Reinforcement Learning Equations

Aaron Hao Tan

1 Finite Markov Decision Processes

Components of MDP : Decision epochs, State, Action, Transition probability, Rewards

$$\{T, S, A_s, p_t(\cdot|s,a), r_t(s,a)\} \tag{1}$$

A state is *Markov* if and only if

$$\mathbb{P}\left[S_{t+1}|S_t\right] = \mathbb{P}\left[S_{t+1}|S_1, \ldots, S_t\right] \tag{2}$$

State-transition probabilities (p characterize the environment's dynamics). More often used than equations 4 and 5.

$$\begin{aligned} p(s'|s,a) &= Pr[S_t = s'|S_{t-1} = s, A_{t-1} = a] \\ &= \sum_{r\in\mathcal{R}} p(s', r|s, a) \end{aligned} \tag{3}$$

Expected rewards for state-action pairs and for state-action-next-state triples. The current reward depends on the previous state and action.

$$r(s,a) \doteq \mathbb{E}\left[R_t|S_{t-1} = s, A_{t-1} = a\right] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p\left(s', r|s, a\right) \tag{4}$$

$$r\left(s,a,s'\right) \doteq \mathbb{E}\left[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'\right] = \sum_{r\in\mathcal{R}} r\frac{p\left(s', r|s, a\right)}{p\left(s'|s, a\right)} \tag{5}$$

Policy
- Probabilistic is inferior to deterministic

$$\pi(a|s) = Pr(A_t = a|S_t = s) \tag{6}$$

Returns (sum of rewards)

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \tag{7}$$

Discounted Returns (γ = discount rate). If $\gamma < 1$, the infinite sum in equation 8 would have a finite value. If $\gamma = 0$, the agent is concerned with only maximizing immediate rewards (myopic). If the reward is +1, the return is equivalent to $G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{8}$$

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots\right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{9}$$

State Value Function
- Value of terminal state (if any) is always zero.
- The expected return (expected cumulative future discounted reward) starting from that state

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \quad \forall s \in S \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1}|S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi\left(S_{t+1}\right)|S_t = s\right] \end{aligned} \tag{10}$$

Bellman Equation for v_π : the value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way. Assumes discrete and is invariant of time (time doesn't matter).

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma \cdot G_{t+1} | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s',r} p\left(s', r|s, a\right)\left[r + \gamma v_\pi\left(s'\right)\right] \quad \forall s \in S \end{aligned} \tag{11}$$

Action Value Function

$$q_\pi(s, a) \doteq \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \tag{12}$$

Bellman Equation for $q_\pi(s, a)$: policy is included to choose action a' from s'

$$q_\pi(s, a) = \sum_{s',r} p\left(s', r|s, a\right)\left[r + \gamma \sum_{a'} \pi\left(a'|s'\right) q_\pi\left(s', a'\right)\right] \tag{13}$$

The following figure represents the state value and action state value functions graphically.

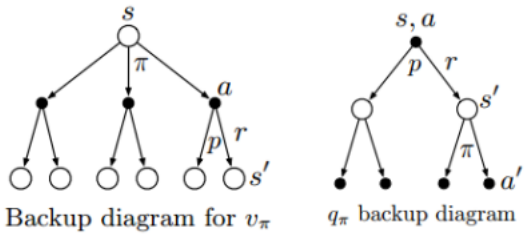

Figure 1: Value and State-Value Functions

Optimal State Value Function A policy π is better or equal to another policy π' if its expected return is greater than or equal to that of π' for all states ($\pi' \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$).

$$v_*(s) \doteq \max_\pi v_\pi(s) \tag{14}$$

Optimal Action Value Function and its relation to v_* . Equation 15 gives the expected return for taking action a in state s and thereafter following an optimal policy. Comparing with equation 11, $v_\pi(s)$ represents the value of a state as the summation of future returns. In equation 15, $v_*\left(S_{t+1}\right)$ gives the optimal value of all future states from the given state and action.

$$\begin{aligned} q_*(s, a) &\doteq \max_\pi q_\pi(s, a) \\ &= \mathbb{E}\left[R_{t+1} + \gamma v_*\left(S_{t+1}\right) | S_t = s, A_t = a\right] \end{aligned} \tag{15}$$

Relationship between q_* **and** v_*
- The value of a state under an optimal policy must equal the expected return for the best action from that state.
- Second last line of equation 16 shows that $G_t = R_{t+1} + \gamma v_*(S_{t+1})$ when following π_*.
- The last line is the same as equation 17. For finite MDPs, the last line of equation 16 has a unique solution.
- R_{t+1} has nothing to do with policy because the first action was not taken with policy
- max operator turns equation 16 in to a system of nonlinear equations

$$\begin{aligned} v_*(s) &= \max_{a \in A(s)} q_*(s, a) \\ &= \max_a q_*(s, a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \end{aligned} \tag{16}$$

Bellman Optimality Equations

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_*\left(S_{t+1}\right) | S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p\left(s', r | s, a\right)\left[r + \gamma v_*\left(s'\right)\right] \end{aligned} \tag{17}$$

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*\left(S_{t+1}, a'\right) | S_t = s, A_t = a\right] \\ &= \sum_{s',r} p\left(s', r | s, a\right)\left[r + \gamma \max_{a'} q_*\left(s', a'\right)\right] \end{aligned} \tag{18}$$

In v_*, the next best action is selected based on the expected reward and value of future states. In q_*, the state and action is given as well as the reward. From the new state s', the best action is chosen.

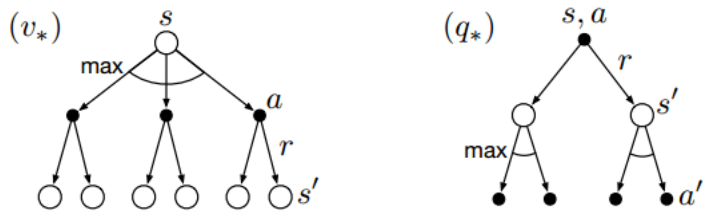

Figure 2: Bellman Optimality Equations

2 Dynamic Programming

Policy Evaluation
- turn Bellman equation from equation 11 in to an update rule
- maximum bootstrapping: $v_k(s')$ is using a previous estimate

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi\left[R_{t+1} + \gamma v_k\left(S_{t+1}\right) | S_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p\left(s', r | s, a\right)\left[r + \gamma v_k\left(s'\right)\right] \end{aligned} \tag{19}$$

Policy Evaluation Steps
- for state in states
—- for action in actions available in each state
——— calculate value of state using equation 19
——— (value = sum all values of each action together for each state (ie. 4 actions in a state = 1 total value))
—- replace old value of state with newly calculated value
—- calculate the change in value of each state with only the maximum difference remembered
- stop looping if the maximum change in state values falls below a threshold

Policy Improvement
- greedy policy = guaranteed to be optimal
- one-step greedy lookahead

$$\begin{aligned}\pi'(s) &\doteq \arg\max_a q_\pi(s,a) \\ &= \arg\max_a \mathbb{E}\left[R_{t+1} + \gamma v_\pi\left(S_{t+1}\right) | S_t = s, A_t = a\right] \\ &= \arg\max_a \sum_{s',r} p\left(s', r | s, a\right)\left[r + \gamma v_\pi\left(s'\right)\right]\end{aligned} \tag{20}$$

Policy is represented $s \times a$ table where each row represents a state and each column represents an action. The value within each entry of the table represent the probability of taking that action, when in that state, using a probabilistic policy.

At this stage, there is a value calculated for every state using policy evaluation already.

Policy Improvement Steps
- for state in states
—- choose the best action with the current policy (argmax)
—- for action in actions available in each state
——— calculate action values for each action using equation 20 without the argmax
———(this creates a $1 \times a$ vector where each element is an action value q)
—- select the best action in the $1 \times a$ vector (equation 20 with argmax)
—- if action chosen with the current policy is not the calculated best action
——— policy is unstable
—- update the policy with the new best action

Policy Iteration
- built on the fundamentals of value iteration, more efficient (CPU time) than value iteration.

To solve calculation problems:
- Initiate policy
- Write equations for value functions using equation 19
- Solve for Values at each state with above equations
- Use the calculated Values and a new action to update the value at each state
- If the value is higher, update the policy

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r | s, \pi(s))\left[r + \gamma V(s')\right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r | s, a)\left[r + \gamma V(s')\right]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Figure 3: Policy Iteration

Value Iteration
- Policy iteration is faster because policy converges quicker than value functions
- Bellman Optimality Equation (equation 17) turned in to an update rule
- Formally requires infinite number of iterations to converge exactly to v_* but we stop before then
- Only guarantees $\epsilon - optimality$ (theoretically)
- All algorithms converge to an optimal policy for discounted finite MDPs
- When γ is close to 1, iteration goes forever = making value iteration inefficient
- Policy iteration converges faster even for smaller γ
- Main difference between Value Iteration and Policy Iteration
—- In value iteration, instead of summing values from all actions per state, take only the best action value per state as the value of that state
—- Value iteration update is identical to policy evaluation except that the max is taken over all actions

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k\left(S_{t+1}\right) | S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p\left(s', r | s, a\right)\left[r + \gamma v_k\left(s'\right)\right] \end{aligned} \tag{21}$$

Value Iteration Steps
- for state in states
—- for action in actions available in each state
——— calculate action values for each action using equation 21 without the max
———(this creates a $1 \times a$ vector where each element is an action value q)
—- value = the best action value in the $1 \times a$ vector (equation 21 with max)
—- calculate the change in value of each state with only the maximum difference remembered
- stop looping if the maximum change in state values falls below a threshold
- for state in states
—- for action in actions available in each state
——— calculate action values for each action using equation 21 without the max
———(this creates a $1 \times a$ vector where each element is an action value q)
—- select the best action in the $1 \times a$ vector (equation 21 with argmax)
—- overwrite policy table with best action

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a)\left[r + \gamma V(s')\right]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \text{argmax}_a \sum_{s',r} p(s', r | s, a)\left[r + \gamma V(s')\right]$

Figure 4: Value Iteration

To solve calculation problems:
- Initiate value at each state
- Use equation 21 to obtain the max value at each state (use all actions and the subsequent states)
- Repeat above step
- Use argmax to update the policy with the best action

Summary of Policy and Value Iteration
- Value Iteration: Find optimal value, then extract optimal policy
- Policy Iteration: Find value, then extract optimal policy

3 Monte Carlo Methods

- Solving RL based on averaging sample returns
- Return after taking an action in one state depends on the actions taken in later states in the same episode
- No boot strapping (DP = max bootstrapping)
- can be considered as a way to go from π to v

First-visit MC Method
- average of the returns following first visit to s
- more popular, samples are independent (faster convergence)
- poor sample efficiency

Every-visit MC Method
- averages the returns following all visits to s
- more natural to function approximation/eligibility traces
- possibly statistically unstable

The following shows first-visit MC methods.

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated
Initialize:
  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
      Append G to $Returns(S_t)$
      $V(S_t) \leftarrow$ average$(Returns(S_t))$

Figure 5: First-visit MC Prediction

Problem with prior, there will only be one action from each state if following a deterministic policy. To fix this:
- Look at state and action pairs as oppose to just states
- Give all state action pair a nonzero probability of being selected (line 6)
- Includes a greedy policy at the end to pick the best action in the next iteration (line 14)
- Cons: this solution is unlikely to learn from real experiences but simulated episodes are fine

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:
  $\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
  $Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
  $Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
  Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
  Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
      Append G to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow$ average$(Returns(S_t, A_t))$
      $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

Figure 6: Exploring Start MC Control

On-policy vs Off-policy Methods
On-policy methods attempt to evaluate/improve the policy that is used to make decisions, whereas off-policy methods evaluate/improve a policy different from that used to generate data. On-policy approach is actually a compromise - it learns action values not for the optimal policy, but for a near optimal policy that still explores.
- On-policy: unbiased, low bias and variance
- Off-policy: biased, high bias and high variance. Slower to converge.

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$
Initialize:
$\pi \leftarrow$ an arbitrary ε-soft policy
$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):
Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
Append G to $Returns(S_t, A_t)$
$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
$A^* \leftarrow \text{argmax}_a Q(S_t, a)$ (with ties broken arbitrarily)
For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Figure 7: On-policy first-visit MC Control

ϵ - greedy Exploration
- with probability ϵ, select an action at random
- all non-greedy actions are given the minimal probability of selection, $\frac{\varepsilon}{|\mathcal{A}(s)|}$
- remaining bulk of probability is given to the greedy action, $1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}(s)|}$
- the following equation either chooses the greedy action A^* or not based on the probabilities stated

$$\pi\left(a|S_t\right) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/\left|\mathcal{A}\left(S_t\right)\right| & \text{if } a = A^* \\ \varepsilon/\left|\mathcal{A}\left(S_t\right)\right| & \text{if } a \neq A^* \end{cases} \tag{22}$$

Off-policy Evaluation via Importance Sampling
- Target policy: policy being learned about. Becomes deterministic (greedy) optimal policy.
- Behavior policy: policy used to generate behavior. Remains stochastic and more exploratory.

Importance Sampling Ratio
In order to use episodes from b to estimate values for π, we require that every action taken under π is also taken, at least occasionally, under b. That is, the following is assumed to be true (known as *coverage*). ISR eliminates bias from sample so that we can use unbiased samples for learning.

$$\pi(a|s) > 0 \quad \text{implies} \quad b(a|s) > 0 \tag{23}$$

We apply importance sampling to off-policy learning by weighting returns according to the relative probability of their trajectories occurring under the target and behavior policies. For example, given a start state S_t, the probability of the subsequent state-action trajectory occuring under any policy π is the following. Note, A_T will take agent to S_{T+1} which is pass terminal state.

$$\begin{aligned} &\Pr\left\{A_t, S_{t+1}, A_{t+1}, \ldots, S_T | S_t, A_{t:T-1} \sim \pi\right\} \\ &\quad = \pi\left(A_t|S_t\right) p\left(S_{t+1}|S_t, A_t\right) \pi\left(A_{t+1}|S_{t+1}\right) \cdots p\left(S_T|S_{T-1}, A_{T-1}\right) \\ &\quad = \prod_{k=t}^{T-1} \pi\left(A_k|S_k\right) p\left(S_{k+1}|S_k, A_k\right) \end{aligned} \tag{24}$$

The relative probability of the trajectory under the target and behavior policies is shown below. Note, the ratio depends on the policies and not on the MDP. Note, $\rho_{t:T-1}$ is a scalar value.

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi\left(A_k|S_k\right) p\left(S_{k+1}|S_k, A_k\right)}{\prod_{k=t}^{T-1} b\left(A_k|S_k\right) p\left(S_{k+1}|S_k, A_k\right)} = \prod_{k=t}^{T-1} \frac{\pi\left(A_k|S_k\right)}{b\left(A_k|S_k\right)} \tag{25}$$

Convert returns G_t due to the behavior policy to expected returns under the target policy.

$$\mathbb{E}\left[\rho_{t:T-1} G_t | S_t = s\right] = v_\pi(s) \tag{26}$$

Ordinary Importance Sampling
- $\mathcal{T}(s)$ represent the time steps in which state s is visited (norm = total number of samples)
- T(t) = total # of steps in the episode where s was visited (since we are dealing with chained episodes: 0-100, 101-200...etc)
- To estimate $v_\pi(s)$, simply scale the returns by the ratios and average the results
- $\{G_t\}_{t \in \mathcal{T}(s)}$ are the returns that pertain to state s
- $\left\{\rho_{t:T(t)-1}\right\}_{t \in \mathcal{T}(s)}$ are the corresponding ratios
- First visit: unbiased, high variance (unbounded)
- Every visit: biased

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|} \tag{27}$$

Weighted Importance Sampling
- Weighted average
- First visit: biased (though the bias converges asymptotically to zero), variance converges to zero
- In practice, this method has dramatically lower variance and is preferred.
- Every visit: biased

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}} \tag{28}$$

Comparison of Ordinary and Weighted
If you only have one sample, weighted importance would equal G_t as the ratio in the numerator and denominator cancel out. For ordinary, the denominator would equal to 1, and the result is equivalent to $\rho_{t:T(t)-1} G_t$. If the ratio was 10, which indicates that the trajectory observed is 10 times as likely under the target policy as under the behavior policy.
- Ordinary: unbiased, high variance, weak
- Weighted: biased, low variance, strong

Incremental Implementation: Ordinary Importance Sampling

$$V_n = \frac{\sum_{k=1}^{n-1} W_k G_t}{n-1} \tag{29}$$

$$V_{n+1} = V_n + \frac{1}{n}\left(W_n G_n - V_n\right) \tag{30}$$

Incremental Implementation: Weighted Importance Sampling
- $W_i = \rho_{t_i:T(t_i)-1}$
- $G_1, G_2, \ldots, G_{n-1}$

To estimate the following,

$$V_n = \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k} \tag{31}$$

Need to keep V up to date as we obtain a single additional return G_n.

$$V_{n+1} = V_n + \frac{W_n}{C_n}\left[G_n - V_n\right] \tag{32}$$

In addition to keeping track of V_n, must also maintain a cumulative sum of the weights C_n given to the first n returns

$$C_{n+1} = C_n + W_{n+1} \tag{33}$$

Off-policy MC Prediction
- Off-policy with weighted importance sampling
- Can be applied to on-policy by choosing the target and behavior policies the same ($\pi = b$) and $W = 1$ where W is the importance sampling ratio.

Off-policy MC prediction (policy evaluation) for estimating $Q \approx q_\pi$

Input: an arbitrary target policy π
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \in \mathbb{R}$ (arbitrarily)
  $C(s,a) \leftarrow 0$

Loop forever (for each episode):
  $b \leftarrow$ any policy with coverage of π
  Generate an episode following b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$, while $W \neq 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
    $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$

Figure 8: Off-policy MC Prediction

Off-policy MC Control
- Since b was initiated as a *soft policy* (without coverage of π), that's why the second last line is there (for the chance that an action selected under the behavior policy does not exist under the target policy)
- The last equation has a 1 in the numerator because the target policy is updated with the argmax action 2 lines prior already, so the target policy is no longer outputting a probability.

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \in \mathbb{R}$ (arbitrarily)
  $C(s,a) \leftarrow 0$
  $\pi(s) \leftarrow \operatorname{argmax}_a Q(s,a)$ (with ties broken consistently)

Loop forever (for each episode):
  $b \leftarrow$ any soft policy
  Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
    $\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a)$ (with ties broken consistently)
    If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
    $W \leftarrow W \frac{1}{b(A_t|S_t)}$

Figure 9: Off-policy MC Control

4 Temporal-Difference Learning

TD:
- Biased samples because of bootstrapping - Sensitive to initial values (initialization of V or Q)
- Learn after transition: implemented in online - fully incremental fashion - used in continuing task
- More efficient: in practice, converge faster than MC

MC:
- Solely dependent on environment - insensitive to initial values
- No bias, high variance
- Learn at the end of episode: delayed learning (only used with episodic tasks)

When sample size is finite, may converge to different values.

Monte Carlo (every visit) - Prediction : need G_t (obtained at the end of the episode) to perform update

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)] \tag{34}$$

Temporal Difference TD(0) - Prediction : need just the next reward $R_{t+1} + \gamma V(S_{t+1})$ to perform update

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{35}$$

where the TD error is

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{36}$$

Tabular TD(0) for estimating v_π

Input: the policy π 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 π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  until S is terminal

Figure 10: One step TD

Solving calculation problems:
- Initiate all Values to zero
- Start simulation: observing first state and reward
- Emission: go to the next state and observe reward
- Calculate update using equation 35 and repeat

SARSA: On-policy TD Control
- Learn an action-value function ($q_\pi(s, a)$) rather than a state-value function
- Consider transitions from state-action pair to state-action pair, as oppose to state to state

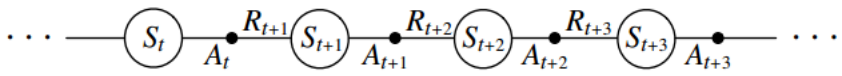

Figure 11: SARSA Diagram

SARSA Update

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\left[R_{t+1} + \gamma Q\left(S_{t+1}, A_{t+1}\right) - Q\left(S_t, A_t\right)\right] \tag{37}$$

- In the code below, line 8 is where the policy is being optimized (choosing the next best action based on the state arrived). Note, this code only returns Q and not Q^*.
- On policy because it estimates the return for state action pairs assuming the current policy continues to be followed. Same policy is used to transition and also update.

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., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-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 12: SARSA: On-policy TD Control

Q-learning: Off-policy Control
- Instead of choosing A' from an $\epsilon - greedy$ policy like in SARSA, choose the best action with max.
- Off policy because Q value is updated using the Q value of the next state and greedy action. In other words, it estimates the return for $Q(s, a)$ assuming a greedy policy were followed but is not actually following a greedy policy. One policy to choose A and another to update Q. If the policy used to choose A is greedy, the distinction between the two disappears.
- Behavior policy is the $\epsilon - greedy$
- Convergence is guaranteed under *Robinson-Mouro Condition*

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\left[R_{t+1} + \gamma \max_a Q\left(S_{t+1}, a\right) - Q\left(S_t, A_t\right)\right] \tag{38}$$

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., ε-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 13: Q-learning: Off-policy TD Control

Calculation problem:
- Initiate Q(s,a) values for all actions within each state
- Start simulation: pick action based on $\epsilon - greedy$
- Emission: uses equation 38 to update Q value
- Update policy by doing argmax (including the Q that was just calculated)

5 n-step Bootstrapping

- Bootstrapping works best if it is over a length of time in which a significant and recognizable state change has occurred
- Eligibility traces: enable bootstrapping over multiple time intervals simultaneously
- 2 step update = based on the first 2 rewards and the estimated value of teh state 2 steps later

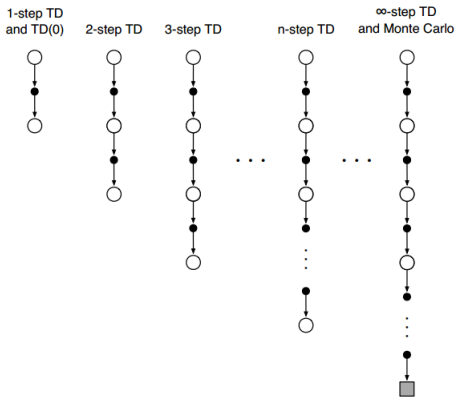

Figure 14: n-step Methods

Monte Carlo Return

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T \tag{39}$$

One Step Return

$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t\left(S_{t+1}\right) \tag{40}$$

Two Step Return

$$G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}\left(S_{t+2}\right) \tag{41}$$

n-step Return

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}\left(S_{t+n}\right) \tag{42}$$

n-step TD Prediction

- Line 15: adding up the rewards
- Example: $t = 3, n = 2, \tau = 2, G = \sum_{i=3}^{min(4,T)} = R_3 + \gamma R_4 + \gamma^2 V(S_4)$

n-step TD for estimating $V \approx v_\pi$

Input: a policy π
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for S_t and R_t) can take their index mod $n + 1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then $T \leftarrow t + 1$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha\left[G - V(S_\tau)\right]$
  Until $\tau = T - 1$

Figure 15: n-step TD Prediction

n-step SARSA
Switch states for state-action pairs and then use $\epsilon - greedy$ policy. Given t+1, t+n-1, t+n, what is the time of learning? Should be at time t.

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}\left(S_{t+n}, A_{t+n}\right), \quad n \geq 1, 0 \leq t < T - n \tag{43}$$

Similar to equation 37, the following is the update rule for n-step SARSA

$$Q_{t+n}\left(S_t, A_t\right) \doteq Q_{t+n-1}\left(S_t, A_t\right) + \alpha\left[G_{t:t+n} - Q_{t+n-1}\left(S_t, A_t\right)\right], \quad 0 \leq t < T \tag{44}$$

- n-step SARSA = SARARARAR...SA
- $\tau < 0$ = means not enough observations (less than n observations)
- need at least n samples/steps to begin learning
- learning happens from **if** $\tau \geq 0$

n-step Sarsa for estimating $Q \approx q_*$ or q_π

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then:
  | $T \leftarrow t+1$
  | else:
  | Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\left[G - Q(S_\tau, A_\tau)\right]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
  Until $\tau = T - 1$

Figure 16: n-step SARSA

n-step Off-policy Learning
Off-policy n-step TD
a_t is where deviation begins, and a_{t+n-1} is where deviation ends. a_{t+n} and s_{t+n} are not taken (not used for learning) but they are needed to calculate $G_{t:t+n}$.

$$V_{t+n}\left(S_t\right) \doteq V_{t+n-1}\left(S_t\right) + \alpha\rho_{t:t+n-1}\left[G_{t:t+n} - V_{t+n-1}\left(S_t\right)\right], \quad 0 \leq t < T \tag{45}$$

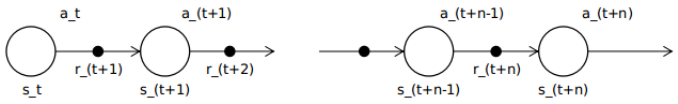

Figure 17: n-step Off-policy n-step TD

Importance Sampling Ratio

$$\rho_{t:h} \doteq \prod_{k=t}^{\min(h,T-1)} \frac{\pi\left(A_k|S_k\right)}{b\left(A_k|S_k\right)} \tag{46}$$

Off-policy n-step SARSA

Note, the ratio starts and ends one step later than off-policy n-step TD (equation 45) because the first action a_t was given. This is because we are updating a state-action pair. When updating a value function, the last action is not taken under policy. When updating a state-action pair, the last action was taken with behavior policy. In other words, we do not care how likely we were to select the action; now that we have selected it we want to learn fully from what happens, with importance sampling only for subsequent actions.

$$Q_{t+n}\left(S_t, A_t\right) \doteq Q_{t+n-1}\left(S_t, A_t\right) + \alpha\rho_{t+1:t+n}\left[G_{t:t+n} - Q_{t+n-1}\left(S_t, A_t\right)\right] \tag{47}$$

There is a typo in the code below where ρ is calculated. Instead of summing from $\tau+1$ to $\tau+n-1$ in line 20, should just be $\tau+n$ as described in equation 47 (the index shift). The last action is chosen by the behavior policy and therefore should be included in the importance sampling ratio's consideration.

One can convert the following code to Q-learning by adding $\max Q(s_{t+n}, a)$ to line 22 like in equation 38. In this case, the previous typo is no longer a mistake. Normally, in Q-learning, you do not need importance sampling ratio. But in n-step, there are n-transitions in which importance sampling ratio must be applied to until the last action which would be chosen by the greedy operator (behavior policy does not matter for the last action so ISR does not need to be applied). Additionally, ρ is applied to $G - Q$ instead of just G is because of the way the incremental implementation works.

Off-policy n-step Sarsa for estimating $Q \approx q_*$ or q_π

Input: an arbitrary behavior policy b such that $b(a|s) > 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be greedy with respect to Q, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim b(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then:
  | $T \leftarrow t+1$
  | else:
  | Select and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$
  | $\tau \leftarrow t-n+1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n-1,T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$ ($\rho_{\tau+1:t+n-1}$)
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau+n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ ($G_{\tau:\tau+n}$)
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho\left[G - Q(S_\tau, A_\tau)\right]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T-1$

Figure 18: n-step Off-policy SARSA

6 Eligibility Trace

2 Issues with n-step td
- which n to use?
- delayed learning (delaying the learning by n-transitions)

When TD methods are augmented with eligibility traces, they produce a family of methods spanning a spectrum that has MC at one end ($\lambda = 1$) and one step TD methods at the other ($\lambda = 0$)

TD(λ) can be implemented in an online and offline fashion.
- Offline: λ-return (forward view)
- Online: Backward view

λ return: Forward View
This is the basis for the forward view of eligibility trace used in TD(λ). This solves the problem of "which n to use?" by applying a weight to every n-step return. For example, when $\lambda = 0$, $n = 1$ is given 100% of the weight. When $\lambda = 0.4$, $n = 1 - 6$ are given a set of weights that decay as seen in figure 19.When $\lambda = 1$, zero weights is given to any of the $n - steps$ while the last term G_t taking full effect (creating MC).

In the offline case, λ-return IS TD(λ).

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t \tag{48}$$

$$\sum \text{weights} = \lambda^0 + \lambda + \lambda^2 + \cdots = \frac{1}{1-\lambda} \tag{49}$$

The average is calculated as shown below. In other words, when $\lambda = 0.4$, the outcome is similar to when n is equal to 1.67.

$$1 * 0.6 + 2 * 0.24 + 3 * 0.144... \tag{50}$$

	n=1	2	3	4	5	6	7	Avg
λ	$1-\lambda$	$(1-\lambda)\lambda$	$(1-\lambda)\lambda^2$	$(1-\lambda)\lambda^3$	$(1-\lambda)\lambda^4$	$(1-\lambda)\lambda^5$	$(1-\lambda)\lambda^6$	
0.000	**1.0000**	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	1.00
0.400	**0.6000**	**0.2400**	**0.1440**	**0.0346**	**0.0050**	**0.0002**	0.0000	1.67
0.800	**0.2000**	**0.1600**	**0.0320**	**0.0051**	**0.0002**	0.0000	0.0000	5.00
0.900	**0.1000**	**0.0900**	**0.0090**	**0.0008**	0.0000	0.0000	0.0000	10.00
0.950	**0.0500**	**0.0475**	**0.0024**	**0.0001**	0.0000	0.0000	0.0000	20.00
0.975	**0.0250**	**0.0244**	**0.0006**	0.0000	0.0000	0.0000	0.0000	40.00
0.990	**0.0100**	**0.0099**	**0.0001**	0.0000	0.0000	0.0000	0.0000	100.00
1.000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	#DIV/0!

Figure 19: λ vs. $n - step$

TD(λ): Backward View

Forward view is not directly implementable because at each step, you need to use info that will happen many steps later. The backward view is an incremental mechanism that approximates the forward view where the offline case of the backward view achieves the forward view exactly.

Eligibility trace has two decay factor, γ and λ (trace decay parameter). The current ET depends on the previous ET.

$$Z_t(s) \leftarrow \gamma \cdot \lambda \cdot Z_{t-1}(s) + I_{s=S_t} \tag{51}$$

If the above is too aggressive, can just set eligibility trace as the following. In other words, eligibility trace indicates the degree to which each state is eligible for undergoing learning changes.

$$\mathrm{Z_t}(s) = \begin{cases} 1 & \text{if } S_t = s \\ \gamma \cdot \lambda \cdot z_{t-1} & \text{if } S_t \neq s \end{cases} \tag{52}$$

One step TD error is calculated as shown below

$$\delta_t = R_{t+1} + \gamma \cdot V_t\left(s_{t+1}\right) - V_t\left(s_t\right) \tag{53}$$

Global TD error signal triggers proportional updates to all recently visited states

$$V_{t+1}(s) = V_t(s) + \alpha \cdot \delta_t \cdot Z_t(s), \forall s \in \mathbb{S} \tag{54}$$

As an example, when $\lambda = 0$, $Z_t = 1$ at S_t which turns equation 54 in to the standard TD(0) equation. When $\lambda = 1$, credit given to earlier states fall by γ which resembles MC behavior.

Algorithm 3 The function that implements the tabular TD(λ) algorithm with replacing traces. This function must be called after each transition.

function TDLAMBDA(X, R, Y, V, z)

Input: X is the last state, Y is the next state, R is the immediate reward associated with this transition, V is the array storing the current value function estimate, z is the array storing the eligibility traces

$\delta \leftarrow R + \gamma \cdot V[Y] - V[X]$

for all $x \in \mathcal{X}$ **do**

  $z[x] \leftarrow \gamma \cdot \lambda \cdot z[x]$

  if $X = x$ **then**

    $z[x] \leftarrow 1 + z[x]$

  end if

  $V[x] \leftarrow V[x] + \alpha \cdot \delta \cdot z[x]$

end for

return (V, z)

Figure 20: Eligibility Trace

The following shows the Online TD(λ) formula. To change it to SARSA, all V are first replaced with Q. Line 5 is not used. After line 6, choose a' from s' using $\epsilon - greedy$. For Q(λ), need to check if the chosen action a' is optimal. To do so, after line 6 (including the previously added line), choose a^* using $argmax_b(s', b)$. Then, line 11 is changed to only update the trace if $a' = a^*$ otherwise it is set to zero.

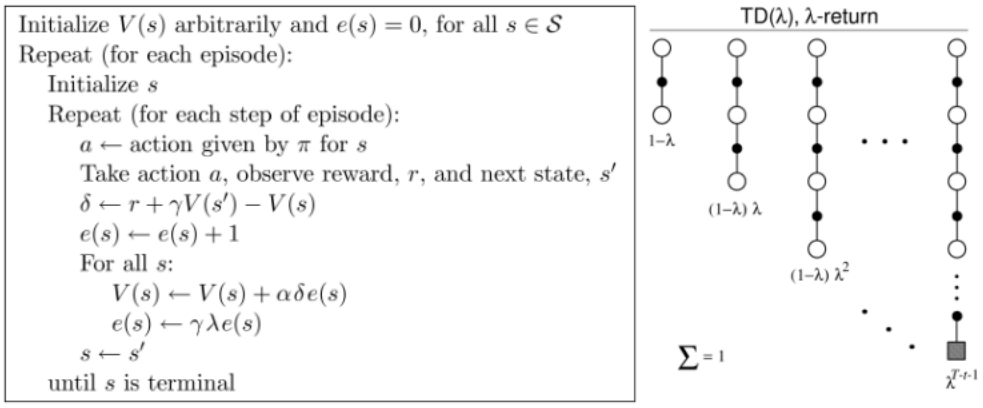

Figure 21: Online TD(λ)

The following shows the results of different algorithms. For TD(λ), the online and offline implementations are similar when λ is small.

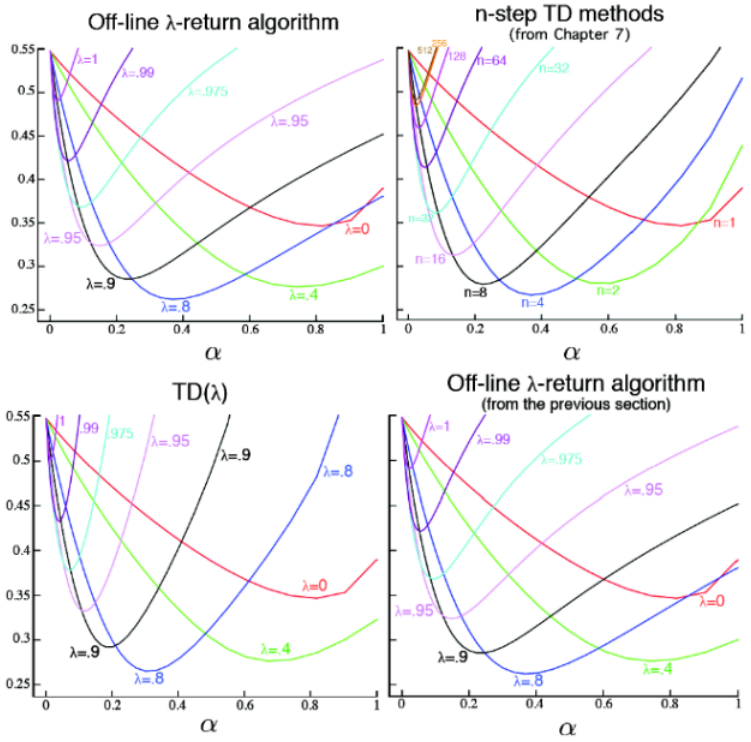

Figure 22: Results

SARSA(λ)
Here, we apply TD(λ) to state action pairs instead of just states; therefore, we need a trace for each state-action pair.

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a) \tag{55}$$

$$\delta_t = r_{t+1} + \gamma Q_t\left(s_{t+1}, a_{t+1}\right) - Q_t\left(s_t, a_t\right) \tag{56}$$

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 & \text{if } s = s_t \text{ and } a = a_t \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise} \end{cases} \tag{57}$$

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all s, a
Repeat (for each episode):
  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)
    $\delta \leftarrow r + \gamma Q(s', a') - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all s, a:
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      $e(s,a) \leftarrow \gamma\lambda e(s,a)$
    $s \leftarrow s'$; $a \leftarrow a'$
  until s is terminal

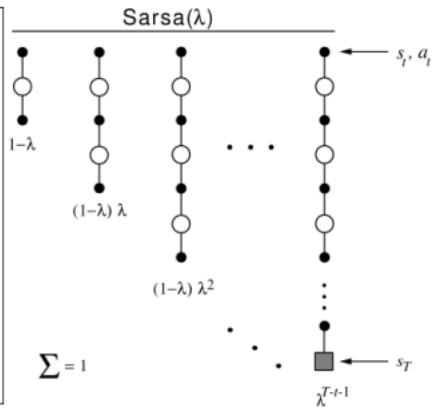

Figure 23: SARSA(λ)

Q(λ)
Q-learning = off-policy = policy learned about need not be the same as the one used to select actions. In learning about the value of the greedy policy, we can use subsequent experience only as long as the greedy policy is being followed. If an exploratory action was chosen, the return for this step does not have any relationship to the greedy policy. Instead of looking ahead to the end of an episode like in TD(λ) and SARSA(λ), Q(λ) only look ahead as far as the next exploratory action.

Eligibility trace is 0 when the action taken is exploratory.

$$e_t(s,a) = \mathcal{I}_{ss_t} \cdot \mathcal{I}_{aa_t} + \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{if } Q_{t-1}\left(s_t, a_t\right) = \max_a Q_{t-1}\left(s_t, a\right) \\ 0 & \text{otherwise} \end{cases} \tag{58}$$

The update algorithm and the TD error is calculated as shown below.

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a) \tag{59}$$

$$\delta_t = r_{t+1} + \gamma \max_{a'} Q_t\left(s_{t+1}, a'\right) - Q_t\left(s_t, a_t\right) \tag{60}$$

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all s, a
Repeat (for each episode):
  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)
    $a^* \leftarrow \arg\max_b Q(s', b)$ (if a' ties for the max, then $a^* \leftarrow a'$)
    $\delta \leftarrow r + \gamma Q(s', a^*) - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all s, a:
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      If $a' = a^*$, then $e(s,a) \leftarrow \gamma\lambda e(s,a)$
        else $e(s,a) \leftarrow 0$
    $s \leftarrow s'$; $a \leftarrow a'$
  until s is terminal

Figure 24: Q(λ)

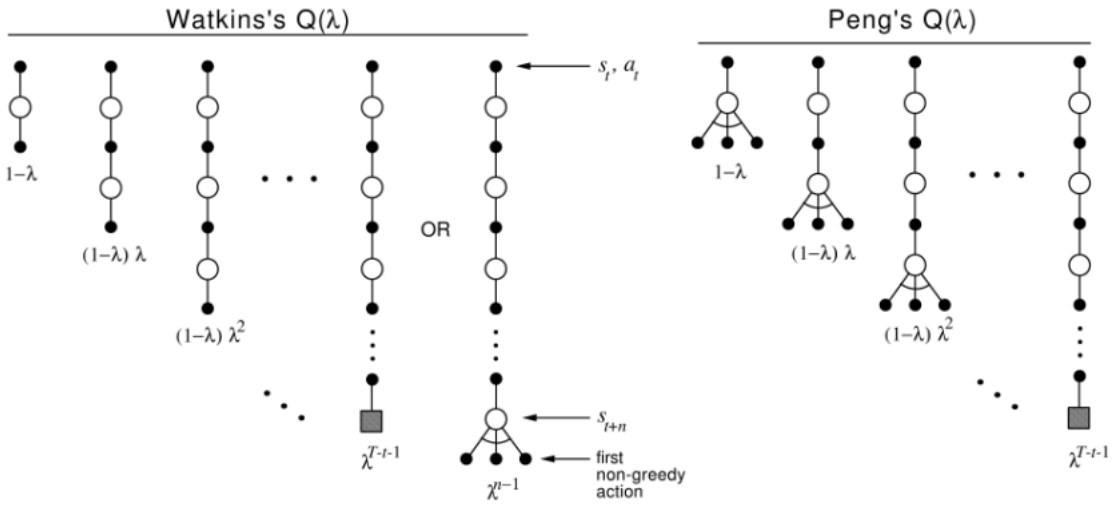

Figure 25: Q(λ) Backup Diagram