy improvement until the policy converges. So, Value iteration algorithm updates the value function by acting greedily with respect to the value function and then improves the policy by following a greedy policy with respect to the value function. The algorithm is given below.

### Value Iteration Algorithm

Assuming a deterministic policy  $\pi(s) = a$

---

#### Algorithm 3 Value Iteration

---

- 1: Input policy  $\pi$ , initial approximation  $v_0(s) \forall s \in S$ ,
  - 2: Set discount factor  $\gamma$ ,  $k = 0$ ,  $V(\text{terminal}) = 0$
  - 3:  $v_k(s) \leftarrow v_0(s)$  for all  $s \in S$
  - 4: **while**  $|v_k - v_{k-1}| < \theta$  **do**
  - 5:      $v_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_k(s')]$  for all  $s \in S$
  - 6:      $k \leftarrow k + 1$
  - 7: **end while**
  - 8: **return**  $v_k(s)$  for all  $s \in S$
-



## Chapter 5

# Monte Carlo Methods

Dynamic Programming methods are suitable for MDPs for which the model dynamics are known. In this chapter, we will discuss the methods that can be used to solve MDPs for which the model dynamics are not known. These methods are called Monte Carlo methods. The main idea behind Monte Carlo methods is to estimate the value function by averaging the returns obtained from the episodes. The main advantage of Monte Carlo methods is that they do not require the model dynamics to be known. The main disadvantage of Monte Carlo methods is that they are computationally expensive.

### 5.1 Monte Carlo Prediction

In earlier chapters, we defined value function as expected sum of discounted rewards. To estimate the expected sum, one way is to average the discounted rewards obtained from the episodes. This is the idea behind the monte carlo methods. Even the core idea of averaging remains same, there are simple variations in the way we consider the rewards for averaging.

Let us discuss two simple algorithms.

- First-visit MC
- Every visit MC

**Algorithm 4** First-visit MC

---

```

1: Initialize  $V(s)$  arbitrarily for each  $s \in S$ , and policy  $\pi$  to be evaluated
2: Initialize  $Returns(s), N(S_t)$  as empty list, for each  $s \in S$ 
3: while each episode do:
4:   Generate an episode following  $\pi$ :  $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$ 
5:    $G \leftarrow 0$ 
6:   while Loop for each step of episode  $t = T - 1, \dots, 2, 1, 0$  do:
7:      $G \leftarrow \gamma G + R_{t+1}$ 
8:     Unless  $S_t$  is already in  $S_0, S_1, \dots, S_{t-1}$ :
9:       Append  $G$  to  $Returns(S_t)$ 
10:       $N(S_t) \leftarrow N(S_t) + 1$ 
11:       $V_{n+1}(S_t) \leftarrow V_n(S_t) + \frac{1}{N(S_t)} [G - V_n(S_t)]$ 
12:   end while
13: end while

```

---

First visit MC averages the rewards following from the first visit of the state.

**Algorithm 5** Every-visit MC

---

```

1: Initialize  $V(s)$  arbitrarily for each  $s \in S$ , and policy  $\pi$  to be evaluated
2: Initialize  $Returns(s), N(S_t)$  as empty list, for each  $s \in S$ 
3: while each episode do:
4:   Generate an episode following  $\pi$ :  $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$ 
5:    $G \leftarrow 0$ 
6:   while Loop for each step of episode  $t = T - 1, \dots, 2, 1, 0$  do:
7:      $G \leftarrow \gamma G + R_{t+1}$ 
8:     Append  $G$  to  $Returns(S_t)$ 
9:      $N(S_t) \leftarrow N(S_t) + 1$ 
10:     $V_{n+1}(S_t) \leftarrow V_n(S_t) + \frac{1}{N(S_t)} [G - V_n(S_t)]$ 
11:   end while
12: end while

```

---

Every visit MC averages the rewards following every visit of the state.

One can modify the algorithm to find the state action value function. The algorithm is same as above except that we have to maintain the state action value function instead of state value function.

Monte Carlo methods doesn't bootstrap and the estimated value function is state independent. It means that monte carlo method doesn't use the estimate of other states to estimate the value of a state. It is also called on-policy method because it uses the same policy to generate the episodes and to evaluate the policy.

## 5.2 Monte Carlo Control

We now have the algorithm to evaluate any given policy  $\pi$  in unknown dynamics MDP. We can use those algorithm to evaluate the policy  $\pi$  and then improve the policy. This is the idea behind Monte Carlo control. Using the same strategy used in policy improvement algorithm, we find the values function for a given policy and then act greedy with respect to the value function which improves the earlier policy. We repeat this process until we find the optimal policy.

There are inherent assumptions in the above algorithms. One assumption is exploring starts, all the state-action pairs are explored enough to average the rewards. Another assumption is we have to sample infinite episodes to converge. The second assumption we can relax by following the general policy improvement algorithm.

We will see modified version of first visit MC algorithm to find the optimal policy. The algorithm is same as first visit MC algorithm except that we use the greedy policy with respect to the value function to generate the episodes.

---

**Algorithm 6** First Visit MC with policy improvement

---

```

1: Initialize  $Q(s, a)$  arbitrarily for each  $s \in S, a \in A$ , and policy  $\pi$  to be evaluated
2: Initialize  $Returns(s, a), N(s, a)$  as empty list, for each  $s \in S, a \in A$ 
3: while each episode do:
4:   Generate an episode following  $\pi$ :  $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$ 
5:    $G \leftarrow 0$ 
6:   while Loop for each step of episode  $t = T - 1, \dots, 2, 1, 0$  do:
7:      $G \leftarrow \gamma G + R_{t+1}$ 
8:     Unless  $S_t$  is already in  $S_0, S_1, \dots, S_{t-1}$ :
9:       Append  $G$  to  $Returns(S_t)$ 
10:       $N(S_t) \leftarrow N(S_t) + 1$ 
11:       $Q_{n+1}(S_t, A_t) \leftarrow Q_n(S_t, A_t) + \frac{1}{N(S_t)} [G - Q_n(S_t, A_t)]$ 
12:       $\pi(S_t) \leftarrow \operatorname{argmax}_a Q_{n+1}(S_t, A_t)$ 
13:   end while
14: end while

```

---

Now, we will apply the same policy improvement idea on every visit MC algorithm.

---

**Algorithm 7** Every Visit MC with policy improvement

---

```

1: Initialize  $Q(s, a)$  arbitrarily for each  $s \in S, a \in A$ , and policy  $\pi$  to be evaluated
2: Initialize  $Returns(s, a), N(s, a)$  as empty list, for each  $s \in S, a \in A$ 
3: while each episode do:
4:   Generate an episode following  $\pi$ :  $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$ 
5:    $G \leftarrow 0$ 
6:   while Loop for each step of episode  $t = T - 1, \dots, 2, 1, 0$  do:
7:      $G \leftarrow \gamma G + R_{t+1}$ 
8:     Append  $G$  to  $Returns(S_t, A_t)$ 
9:      $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$ 
10:     $Q_{n+1}(S_t, A_t) \leftarrow Q_n(S_t, A_t) + \frac{1}{N(S_t)} [G - Q_n(S_t, A_t)]$ 
11:     $\pi(S_t) \leftarrow \operatorname{argmax}_a Q_{n+1}(S_t, A_t)$ 
12:   end while
13: end while

```

---

From sutton and Barto, the convergence to an optimal policy is inevitable. However, there is no formal proof to show the convergence.

To relax the first assumption, we should make sure that policy is epsilon soft.  $\pi(\cdot|S_t) > 0$ . This is achieved by using  $\epsilon$ -greedy policy. In contrast to the greedy approach, we do not choose action greedily rather we choose action with probability  $\epsilon$  randomly and with probability  $1 - \epsilon$  greedily.

This is called  $\epsilon$ -greedy policy. We will see the algorithm to find the optimal policy using  $\epsilon$ -greedy policy.

---

**Algorithm 8** Every Visit MC with  $\epsilon$ -greedy policy improvement

---

```

1: Initialize  $Q(s, a)$  arbitrarily for each  $s \in S, a \in A$ , and policy  $\pi$  to be evaluated
2: Initialize  $Returns(s, a), N(s, a)$  as empty list, for each  $s \in S, a \in A$ 
3: while each episode do:
4:   Generate an episode following  $\pi$ :  $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$ 
5:    $G \leftarrow 0$ 
6:   while Loop for each step of episode  $t = T - 1, \dots, 2, 1, 0$  do:
7:      $G \leftarrow \gamma G + R_{t+1}$ 
8:     Append  $G$  to  $Returns(S_t, A_t)$ 
9:      $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$ 
10:     $Q_{n+1}(S_t, A_t) \leftarrow Q_n(S_t, A_t) + \frac{1}{N(S_t)} [G - Q_n(S_t, A_t)]$ 
11:     $\pi(a|S_t) \leftarrow \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|} & \text{if } a = \operatorname{argmax}_a Q_{n+1}(S_t, A_t) \\ \frac{\epsilon}{|A|} & \text{otherwise} \end{cases}$ 
12:   end while
13: end while

```

---

From sutton and barto, the above epsilon-greedy policy improvement algorithm also converges to an optimal policy. Till now, we have seen the on-policy algorithms to find the optimal policy. On-policy algorithms suffer from disadvantage that it can't reuse the generated episodes. To overcome this disadvantage. On-policy approach means we use a policy to generate episodes and use the same policy to improve the policy. Off-policy approach means we use a policy called *behaviour policy* to generate episodes and use another policy called *target policy* to improve continuously.

we first solve the prediction problem of value function using off-policy approach. We will use the same algorithm as we used for on-policy approach. The only difference is that we will use the behaviour policy to generate episodes and use the target policy to improve the policy. we make an assumption that  $b(a|s) > 0$  to facilitate the exploring starts behaviour. On the otherhand, target policy can be deterministic. While the behaviour policy must be stochastic to facilitate the exploration.

*Importance Sampling* is the trick followed by all off-policy algorithms. With the help of importance sampling, we can reuse the generated episodes. Instead of accumulating the discounted sum of rewards, we will accumulate the weighted discounted sum of rewards. The weight is the ratio of the probability of the action taken by the behaviour policy to the probability of the action taken by the target policy.

Given a starting state  $S_0$  under a stochastic policy  $\pi$ , the resulting trajectory will have a probability distribution

$$P_\pi(S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T) = \prod_{t=0}^{T-1} \pi(A_t|S_t) \cdot P(S_{t+1}|S_t, A_t) \cdot R_{t+1}$$

We define the relative probability of the trajectory under the target policy  $\pi$  and the behaviour policy  $b$  as



$$\begin{aligned}\rho_{t:T-1} &= \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k) Pr(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k) Pr(S_{k+1}|S_k, A_k)} \\ &= \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}\end{aligned}$$

This is called importance sampling ratio. If you sample with behaviour policy  $b$  and then estimate  $V(s)$  as  $\mathbb{E}[G_t|S_t]$  which will result in  $V_b(s)$ . Nevertheless, we need to estimate  $V_\pi(s)$  which is the value function under the target policy. We can use the importance sampling ratio to estimate  $V_\pi(s)$  as expected weighted sum of rewards, where the weight is given by the importance sampling ratio. Instead of  $\mathbb{E}[G_t|S_t]$ , you approximate  $\mathbb{E}[\rho_{t:T-1} G_t|S_t] = V_\pi(s)$ .  $\rho$  is the importance sampling ratio.

Off-policy Every Visit MC Prediction Algorithm needs to track where the state-action pair occurs in the episode. Let it be  $T(s, a)$ . We store the time steps at which the state-action pair occurs in the episode. This favours us to calculate the importance sampling ratio from those time steps. We will use the importance sampling ratio to calculate the weighted sum of rewards. This will give us the value function under the target policy.

$$Q_\pi(s, a) = \frac{\sum_{t \in T(s, a)} \rho_{t:T(t)-1} G_t}{|T(s, a)|}$$

This is called ordinary importance sampling.

$$Q_\pi(s, a) = \frac{\sum_{t \in T(s, a)} \rho_{t:T(t)-1} G_t}{\sum_{t \in T(s, a)} \rho_{t:T(t)-1}}$$

This is an alternative way of calculating the value function under the target policy. This is called weighted importance sampling. Now the question is which algorithm to follow or which algorithm leads to convergence and at what rate.

Let's see in case of first visit algorithms. The differences are generally expressed in terms of biases and variances. Ordinary importance sampling is unbiased whereas weighted importance sampling is biased (although the bias converges to zero finally). Ordinary importance sampling has unbounded variance. On the other hand, weighted importance sampling had lower variance and is the preferred algorithm. Nevertheless, we will use the ordinary importance sampling while using approximate value function methods.

In every visit monte carlo case, both ordinary importance sampling and weighted importance sampling are biased but eventually converge to zero.

We will look at the implementation of off-policy every visit monte carlo prediction algorithm. we are estimating

$$Q_\pi(s, a) = \frac{\sum_{t \in T(s, a)} \rho_{t:T(t)-1} G_t}{\sum_{t \in T(s, a)} \rho_{t:T(t)-1}}$$

we approximate it with the weight parameter  $W_k$ .

$$Q_\pi(s, a) = \frac{\sum_{t \in T(s, a)} W_k G_t}{\sum_{t \in T(s, a)} W_k}$$

An incremental update rule for  $Q_\pi(s, a)$  is

$$Q_{n+1}(s, a) = Q_n(s, a) + \frac{W_n}{C_n} [G_n - V_n]$$

where  $C_{n+1} = C_n + W_{n+1}$

**Off policy Every Visit Monte Carlo Prediction Algorithm**

---

**Algorithm 9** Off policy Every Visit Monte Carlo Prediction Algorithm

---

```

1: Input an arbitrary policy  $\pi$  and a behaviour policy  $b$ 
2: Initialize  $Q(s, a)$  arbitrarily for all  $s \in S$  and  $a \in A(s)$ 
3: Initialize  $C(s, a)$  with zeros for all  $s \in S$  and  $a \in A(s)$ 
4: while each episode do
5:   Generate an episode  $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$  under behaviour policy  $b$ 
6:    $G \leftarrow 0$ 
7:    $W \leftarrow 1$ 
8:   for  $t = T - 1$  to 0 do
9:      $G \leftarrow \gamma G + R_{t+1}$ 
10:     $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$ 
11:     $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$ 
12:     $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$ 
13:   end for
14: end while

```

---

we now the algorithm to evaluate a state value function under a target policy  $\pi$  and any other behaviour policy  $b$ . To improve the policy  $\pi$ , we follow the general policy improvement approach by acting greedy.

$$\pi'(s) = \arg \max_{a \in A(s)} Q_\pi(s, a)$$

such that  $\pi'(s) \geq \pi(s)$  for all  $s \in S$ .

(Off policy Every Visit Monte Carlo Control Algorithm)

**Algorithm 10** Off policy Every Visit Monte Carlo Control Algorithm

---

```

1: Initialize  $Q(s, a)$  arbitrarily for all  $s \in S$  and  $a \in A(s)$ 
2: Initialize  $C(s, a)$  with zeros for all  $s \in S$  and  $a \in A(s)$ 
3: Input an arbitrary soft behaviour policy  $b$  ▷ e.g.  $\epsilon$ -soft
4:  $\pi \leftarrow \operatorname{argmax}_a Q(s, a)$ 
5: while each episode do
6:   Generate an episode  $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$  under behaviour policy  $b$ 
7:    $G \leftarrow 0$ 
8:    $W \leftarrow 1$ 
9:   for  $t = T - 1$  to 0 do
10:     $G \leftarrow \gamma G + R_{t+1}$ 
11:     $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$ 
12:     $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$ 
13:     $\pi'(s) \leftarrow \operatorname{argmax}_{a \in A(s)} Q_\pi(s, a)$ 
14:     $\pi \leftarrow \pi'$ 
15:    if  $A_t \neq \pi(S_t)$  then
16:      Break
17:    end if
18:  end for
19:   $W \leftarrow W \frac{1}{b(A_t|S_t)}$ 
20: end while

```

---



## Chapter 6

# Temporal Difference Learning