Notes

Mathematical details of Book-Reinforcement Learning: an introduction

---

Yu Chen

The Chinese University of HongKong, Department of Mechanical and Automation Engineering

E-mail: anschen@link.cuhk.edu.hk

Contents

1 Gradient banits

For gradient banits, we do not use estimated rewards to choose current action, instead, we use a local score $H_t(a)$ to evaluate the goodness of action a. Specificly, we introduce a softmax-like distribution,

$$\pi_t(a) = \frac{e^{H_t(a)}}{\sum_b e^{H_t(b)}}. \tag{1.1}$$

Theorem 1.0.1 (Updating strategy)**.**

$$H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \overline{R}_t)(1 - \pi_t(A_t)), \tag{1.2}$$
$$H_{t+1}(a) = H_t(a) - \alpha(R_t - \overline{R}_t)\pi_t(a), \forall a \neq A_t, \tag{1.3}$$

where α is stepsize, and $\overline{R}_t$ is the averaged rewards up through and including time t.

Proof: The above the updating strategy is the direct result of gradient ascent algorithm. Whatever action takes, we can update the local score by the following formulas,

$$H_{t+1}(a) = H_t(a) + \alpha \frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)}, \tag{1.4}$$

where $\mathbb{E}[R_t] = \sum_b \pi_t(b) q(b)$, here $q(b)$ is the true value of action b. Although we may not know what true $q(b)$ is, we can derive it updating rule, which is advantage of gradient banits.

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \sum_b q(b) \frac{\partial \pi_t(b)}{\partial H_t(a)} \tag{1.5}$$

$$= \sum_b q(b) \sum_c \frac{\delta_{a,b} e^{H_t(a)} \sum_c e^{H_t(c)} - e^{H_t(b)} e^{H_t(a)}}{(\sum_c e^{H_t(c)})^2} \tag{1.6}$$

$$= \sum_b q(b) \pi_t(b) (\delta_{a,b} - \pi_t(a)). \tag{1.7}$$

Since $\sum_n \frac{\partial \pi_t(b)}{\partial H_t(a)} = 0$, we can add a constant $-\overline{R}_t$ in the above equation, which means,

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \sum_b (q(b) - \overline{R}_t) \pi_t(b) (\delta_{a,b} - \pi_t(a)). \tag{1.8}$$

We can regard b is yummy varible of random variable of A_t, and then,

$$\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} = \mathbb{E}[(q(A_t) - \overline{R}_t)(1_{A_t=a} - \pi_t(a))] = \mathbb{E}[(R_t - \overline{R}_t)(1_{A_t=a} - \pi_t(a))], \tag{1.9}$$

where the second equality is because $R_t = \mathbb{E}[q(A_t)]$ and other variables are not random. And now we can justified the theorem.

2 Markov Decision Process

Markov decision process is composed of environment, agent. Mathematially, it contains a state space $\mathcal{S}$ describing the environment state, which can not be described exactly in many applications, an actions space $\mathcal{A}(s)$, which can be dependent on the current state, and a reward process, which is also stochastic. And the 'transition' probability can be described by the following tensor,

$$p(s', r|s, a) = Pr(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a), \tag{2.1}$$

which satifies Markovian property. Also, we can define some rewards functions,

Definition 2.0.1.

$$r(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_r r \sum_{s'} p(s', r|s, a) \tag{2.2}$$

$$r(s', s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s'] = \frac{\mathbb{E}[R_{t+1} 1_{S_{t+1}=s'}|S_t = s, A_t = a]}{Pr(S_{t+1} = s')|S_t = s, A_t = a} = \frac{\sum_r r p(s', r|s, a)}{\sum_r p(s', r|s, a)}. \tag{2.3}$$

For a episodic process, we can define returns as,

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T. \tag{2.4}$$

And for a continuous process, we can define the returns as,

$$G_t = \sum_{k=0}^{+\infty} \gamma^k R_{t+k+1} \tag{2.5}$$

If we define the terminal state of episodic process as an absorbing set with reward 0, we can unify the retunr expression as,

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}, \tag{2.6}$$

where $T = \infty$ corresponding to continuous process, and $\gamma = 1$ and finite T corresponding to eposidic process. Here, we should note that $T = +\infty$ and $\gamma = 1$ can not hold on simultaneously.(There are some research focus on it.)

Definition 2.0.2. *Value function Given a policy π, we define state value function of this policy as,*

$$v_\pi(s) = \mathbb{E}[G_t | S_t = a]. \tag{2.7}$$

Also, we define state-action pair function of this policy as,

$$q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]. \tag{2.8}$$

Theorem 2.0.1 (Bellman equation)**.**

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)(r + \gamma v_\pi(s')) \tag{2.9}$$

With the help of Bellman equation, we can find value function for a given policy and state by iteration.

Definition 2.0.3. *Optimal value function*

$$v_*(s) = \max_\pi v_\pi(s), \forall s \in \mathcal{S}. \tag{2.10}$$

$$q_*(s, a) = \max_\pi q_\pi(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{2.11}$$

From this definition, we can find the below relation easily,

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]. \tag{2.12}$$

Also, we can find the Bellman optimality equation as,

Theorem 2.0.2 (Bellman optimality equation)**.**

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)(r + \gamma v_*(s')), \tag{2.13}$$

$$q_*(s, a) = \sum_{s',r} p(s', r|s, a)(r + \gamma \max_{a'} q_*(s', a')). \tag{2.14}$$

Proof:

$$v_*(s) = \max_a q_*(s,a) \tag{2.15}$$
$$= \max_a \mathbb{E}_*[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \tag{2.16}$$
$$= \max_a \sum_{s',r} r p(s', r|s, a) + \gamma v_*(s') p(s', r|s, a). \tag{2.17}$$

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \tag{2.18}$$
$$= \sum_{s',r} r p(s', r|s, a) + \gamma \sum_{r,s'} p(s', r|s, a) \mathbb{E}[v_*(s') | S_t = a, A_t = a, S_{t+1} = s'] \tag{2.19}$$
$$= \sum_{s',r} p(s', r|s, a)(r + \gamma \max_{a'} q_*(s', a')) \tag{2.20}$$

3 Dynamic programming

Dynamics programming for reinforce learning is divide into two parts: 1. Policy evaluation, 2. Policy iteration.

3.1 Policy evaluation

Policy evaluation is used Bellman equation to compute the value function for a given policy π.

Input π, the policy to be evaluated.
Initialize $V(s)$, such as, $V(s) \leftarrow 0$.
Repeat
    $\Delta \leftarrow 0$.
    For $s \in \mathcal{S}$
        $v \leftarrow V(s)$,
        $V(s) = \sum \pi(a|s) \sum_{s',r} p(s', r|s, a)(r + \gamma v_\pi(s'))$*.
        $\Delta \leftarrow \max(0, v - V(s))$
    until $\Delta < \theta$(a small positive number)
return $V(s)$.

where $*$ step, we can choose inplace operation, or two-array version. In most cases, in-place converges more rapidly. For in-place case, the order in which states are backed up during the sweep has a significant influence on converging rate.

3.2 Policy iteration

Theorem 3.2.1 (Policy improvement theorem)**.** *For two deterministic policies π, π', if*

$$q_\pi(s, \pi'(s)) \geq v_\pi(s), \forall s \in \mathcal{S}, \tag{3.1}$$

then, we have,

$$v_{\pi'}(s) \geq v_\pi(s). \tag{3.2}$$

Proof:

$$
\begin{aligned}
v_\pi(s) &\le q_\pi(s, \pi'(s)) && (3.3)\\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] && (3.4)\\
&\le \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] && (3.5)\\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2})] | S_t = s] && (3.6)\\
&\cdots && (3.7)\\
&\le \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s] && (3.8)\\
&= v_{\pi'}(s) && (3.9)
\end{aligned}
$$

Using this theorem, we can design a greedy algorithm to update our policy such that the short term is very large.

$$\pi'(s) = \arg\max_a q_\pi(s, a) \tag{3.10}$$

And now, we can prove that this short term optimal updating rule will make our policy converge to one satisfying Bellman optimality equation.

$$v_{\pi'}(s) = \max_a q_\pi(s, a) = \max_a q_{\pi'}(s, a), \tag{3.11}$$

where the second equality is because the convergence.
Notes: Everything here in the deterministic scheme can be extended to random policy. We just need to keep all submaximal actions has zero probability.

policy stable $\leftarrow$ true.
For each $s \in \mathcal{S}$
    $a \leftarrow \pi(s)$
    $\pi(s) \leftarrow \arg\max_{a'} \sum_{s',r} p(s', r|s, a')(r + \gamma V(s'))$
    if $a \neq \pi(s)$, policy stable $\leftarrow$ false.
If policy stable is true, return π, V.

3.3 Practical implement

The simplest algorithm to combine these two process, policy evaluation and policy iteration is,

$$\pi_0 \to v_{\pi_0} \to \pi_1 \to v_{\pi_1} \to ....\pi_* \to v_*$$

However, this trivial algorithm has obivious disadvantage. To obtain value function, we need to iterate Bellman equation many many times, theoretically infinite, but in most cases, the approximate value function $\widetilde{v}$ is enough to do policy improvement. Hence, we can use the following algorithm

$$\pi_0 \to \widetilde{v}_{\pi_0} \to \pi_1 \to \widetilde{v}_{\pi_1} \to \cdots \to \pi_* \to v_*$$

However, if the state is very large, for each iteration, we need to sweep through the whole space, which is very expensive, we can just update value function for one state, by the formula

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)(r + \gamma v_k(s')) \tag{3.12}$$

This algorithm does not decrease complexity generally. But in some problems, if we only are interested in partial state of the state, we can choose proper state sequences $\{s_k\}$ for this algorithm.

4 Monte Carlo Methods

In most applications, we can not know the environment very exactly, which means the ture transistion probability $p(s', r|s, a)$ is unknown. Hence, we need to do estimation and Monte Carlo Methods using sampling method to estimate state-value function and state-action value function. In this setting, whatever Monte Carlo Methods or TD methods, policy evalution can be divided in to two kinds, on-policy and off-policy.

4.1 On-policy

Initialize:
    $\pi \leftarrow$ policy to be evaluated
    $V \leftarrow$ an arbitrary state-value function
    $Returns(s) \leftarrow$ an empty list, for all $s \in S$
Repeat forever:
    Generate an episode using π
    For each state s appearing in the episode:
        $G \leftarrow$ return following the first occurence of s
        Append G to $Returns(s)$
        $V(s) \leftarrow$ Average of $Returns(s)$

Here, we have used first-visit MC methods. Also every-visit MC method extends more naturally to function approximation and eligibility traces. This algorithms can be extended to estimate state-action value function. For control problems, we also use GPI(general policy iteration) scheme, which is induced by greedy algorithm,

$$\pi_0 \to q_{\pi_0} \to \pi_1 \to q_{\pi_1} \to \pi_2 \to \cdots \to \pi_* \to q_* \tag{4.1}$$

And policy iteration is updated by the following formula,

$$\pi_{k+1}(s) = \arg\max_a q_{\pi_k}(s, a). \tag{4.2}$$

Notes: Here we have two assumptions: 1. policy evaluation can be done with infinite number of episodes; 2. episodes have exploring starts. To remove the first assumptions, we have two ways: The first one is to evaluate value function as much steps as you can, and then you will obtain a function very close to true value function, but it wastes much

resources. The second one is to use GPI scheme. For Monte Carlo policy evaluation it is natural to alternate between evaluation and improvement on an episode-by-episode basis. To remove the second assumption, we can always method, so called $\varepsilon-$greedy. Before we explain it, we should introduce an important concept-*soft*. A policy is called soft, meaning that $\pi(a|s) > 0.\forall a \in S, \forall a \in \mathcal{A}(s)$, but gradually shifted closer and closer to a deterministic optimal policy. Now, we can introduce ε-greedy algorithm. That is, all nongreedy actions are given the minimal probability of selcetion, $\frac{\varepsilon}{|\mathcal{A}(s)|}$, and the remaining bulk of the probability, $1-\varepsilon+\frac{\varepsilon}{|\mathcal{A}(s)|}$, is given to the greedy action. Afterall, the algorithm can be written as,

Initialize, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
  $Q(s,a) \leftarrow$ arbitrary
  $Returns(s,a) \leftarrow$ empty list
  $\pi(s|a) \leftarrow$ an arbitrary ε-soft policy.
Repeat forever:
  Generate an episode using π
  For each pari s, a appearing in the epsiode:
    $G \leftarrow$ return following the first occurrence of s, a
    Append G to $Returns(s,a)$
    $Q(s,a) \leftarrow$ average of $Returns(s,a)$.
  For each s in the episode:
    $a^* \leftarrow \arg\max_a Q(s,a)$
    For all $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} 1-\varepsilon+\varepsilon/|\mathcal{A}(s)|, a = a^*, \\ \varepsilon/|\mathcal{A}(s)|, a \neq a^* \end{cases} \tag{4.3}$$

4.2 Off-policy

Off-policy is method using samples generated by a different policy μ, to estimate value function of interested policy π. In order to implement this method, we must have the following assumption–*coverage*: If $\pi(a|s) > 0$, and then $\mu(a|s) > 0$. For a particular trajectory $A_t, S_{t+1}, A_{t+1}, \cdots, S_T$, we can find the relative probability, called *importance-sampling ratio*. The absolute probability of this trajectory is,

$$\prod_{j=t}^{T-1} \pi(A_j|S_j) p(S_{j+1}|S_j, A_j)$$

And now, we can compute the relative probability,

$$\rho_t^T = \frac{\prod_{j=t}^{T-1} \pi(A_j|S_j) p(S_{j+1}|S_j, A_j)}{\prod_{j=t}^{T-1} \mu(A_j|S_j) p(S_{j+1}|S_j, A_j)} = \prod_{j=t}^{T-1} \frac{\pi(A_j|S_j)}{\mu(A_j|S_j)}, \tag{4.4}$$

which is independent of transition proability, which usually is unknown for us. And then, we have two approaches to do sampling, one is *ordinary importance sampling*:

$$V(s) = \frac{\sum_{t\in\mathcal{T}(s)} \rho_t^{T(t)} G_t}{|\mathcal{T}(s)|}, \tag{4.5}$$

where $\mathcal{T}(s)$ is the collection of first occurence time for all samples. The other is *weighted importance sampling*,

$$V(s) = \frac{\sum_{t\in\mathcal{T}(s)} \rho_t^{T(t)} G_t}{\sum_{t\in\mathcal{T}(s)} \rho_t^{T(t)}}. \tag{4.6}$$

The first algorithm can be understood very easily by large number law,

$$\begin{aligned}
\mathbb{E}_\pi[G] &= \int \mathcal{D}\phi p_\pi(\phi) G(\phi) && (4.7)\\
&= \int \mathcal{D}\phi p_\mu(\phi) \frac{p_\pi(\phi)}{p_\mu(\phi)} G(\phi) && (4.8)\\
&\approx \frac{1}{|\Omega|} \sum_{\phi\in\Omega} \frac{p_\pi(\phi)}{p_\mu(\phi)} G(\phi) && (4.9)
\end{aligned}$$

And the second is can be understood as,

$$1 = \int \mathcal{D}\phi p_\mu(\phi) \frac{p_\pi(\phi)}{p_\mu(\phi)} = \frac{1}{|\Omega|} \sum_{\phi\in\Omega} \frac{p_\pi(\phi)}{p_\mu(\phi)} \tag{4.10}$$

The summation are all sampled with measure p_μ. *Notes:* The first one is unbiased, but has possibility with diverged variance. The second is biased, but with bounded variance. But if we directly use this algorithm, the memory cost is very large, since we need to store each G_t. To solve it, we use incremental implementation by introducing another quantity C_n. Then we can iterate value function $V_n = \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k}$ by,

$$\begin{aligned}
V_{n+1} &= V_n + \tfrac{W_n}{C_n}(G_n - V_n), && (4.11)\\
C_{n+1} &= C_n + W_{n+1}, && (4.12)
\end{aligned}$$

where $C_0 = 0$.

Initialize, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
    $Q(s,a) \leftarrow$ arbitrary
    $C(s,a) \leftarrow 0$
    $\mu(a|s) \leftarrow$ an arbitrary soft behavior policy
    $\pi(a|s) \leftarrow$ an arbitrary target policy
Repeat forever:
    Generate an episode using μ:
        $S_0, A_0, R_1, \cdots, S_{T-1}, A_{T-1}, R_T, S_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    For $t = T-1, T-2, \cdots$ downto 0:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(A_t,S_t)(G-Q(S_t,A_t))}$
        $W \leftarrow W \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}$
        If $W = 0$ then ExitForLoop

For control problem, we need not obtain an very exact value function. Instead, we can alternate exploration and exploitation episode-by-episode.

Initialize, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
    $Q(s,a) \leftarrow$ arbitrary
    $C(s,a) \leftarrow 0$
    $\pi(s) \leftarrow$ a deterministic policy that is greedy with respect to Q
Repeat forever:
    Generate an episode using any soft policy μ:
        $S_0, A_0, R_1, \cdots, S_{T-1}, A_{T-1}, R_T, S_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    For $t = T-1, T-2, \cdots$ downto 0:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t,A_t)}(G - Q(S_t, A_t))$
        $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$
        $W \leftarrow W \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}$
        If $W = 0$ then ExitForLoop.

5 Temporal-Difference learning

If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference(TD) learning. As we known, Monte Carlo use the

following formula to do evaluation,

$$V(S_t) \leftarrow V(t) + \alpha(G_t - V(S_t)) \tag{5.1}$$

which requires us to wait for termination of one episode. And TD methods only need us wait unitl next time. For example, the simplest TD method, $TD(0)$, is

$$V(S_t) \leftarrow V(t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)). \tag{5.2}$$

Since TD method bases its update in part on an existing estimate, we say it is a bootstrapping method, like DP. And now we can compare these three methods,

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] && (5.3)\\ &= \mathbb{E}_\pi[\sum_{k=0} \gamma^k R_{t+k+1}|S_t s] && (5.4)\\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]. && (5.5) \end{aligned}$$

Monte Carlo use the first line to do estimation. Since we do not exact transition probability, we use return G_t in each episode to replace 'true' G_t, and obtain the expectation with many many samples. DP actually use the last equation, and in DP setting, we know the exact transition probability, so we can find $v_\pi(s)$ by back-up iteration directly. TD method uses the last equation too, instead, we do not know transition probability. Hence, we use $R_{t+1}, v_\pi(S_{t+1})$ appears in each sampled episode to represent the true random variable.

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily
Repeat for each epsiode:
    Initialize S
    Repeat for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A: observe reward, R, and next state, S'
        $V(S) \leftarrow V(S) + \alpha(R + \gamma V(S') - V(S))$
        $S \leftarrow S'$
    until S is terminal.

Notes: The advantage of TD methods have an advantage over DP methods in that they do not require a model of the environment, of its reward and next-state probability distributions. The next advantage of TD methods over Monte Carlo is that they are naturally implemented in an on-line, fully incremental fashion.

Also, fot the control problems, TD control is divided into two: on-policy(SARSA), off-policy(Q-learning):

5.1 SARSA

SARSA means,

$$S_t \to A_t \to R_{t+1} \to S_{t+1} \to A_{t+1} \to \cdots$$

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}$, arbitrarily, and $Q(terminalstate, \cdot) = 0$
Repeat for each episode:
    Initialize S
    Choose A from $\mathcal{A}(S)$ using policy derived from Q(e.g. $\varepsilon-$ greedy)
    Repeat for each step of episode:
        Take action A, observe R, S'
        Choose A' from S' using policy derived from $Q(\varepsilon-$ greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S', A') - Q(S,A))$
        $S \leftarrow S', A \leftarrow A'$
    until S is terminal.

From the above, we can see SARSA is on-policy control.

5.2 Q-learning

Q-learning is the most important breakthroughs in reinforce learning, which can be stated by,

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)). \tag{5.6}$$

The advantage of this algorithm is early convergence to q_*.

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}$, arbitrarily, and $Q(terminated, \cdot) = 0$
Repeat for each episode
    Initialize S
    Repeat for each step of episode:
        Choose A from $\mathcal{A}(S)$ using policy derived from Q(i.e $\varepsilon-$ greedy)
        Take action A, observe R, S'
        $Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma \max_a Q(S', a) - Q(S,A))$
        $S \leftarrow S'$
    until S is terminal.

6 Eligibility Traces

$TD(0)$ and Monte Carlo uses one step and one episode date to update value function. How about intermediate methods? For example, we use n steps rewards to update.

$$G_t^{t+n}(c) = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n c, \tag{6.1}$$

where c is a quantity we use to offset the difference between G_t and $R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n R_{t+n}$. Normally, the choise is $c = V_t(S_{t+n})$. As usual, we can update the value function on-line(step-by-step) or off-line(episode-by-episode). Specificly, on-line updating formula is,

$$V_{t+1}(s) = V_t(s) + \Delta_t(s), \tag{6.2}$$

where,

$$\Delta_t(S_t) = \alpha[G_t^{t+n}(V_t(S_{t+n})) - V_t(S_t)] \tag{6.3}$$

Off-line updating formula is given as,

$$V_{t+1}(s) = V_t(s), t < T-1, V_T(s) = V_{T-1} + \sum_{t=1}^{T-1} \Delta_t(s). \tag{6.4}$$

6.1 Forward TD(λ)

In general, we extend the above idea to an approach which seems to be more comprehensive. We do average over all 1-step, 2-step... For example, we can choose $\Delta_t(S_t) = \alpha(\frac{1}{2}G_t^{t+2} + \frac{1}{2}G_{t+1}^{t+5} - V_t(S_t))$. Any weighted choice is OK. However, the normal choice is,

$$L_t = (1-\lambda)\sum_{n=1}^{+\infty} \lambda^{n-1} G_t^{t+n}(V_t(S_{t+n})) \tag{6.5}$$

If process is terminated at finite T, we can do as the previous section by defining terminated state as self-absorbing state with reward 0, then we have,

$$L_t = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{t+n}(V_t(S_{t+n})) + \lambda^{T-t-1} G_t. \tag{6.6}$$

And then increment is,

$$\Delta_t(S_t) = \alpha(L_t - V_t(S_t)). \tag{6.7}$$

Notes: Why we use TD(λ) instead of n-steps TD is the latter is very difficult to implement in practise, since if we want update V_t, we should look forward n steps. So far, TD(λ) seems to have the same disadvantages, but there were some excellent people, who developed an equivalent, practical algorithm to TD(λ), whatever off-line, on-line updating. on-line updating equivalence is a very new result appeared in 2014.

6.2 Backward TD(λ)

We introduce a quantity, called eligibility trace $E_t(s)$, which updated as,

$$E_t(S_t) = (1-\beta)\lambda\gamma E_t(S_t) + 1, \tag{6.8}$$

$$E_t(s) = \lambda\gamma E_t(s), \forall s \neq S_t. \tag{6.9}$$

This eligibility trace is called *dutch traces*, and if $\beta = 1$, it is called *replacing traces*, if $\beta = 0$, it is called *accumulating traces*. Also we define,

$$\delta_t = R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t), \tag{6.10}$$

where, we should emphasis to obtain δ_t, we just need to look forward one step. And then TD error is,

$$\Delta V_t(s) = \alpha\delta_t E_t(s), \forall s \in \mathcal{S}. \tag{6.11}$$

And now, we can prove the equivalence, when we use off-line updating. To this end, we should prove,

$$\sum_{t=1}^{+\infty} \Delta_t(s) = \sum_{t=1}^{+\infty} \Delta V_t(s) \leftrightarrow \sum_{t=1}^{+\infty} L_t - V_t(s) = \sum_{t=1}^{+\infty} \delta_t E_t(s). \tag{6.12}$$

For simplicity, we assume s only has one click at time τ, which can be extended to many clicks easily and $\beta = 0$. Then,

$$E_t(s) = (\lambda\gamma)^{t-\tau} 1_{\tau \leq t < T}, \forall t < T \tag{6.13}$$

$$\begin{aligned}
\text{LHS} &= L_\tau - V_t(s) && (6.14)\\
&= (1-\lambda) \sum_{n=1}^{+\infty} \lambda^{n-1} G_\tau^{\tau+n}(V(S_{\tau+n})) - V(s) && (6.15)\\
&= (1-\lambda) \sum_{n=1}^{+\infty} \lambda^{n-1} \left(\sum_{k=0}^{n-1} \gamma^k R_{\tau+k+1} + \gamma^n V(S_{\tau+n}) \right) - V(s) && (6.16)
\end{aligned}$$

$$\begin{aligned}
\text{RHS} &= \sum_{n=1}^{+\infty} (R_{\tau+n} + \gamma V(S_{\tau+n}) - V(S_{\tau+n-1}))(\lambda\gamma)^{n-1} && (6.17)\\
&= \sum_{n=1}^{+\infty} (\lambda\gamma)^{n-1} R_{\tau+n} + \lambda^{n-1}\gamma^n(1-\lambda)V(S_{\tau+n}) - V(s). && (6.18)
\end{aligned}$$

And now, we need to do simplification,

$$\begin{aligned}
(1-\lambda) \sum_{n=1}^{+\infty} \lambda^{n-1} \sum_{k=1}^{n} \gamma^{k-1} R_{\tau+k} &= (1-\lambda) \sum_{k=1}^{+\infty} \sum_{n=k}^{+\infty} \lambda^{n-1}\gamma^{k-1} R_{\tau+k} && (6.19)\\
&= \sum_{k=1}^{+\infty} \lambda^{k-1}\gamma^{k-1} R_{\tau+k} && (6.20)
\end{aligned}$$

Hence, we have proved equivalence bewtween forward TD(λ) and backward TD(λ), when using off-line updating. The on-line proof is more difficult, since V_t changes at each time. Whatever, we can write the alogrithm as,

Initialize $V(s)$ arbitrarily(but set to 0 if s is terminated)
Repeat for each episode:
    Initialize $E(s) = 0$, for all $s \in \mathcal{S}$
    Initialize S
    Repeat for each step of episode:
        Take action A, observe reward R, and next state, S'
        $\delta \leftarrow R + \gamma V(S') - V(S)$
        $E(S) \leftarrow (1 - \beta)E(S) + 1$
        For all $s \in S$:
            $V(s) \leftarrow V(s) + \alpha\delta E(s)$
            $E(s) \leftarrow \gamma\lambda E(s)$
        $S \leftarrow S'$
    until S is terminal.

Also, there is an improved version, called *True online TD*(λ).

$$V_{t+1}(s) = V_t(s) + \alpha[\delta_t + V_t(S_t) - V_{t-1}(S_t)]E_t(s) - \alpha 1_{s=S_t}[V_t(S_t) - V_{t-1}(S_t)]. \tag{6.21}$$

Initialize $V(s)$ arbitrarily(but set to 0 if s is terminated)
$V_{\text{old}} \leftarrow 0$
Repeat for each episode:
    Initialize $E(s) = 0, \forall s \in \mathcal{S}$
    Initialize S
    Repeat for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe reward, R, and next state, S'
        $\Delta \leftarrow V(S) - V_{\text{old}}$
        $V_{\text{old}} \leftarrow V(S')$
        $\delta \leftarrow R + \gamma V(S') - V(S)$
        $E(S) \leftarrow (1 - \alpha)E(S) + 1$
        For all $s \in \mathcal{S}$:
            $V(s) \leftarrow V(s) + \alpha(\delta + \Delta)E(s)$
            $E(s) \leftarrow \gamma\lambda E(s)$
        $V(S) \leftarrow V(S) - \alpha\Delta$
        $S \leftarrow S'$
    until S is terminal.

6.3 SARSA(λ)

For control problem, we use the following formulas to compute state-action function,

$$E_t(s,a) = \gamma\lambda E_{t-1}(s,a) + 1_{s=S_t}1_{a=A_t}, \textbf{accumulating} \tag{6.22}$$

$$E_t(s,a) = (1-\alpha)\gamma\lambda E_{t-1}(s,a) + 1_{s=S_t}1_{a=A_t}, \textbf{dutch} \tag{6.23}$$

$$E_t(s,a) = (1 - 1_{s=S_t}1_{a=A_t})\gamma\lambda E_{t-1}(s,a) + 1_{s=S_t}1_{a=A_t}\textbf{replacing} \tag{6.24}$$

And

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t E_t(s,a), \tag{6.25}$$

where,

$$\delta_t = R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_t(S_t, A_t) \tag{6.26}$$

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in A$
Repeat for each episode:
$E(s,a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$
Initialize S, A
Repeat for each step of episode:
Take action A, observe R, S'
Choose A' from S' using policy derived from Q (e.g. ε-greedy)
$\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
$E(S,A) \leftarrow E(S,A) + 1$
or $E(S,A) \leftarrow (1-\alpha)E(S,A) + 1$
or $E(S,A) \leftarrow 1$
For all $s \in \mathcal{S}, a \in \mathcal{A}$:
$Q(s,a) \leftarrow Q(s,a) + \alpha\delta E(s,a)$
$E(s,a) \leftarrow \lambda\gamma E(s,a)$
$S \leftarrow S', A \leftarrow A'$
until S is terminal.