Model-Free Prediction

Model-Free MDP

- Many applications can be modeled as a MDP $< S, A, P, R, \gamma >$: Backgammon, Go, Robot locomotion, Helicopter flight, Autonomous driving, Stock prediction, Patient treatment

- For many of these and other problems either:
  - MDP model is unknown but can be sampled or
  - MDP model is known but it is computationally infeasible to use directly, except through sampling

Model-Free Prediction

- So far, we have assumed we have a complete knowledge of model
  - Dynamic Programming

- In many cases, the exact model are not known to us: model-free MDP
  - i.e. **transition probability and rewards not known** $< S, A, \cancel{P, R}, \gamma >$:
- How do we compute value functions and/or policy in model-free environment?
  - requires only experience—sample sequences of states, actions, and rewards from actual or simulated interaction with an environment.

- Model-free prediction: given π, estimate the value function of an unknown MDP
  - the goal is to measure how well it performs
- Model-free control: goal is to find the policy that maximizes the expected total reward from any given state
  - usually works by also solving the prediction problem

Model-Free Prediction

- Since we have no prior knowledge of the environment's dynamics, we can **not** apply dynamic programming anymore

- Model-free prediction: Policy evaluation in model-free MDP
  - Interact with environment following the policy π, and use the data to efficiently compute a good estimate(value function) of the policy

- Methods in model-free prediction
  - Monte-Carlo(MC) Learning
  - Temporal-Difference(TD) Learning

Recap

- Definition of Return, G_t
  - Discounted sum of rewards from time step t to horizon

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots$$

- Definition of State Value Function, $V_\pi(s)$
  - Expected return from starting in state s under policy

$$V_\pi(s) = E_\pi[G_t | s_t = s] = E_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots | s_t = s]$$

- Definition of State-Action Value Function, $Q_\pi(s,a)$
  - Expected return from starting in state s, taking action a and then following policy π

$$\begin{aligned} Q_\pi(s,a) &= E_\pi[G_t | s_t = s, a_t = a] \\ &= E_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots | s_t = s, a_t = a] \end{aligned}$$

Recap: Dynamic Programming for Policy Evaluation

- If model-based, use Iterative Policy Evaluation (dynamic programming) method to compute state value function
  - Initialize $V_0^\pi(s) = 0$ for all s
  - For k=1 until convergence
    - For all s in S

$$V_{k+1}^\pi(s) = \sum_{a \in A} \pi(a|s)\left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_k^\pi(s')\right)$$

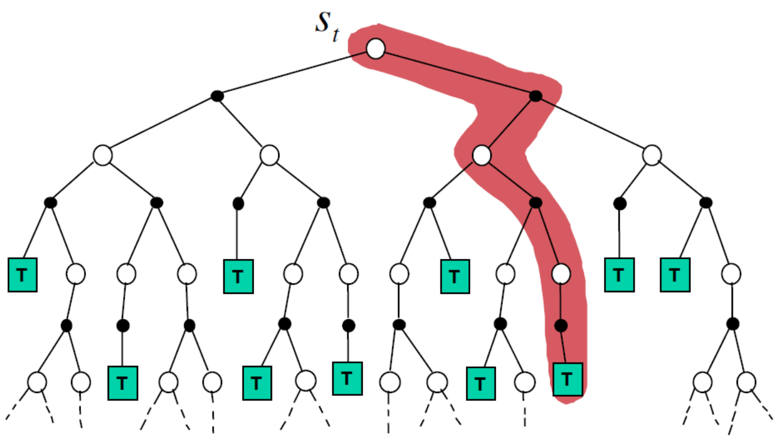
- Since we don't know the values of R_s^a and $P_{ss'}^a$, we can't use DP

Monte-Carlo(MC) RL

- The first method we can use is Monte Carlo method.
- MC is model-free: no knowledge of MDP transitions/rewards
  - Estimating value functions in unknown MDP

- MC methods learn directly from episodes of experience
- MC uses the simplest possible idea:
  - state value = average sample return

- MC learns from complete episodes
- But: still assuming finite MDP problems
- Can only be applied to episodic MDPs
  - Averaging over returns from a complete episode
  - Requires each episode to terminate



https://www.davidsilver.uk/teaching/

Monte Carlo Policy Evaluation

- Goal: estimate $V_\pi(s)$ given episodes generated under policy π
  - $s_1, a_1, r_2, s_2, a_2, r_3, \ldots$ where the actions are sampled from π

- Recall that the return is the total discounted reward:
  - $G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots$ under policy π

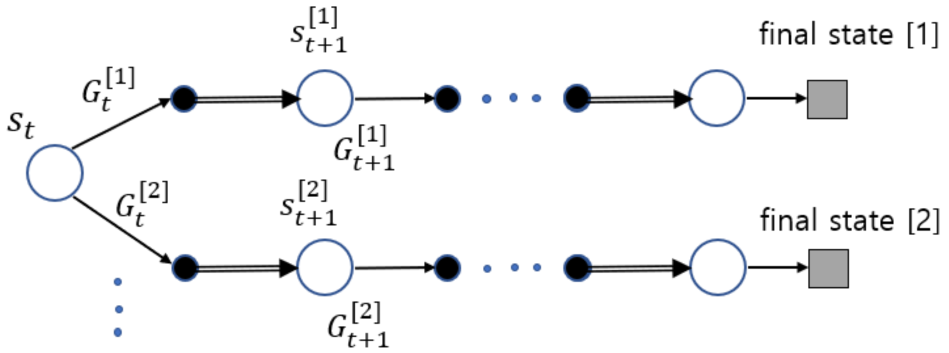
- Recall that the value function is the expected return:

$$V_\pi(s) = E_\pi[G_t | S_t = s]$$

- Monte-Carlo policy evaluation uses empirical mean return
- If trajectories are all finite, sample set of trajectories & average returns

Task Description and Basic Solution

- MC prediction problem statement
  - Estimate state value $v_\pi(s)$ for a given policy π



- MC solution approach:
  - Over episodes $j = 1, \ldots$

$$v_\pi(s) = E_\pi[G_t | S_t = s] = v_\pi(s_t) = \frac{1}{J}\sum_{j=1}^{J} G_t^{[j]} = \frac{1}{J}\sum_{j=1}^{J}\sum_{i=0}^{T_j} \gamma^i r_{t+i+1}^{[j]}$$

  - $r_t^{[i]}$: reward at s_t
  - Above, T_j denotes the terminating time step of each episode j.
  - First-visit MC: Apply update only to the first state visit per episode.
  - Every-visit MC: Apply update each time visiting a certain state per episode (if a state is visited more than one time per episode).

First-Visit Monte Carlo Policy Evaluation

- To evaluate state s,
- the first-visit MC method averages the returns following the **first time** in each episode that the state was visited and the action was selected.

- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $Return(s) \leftarrow Return(s) + G_t$
- Value is estimated by mean return V(s) = $Return$(s)/$N(s)$
- By law of large numbers, $V(s) \rightarrow V_\pi(s)$ as $N(s) \rightarrow \infty$

First-Visit Monte Carlo Policy Evaluation

Algorithm 1: First-Visit MC Prediction

Input: policy π, positive integer *num_episodes*
Output: value function V ($\approx v_\pi$, if *num_episodes* is large enough)
Initialize $N(s) = 0$ for all $s \in \mathcal{S}$
Initialize $\text{Returns}(s) = 0$ for all $s \in \mathcal{S}$
for *episode* $e \leftarrow 1$ ***to*** $e \leftarrow num\_episodes$ **do**
  Generate, using π, an episode $S_0, A_0, R_1, S_1, A_1, R_2 \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  for *time step* $t = T - 1$ ***to*** $t = 0$ *(of the episode e)* **do**
    $G \leftarrow G + R_{t+1}$
    if *state* S_t *is* ***not*** *in the sequence* $S_0, S_1, \ldots, S_{t-1}$ **then**
      $\text{Returns}(S_t) \leftarrow \text{Returns}(S_t) + G_t$
      $N(S_t) \leftarrow N(S_t) + 1$
  end
end
$V(s) \leftarrow \frac{\text{Returns}(s)}{N(s)}$ for all $s \in \mathcal{S}$
return V

Every-Visit Monte Carlo Policy Evaluation

- To evaluate state s,
- the every-visit MC method estimates the value of a state–action pair as the average of the returns that have followed **all** the visits to it.

- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $Return(s) \leftarrow Return(s) + G_t$
- Value is estimated by mean return V(s) = $Return$(s)/$N(s)$
- Again, $V(s) \rightarrow V_\pi(s)$ as $N(s) \rightarrow \infty$

- V(s) every-visit MC estimator is a *biased* estimator of $V_\pi(s)$
- But *consistent* estimator and often has better MSE

Every-Visit Monte Carlo Policy Evaluation

Algorithm 2: Every-Visit MC Prediction

Input: policy π, positive integer $num\_episodes$
Output: value function V ($\approx v_\pi$, if $num\_episodes$ is large enough)
Initialize $N(s) = 0$ for all $s \in \mathcal{S}$
Initialize Returns$(s) = 0$ for all $s \in \mathcal{S}$
for $episode\ e \leftarrow 1$ ***to*** $e \leftarrow num\_episodes$ **do**
  Generate, using π, an episode $S_0, A_0, R_1, S_1, A_1, R_2 \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  for $time\ step\ t = T-1$ ***to*** $t = 0$ *(of the episode e)* **do**
    $G \leftarrow G + R_{t+1}$
    $\text{Returns}(S_t) \leftarrow \text{Returns}(S_t) + G_t$
    $N(S_t) \leftarrow N(S_t) + 1$
  end
end
$V(s) \leftarrow \frac{\text{Returns}(s)}{N(s)}$ for all $s \in \mathcal{S}$
return V

Incremental Monte-Carlo Updates

- V(s) = $Return$(s)/$N(s)$
  - $N(S_t)$, $V(S_t)$ values can be computed incrementally
  - After each episode, update estimate of $V(S_t)$

- Update $V(s)$ incrementally after episode $S_1, A_1, R_2, \ldots S_T$
- For each state S_t with return G_t

$$N(S_t) \leftarrow N(S_t) + 1$$
$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

$$\begin{aligned}\mu_k &= \frac{1}{k}\sum_{j=1}^{k} x_j \\ &= \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right) \\ &= \frac{1}{k}(x_k + (k-1)\mu_{k-1}) \\ &= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})\end{aligned}$$

- In non-stationary problems, it can be useful to track a running mean,
  - i.e. forget old episodes due to non-stationary.

$$V(S_t) \leftarrow V(S_t) + \alpha\big(G_t - V(S_t)\big)$$

Temporal-Difference(TD) Learning

- “If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning." (Sutton and Barto)

- TD is model-free: no knowledge of MDP transitions/rewards
- TD methods learn directly from episodes of experience

- TD learns from incomplete episodes by bootstrapping
- TD updates a guess towards a guess
- Can be used in episodic or infinite-horizon non-episodic settings
- Immediately updates estimate of V after each (s,a,r,s’) tuple

MC and TD

- Goal: learn $V(S_t)$ from experience under policy π
- $G_t = R_{t+1} + \gamma R_{t+2} + \cdots$ in MDP M under policy π

- Incremental first/every-visit Monte-Carlo: update estimate using 1 sample of return (for the current t-th episode)
  - Update value $V(S_t)$ toward *actual* return G_t

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

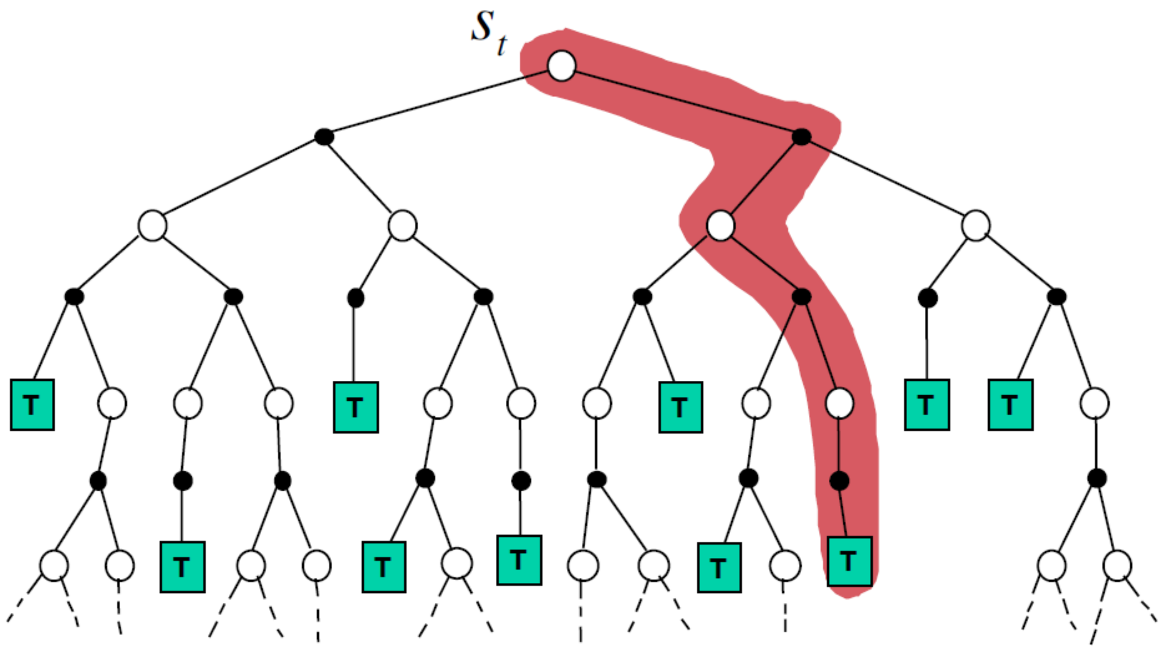
- Simplest Temporal-Difference learning algorithm: TD(0)
  - Update value $V(S_t)$ toward *estimated* return $R_{t+1} + \gamma V(S_{t+1})$

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

  - $R_{t+1} + \gamma V(S_{t+1})$ is called the *TD target*
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*

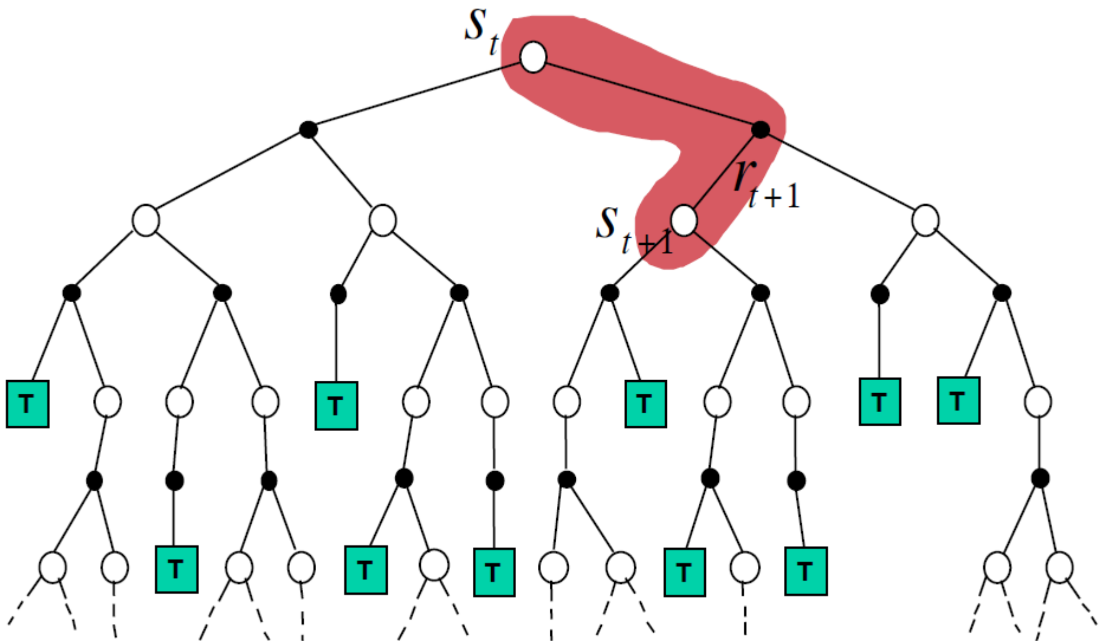
MC Backup

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$



TD Backup

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



https://www.davidsilver.uk/teaching/

TD(0) Learning Algorithm

- Input α
- Initialize $V_\pi(s) = 0$ for all s
- Loop
  - Sample tuple $(s_t, a_t, r_{t+1}, s_{t+1})$ based on π
  - $V_\pi(s_t) = V_\pi(s_t) + \alpha([r_{t+1} + \gamma V_\pi(S_{t+1})] - V_\pi(S_t))$

- Can immediately update value estimate after (s, a, r, s') tuple
- Don't need episodic setting

Pros and Cons of MC vs. TD

- TD can learn before knowing the final outcome
  - TD can learn online after every step
  - MC must wait until end of episode before return is known
  - TD can learn from incomplete sequences
  - MC can only learn from complete sequences
  - TD works in non-terminating environments
  - MC only works for episodic terminating environments
    - Sometime, terminal state is too costly to reach

Bias/Variance Tradeoff

- Return $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$ is *unbiased* estimate of $v_\pi(s_t)$

- TD target $R_{t+1} + \gamma v(S_{t+1})$ is *biased* estimate of $v_\pi(s_t)$
  - *biased*: expected value of TD target is not equal to true value
  - Over time, as your estimate of $v(S_{t+1})$ improves, the bias subsides

- TD target is much lower variance than the return:
  - Return depends on many random actions, transitions, rewards
  - TD target depends on one random action, transition, reward

Pros and Cons of MC vs. TD (2)

- MC has high variance, zero bias
  - Good convergence properties (even with function approximation)
  - Not very sensitive to initial value
  - Very simple to understand and use

- TD has low variance, some bias
  - Usually more efficient than MC
  - TD(0) converges to $v(s)$ (but not always with function approximation)
  - More sensitive to initial value

Batch MC and TD

- MC and TD converge: $V(s) \rightarrow v_\pi(s)$ as experience $\rightarrow \infty$
- But what about **batch** solution for finite experience?

$$\left(s_1^{[1]}, a_1^{[1]}, r_2^{[1]}, \dots, s_{T_1}^{[1]}\right), \cdots, (s_1^{[K]}, a_1^{[K]}, r_2^{[K]}, \dots, s_{T_K}^{[K]})$$

  - $s_t^{[i]}$: state s_t in episode i
  - e.g. Repeatedly sample episode $k \in [1, k]$
  - Apply MC or TD(0) to episode k

Certainty Equivalence

- MC in batch mode converges to minimum mean-squared error(MSE)
  - Minimize loss with respect to observed returns
  - ($G_t^{[k]}$: return of k-th episode, $v(s_t^{[k]})$: state value of k-th timestep)

$$\sum_{k=1}^{K}\sum_{t=1}^{T_k}\left(G_t^{[k]} - v(s_t^{[k]})\right)^2$$

- TD(0) converges to converges to DP policy V_π MDP with Maximum Likelihood model estimates
  - Solution to the MDP $< S, A, \hat{P}, \hat{R}, \gamma >$ that best fits the data

$$\hat{P}_{ss'}^{a} = \frac{1}{n(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k} 1(s_t^{[k]} = s, a_t^{[k]} = a, s_{t+1}^{[k]} = s')$$

$$\hat{R}_{s}^{a} = \frac{1}{n(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k} 1\left(s_t^{[k]} = s, a_t^{[k]} = a\right) r_t^{[k]}$$

AB Example

R. Sutton and A. Barto, Reinforcement Learning: An Introduction, 2018

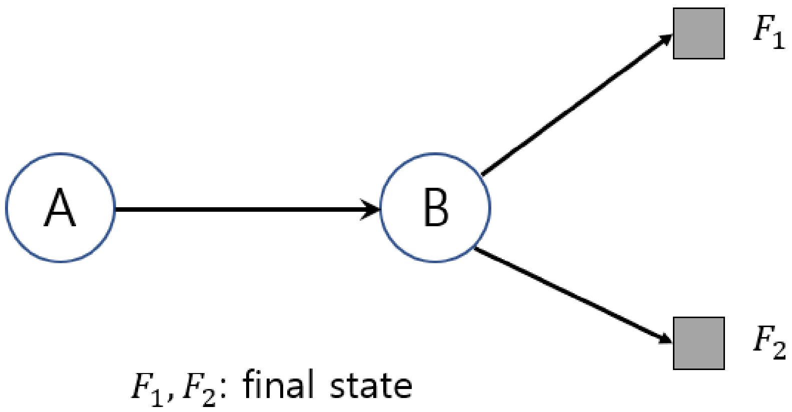
- Two states A,B; no discounting; 8 episodes of experience
- {A,0,B,0}, {B,1}, {B,1}, {B,1}, {B,1}, {B,1}, {B,1}, {B,0}
- What is V(A), V(B)?

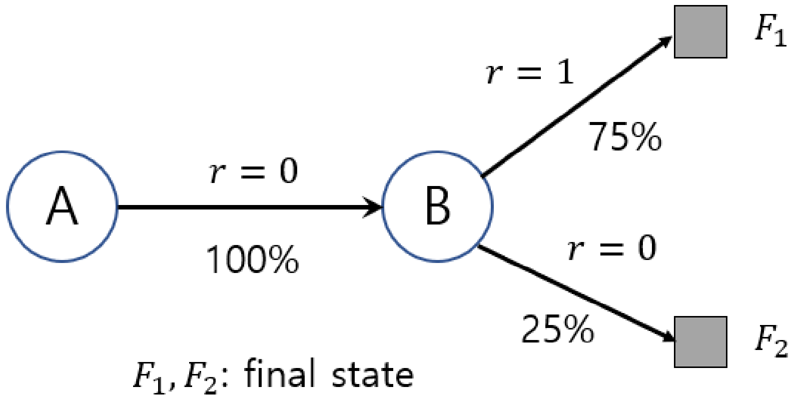
- Given, {A,0,B,0}, {B,1}, {B,1}, {B,1}, {B,1}, {B,1}, {B,1}, {B,0}
- MC
  - In the AB example, V(A) = 0. There is only one episode which include A.
  - There are 8 episode which include B. V(B)=6/8=3/4

$$\text{Min } \left(0 - v(B)\right)^2 + \left(1 - v(B)\right)^2 + \left(1 - v(B)\right)^2 + \cdots + \left(1 - v(B)\right)^2$$

AB Example

- TD(0)
  - In the AB example, B is final state and compute average
    - Assume run in batch mode, and α=1 & γ=1,
    - MRP Bellman Equation: $v(s) = R_s + \gamma \sum_{s' \in \boldsymbol{S}} P_{ss'} v(s')$
    - Therefore,
    - $v(B) = 0 + (P_{BF_1} v(F_1) + P_{BF_2} v(F_2)) = 0.75 * 1 + 0.25 * 0 = 0.75$
  - Since V(B)=3/4, $v(A) = 0 + \sum_{s' \in \boldsymbol{S}} P_{AB} v(B) = 0 + 1 * v(B) = 0.75$

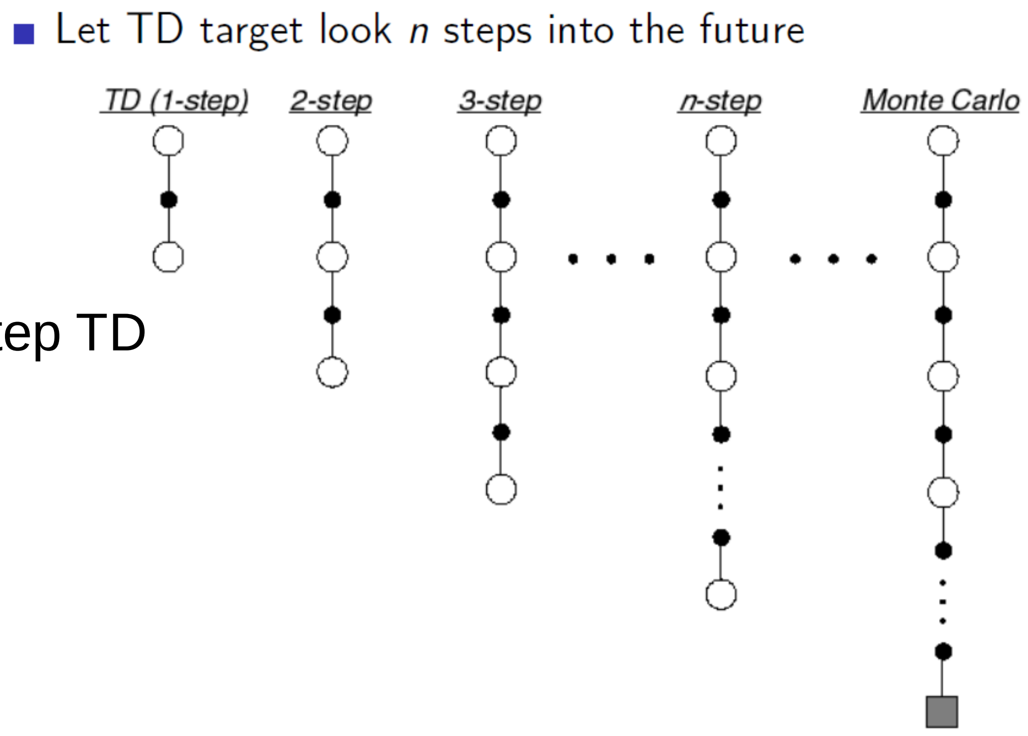




TD(n) Learning

- However, the effect of an action often occurs after several steps.
- Using data from only one step will make learning slow.
- n-step return

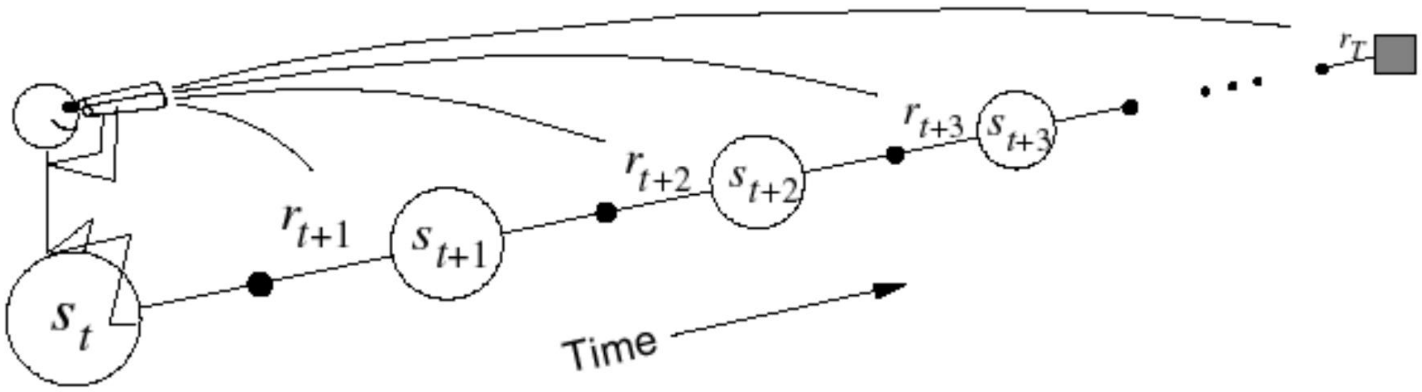
$$G_t^{(1)} = R_{t+1} + \gamma v(s_{t+1}) \text{ : (TD(0))}$$
$$G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 v(s_{t+2})$$
$$\dots$$
$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n v(s_{t+n}) \text{ : n-step TD}$$
$$\dots$$
$$G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_{t+T} \text{ : MC}$$



- TD(n) learning: uses $G_t^{(n)}$ instead of $G_t^{(1)}$

$$v(s_t) \leftarrow v(s_t) + \alpha\left(G_t^{(n)} - v(s_t)\right)$$

TD(λ) Learning



R. Sutton and A. Barto, Reinforcement Learning: An Introduction, 2018

- Rather than using a return for a specific value of n, think about how to compute a return for all possible values of n.
- Then weight each of them and use the average of them as the final return.
- The advantage: we don't need to determine the value of n separately.

- Forward-view looks into the future to compute G_t^{λ}
- Like MC, can only be computed from complete episodes
- Have to wait until episode is finished

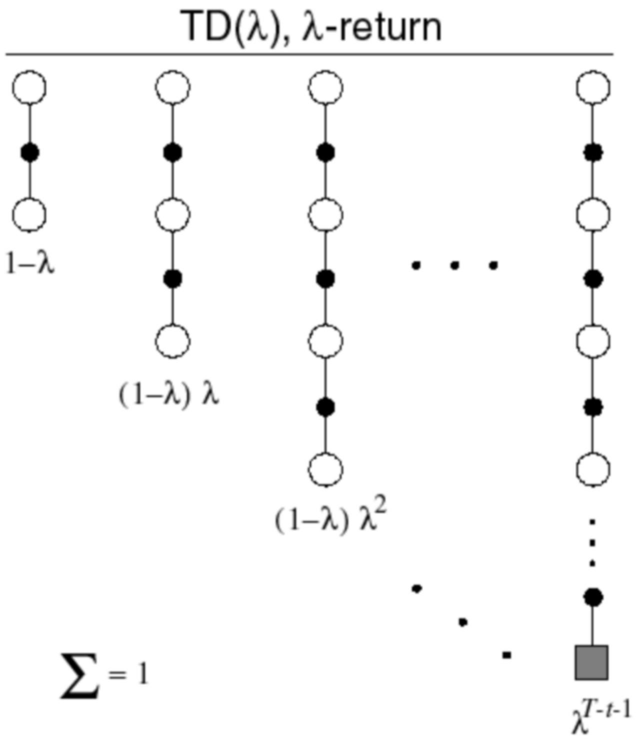
TD(λ) Learning

- Assume λ^{n-1} is the weight of n-step return: $\lambda^{n-1} G_t^{(n)}$
- Use normalized weight $(1-\lambda)\lambda^{n-1} G_t^{(n)}$ since $\sum_{n=1}^{\infty} \lambda^{n-1} = \sum_{n=0}^{\infty} \lambda^{n} = \frac{1}{1-\lambda}$
- Now TD(λ) uses (called λ return)

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

$$v(s_t) \leftarrow v(s_t) + \alpha \left(G_t^{\lambda} - v(s_t) \right)$$

- Forward TD(λ)



Truncated λ Return

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

- Once agent reaches final state, return value remains constant. Modify the formula

$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1}\, G_t^{(n)} + (1-\lambda)\sum_{n=T-t}^{\infty} \lambda^{n-1}\, G_t$$

- Since $(1-\lambda)\sum_{n=T-t}^{\infty} \lambda^{n-1}\, G_t = \lambda^{T-t-1} G_t$

$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1}\, G_t^{(n)} + \lambda^{T-t-1} G_t$ (called Truncated $\boldsymbol{\lambda}$ return)

- When $\boldsymbol{\lambda}$=0, it becomes TD(0)

$$G_t^{\lambda} = (1-0)\sum_{n=1}^{T-t-1} 0^{n-1}\, G_t^{(n)} + 0^{T-t-1} G_t = G_t^{(1)} = R_{t+1} + \gamma v(s_{t+1})$$

- When $\boldsymbol{\lambda}$=1, it becomes TD(1) (Monte Carlo method)

$$G_t^{\lambda} = (1-1)\sum_{n=1}^{T-t-1} 1^{n-1}\, G_t^{(n)} + 1^{T-t-1} G_t = G_t$$

Backward TD(λ)

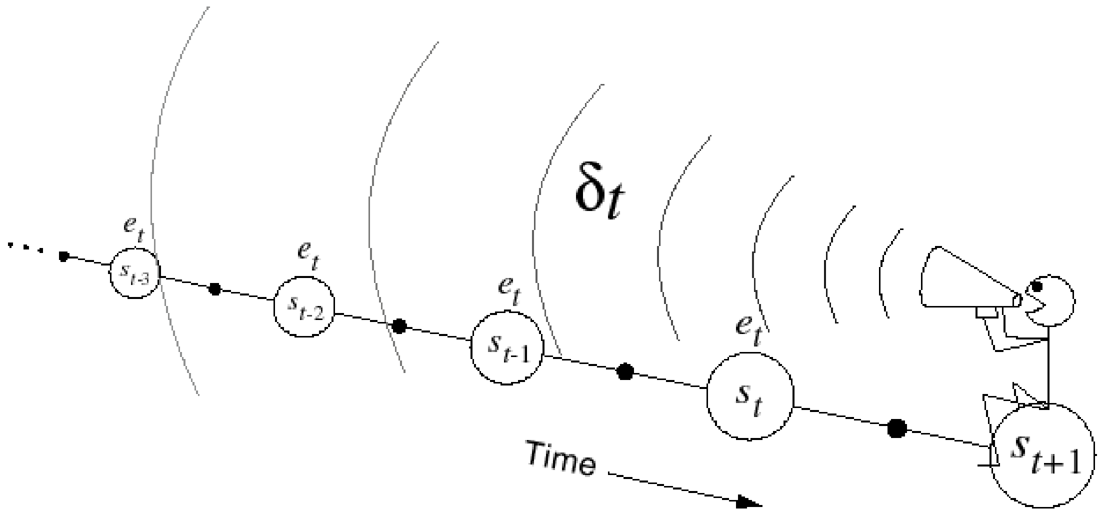
- In forward TD(λ), the agent has to wait (or reach the end state) to perform n steps to perform the update the return values in the future time step are used.
  - the return values in the future time step are used.
  - Similar to MC
  - forward TD(λ) is combination of TD learning and MC
  - offline learning

- Backward TD(λ) can remember what happened in the past step and update values for past states using data in the current step.
- Backward TD(λ) can be learned online and can be updated immediately whenever an agent's behavior is performed.
- General idea:
  - Use λ returns looking into the past.
  - Therefore, an eligibility trace $e_t(s)$ denoting the importance of past events to the current state update is introduced.

Eligibility Trace

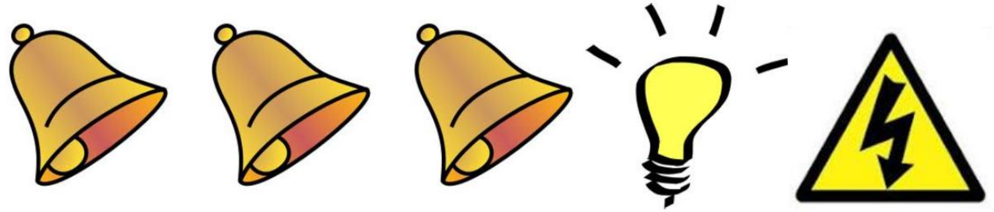
- In order to perform Backward TD(λ) learning, it is necessary to measure how much a specific state in the past has affected the current state.
- It is impossible to predict whether the current state will occur in the future from the past state.

- For this purpose, the method devised (measuring how much the past state has affected the current state) is Eligibility Trace.



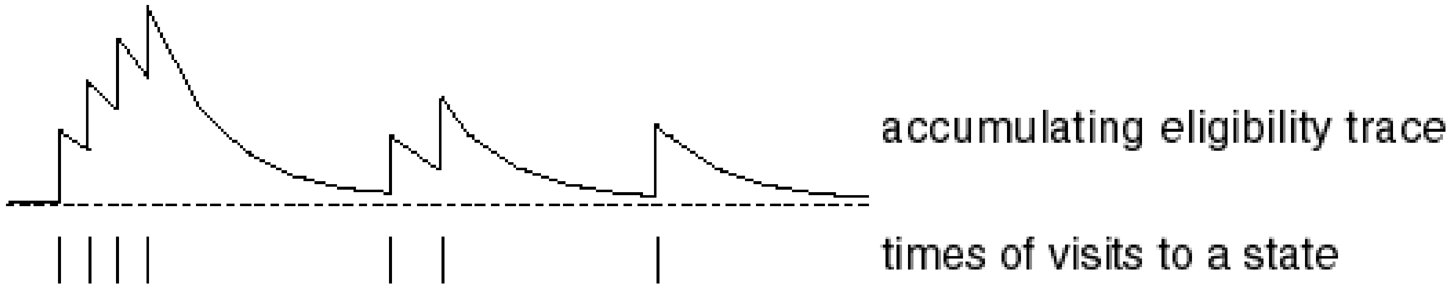
R. Sutton and A. Barto, Reinforcement Learning: An Introduction, 2018

Eligibility Trace



- Credit assignment problem: did bell or light cause shock?
  - Frequency heuristic: assign credit to most frequent states
  - Recency heuristic: assign credit to most recent states
- Eligibility traces combine both heuristics

$$e_t(s) = \begin{cases} \lambda\gamma e_{t-1}(s), & \text{if } s_t \neq s \\ \lambda\gamma e_{t-1}(s) + 1, & \text{if } s_t = s \end{cases}$$



R. Sutton and A. Barto, Reinforcement Learning: An Introduction, 2018

Eligibility Trace

- Keep an eligibility trace for every state s
- Update value V(s) for every state s
- In proportion to TD-error δ_t and eligibility trace $e_t(s)$

$$\delta_t = r_{t+1} + \gamma v(s_{t+1}) - v(s_t)$$
$$v(s) \leftarrow v(s) + \alpha\delta_t e_t(s), \forall s$$

- If $\lambda = 0$, TD(0)

$$e_t(s) = \begin{cases} 0, & \text{if } s_t \neq s \\ 1, & \text{if } s_t = s \end{cases}$$
$$\alpha\delta_t e_t(s) = \begin{cases} 0, & \text{if } s_t \neq s \\ \alpha \cdot \delta_t \cdot 1, & \text{if } s_t = s \end{cases}$$
$$v(s_t) \leftarrow v(s_t) + \alpha\delta_t$$

- If $\lambda = 1$, MC (TD(1))

Eligibility Trace

Initialize $V(s)$ arbitrarily and $e(s) = 0$, for all $s \in \mathcal{S}$
Repeat (for each episode):
    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'
        $\delta \leftarrow r + \gamma V(s') - V(s)$
        $e(s) \leftarrow e(s) + 1$
        For all s:
            $V(s) \leftarrow V(s) + \alpha \delta e(s)$
            $e(s) \leftarrow \gamma \lambda e(s)$
        $s \leftarrow s'$
    until s is terminal

Replacing Trace

- Eligibility trace value keeps growing

$$e_t(s) = \begin{cases} \lambda\gamma e_{t-1}(s), & \text{if } s_t \neq s \\ 1, & \text{if } s_t = s \end{cases}$$

- The value is normalized [0,1]