BASIC ALGORITHMS

Q- and TD-learning

Diego Klabjan
Professor, Industrial Engineering and Management Sciences

Northwestern | McCORMICK SCHOOL OF ENGINEERING

Outline

- Q-learning
  - Variants
  - Convergence
- TD-learning
  - Monte-Carlo sampling
  - TD(λ) learning

Q-LEARNING

Q Factor

$$Q(s,a) = r(s,a) + \gamma E_{\bar{s}\sim p(\bar{s}|s,a)} V(\bar{s})$$

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \left[r(s,a) + E_{\bar{s}\sim p(\bar{s}|s,a)} V(\bar{s}) \right]$$

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a)$$

- Q is the evaluation function the agent learns
  - From Q we can get V
  - From V we can get Q

$$V(s) = \max_{a'} Q(s,a')$$

Q-learning

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a')$$

- $\hat{Q}$ denotes learner's current approximation to Q
- At state s and action a

$$\hat{Q}(s, a) \leftarrow r(s, a) + \gamma \max_{a'} \hat{Q}(s', a')$$

  - s' the state resulting from applying action a in state s

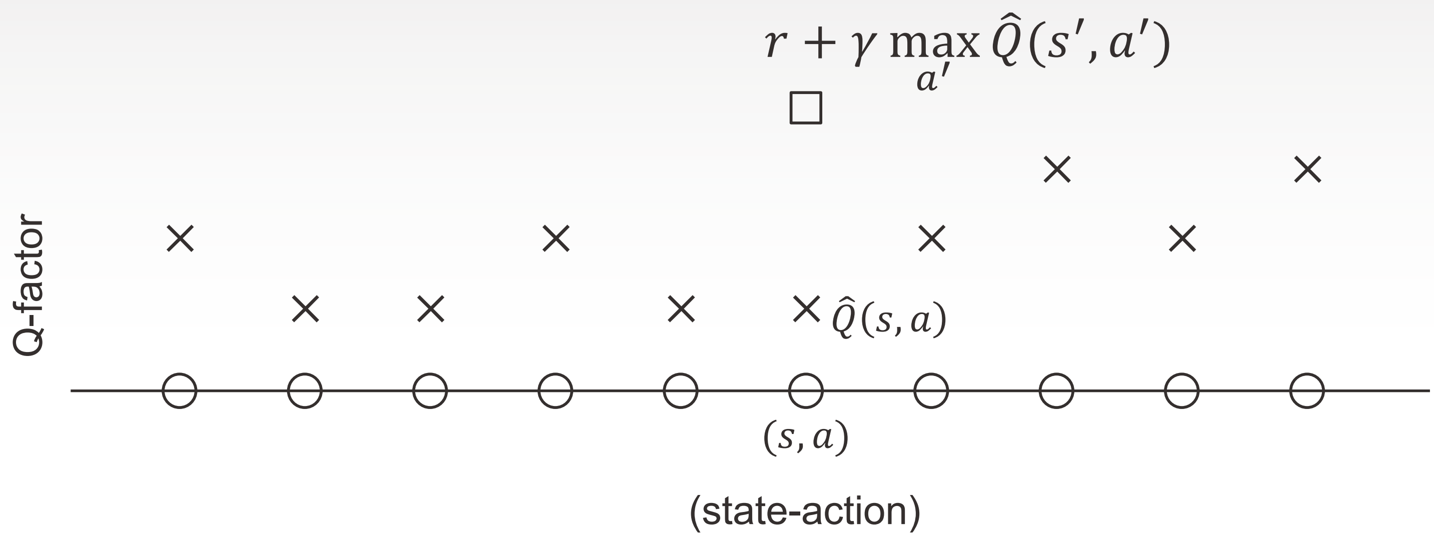
    $$s' \sim p(s' | s, a)$$

    - Take a single sample
  - Repeat

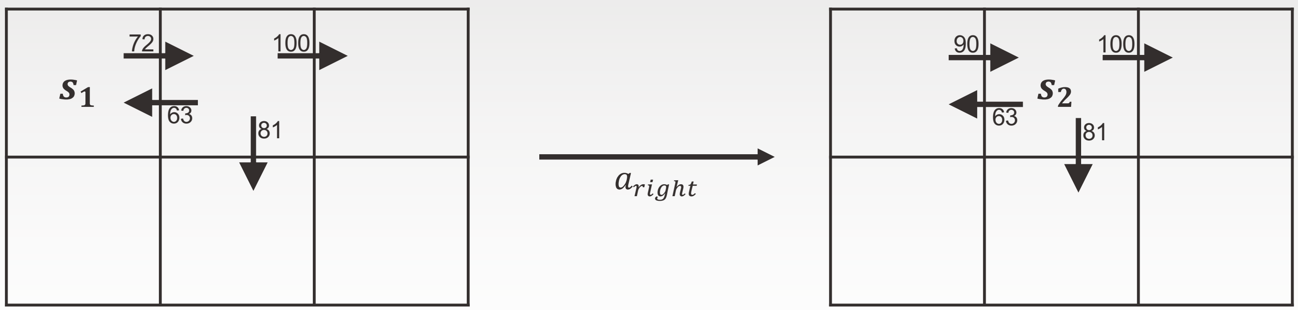
Q-learning for Deterministic Model

- For each s, a initialize table entry $\hat{Q}(s, a) \leftarrow 0$
- Observe current state s
- Loop forever
  - Select an action a and execute it
  - Receive immediate reward r
  - Observe the new state s'
  - Update the table entry for $\hat{Q}(s, a)$ as follows
  $$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$
  - $s \leftarrow s'$

Q-learning



Updating $\hat{Q}$



current state: s_1 next state: s_2

$$\hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a')$$
$$\leftarrow 0 + 0.9 \max\{63,81,100\}$$
$$\leftarrow 90$$

- Reward non-negative

$$(\forall s, a, n)\ \hat{Q}_{n+1}(s,a) \geq \hat{Q}_n(s,a)$$
$$(\forall s, a, n)\ 0 \leq \hat{Q}_n(s,a) \leq Q(s,a)$$

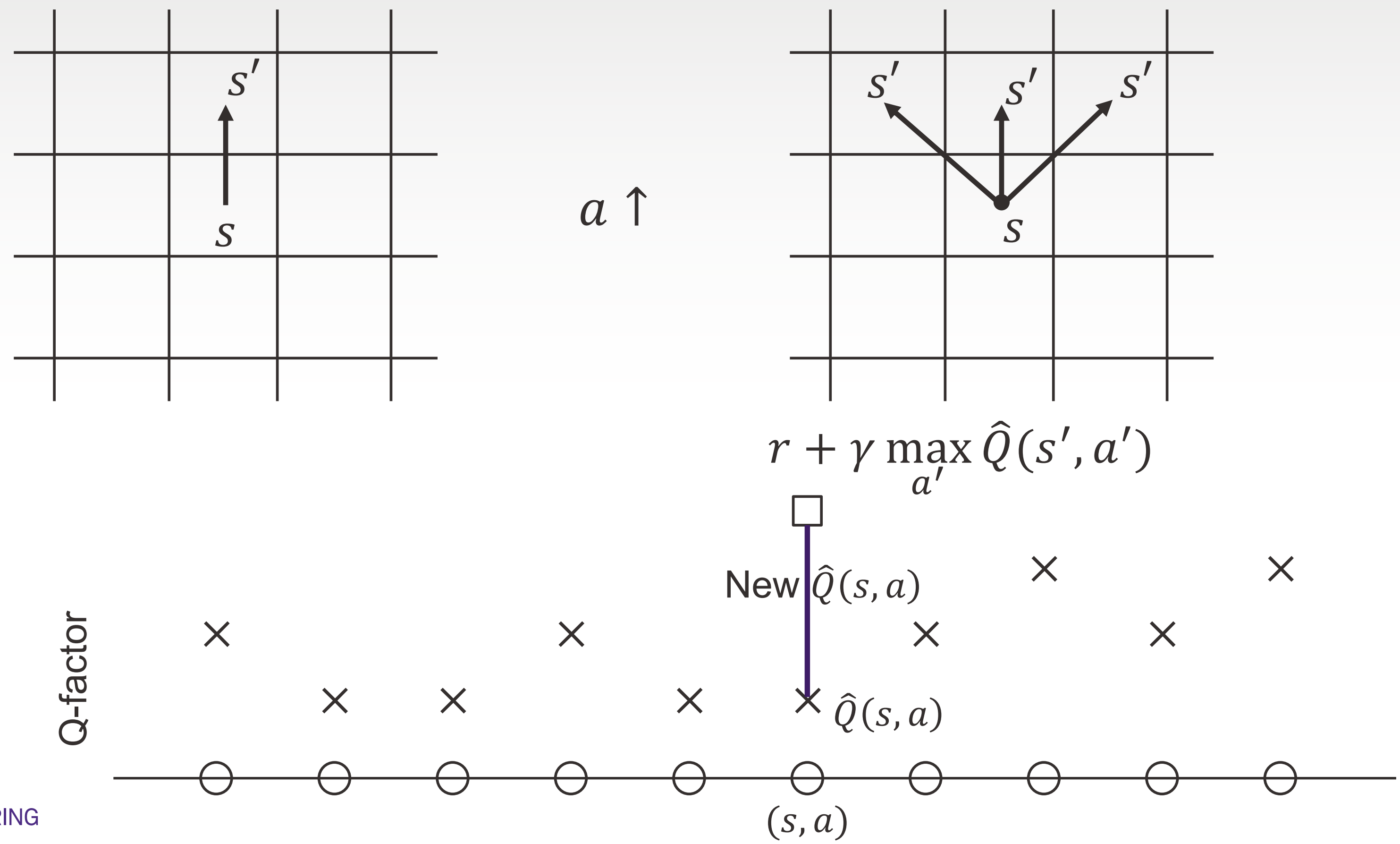
- New Q better
- Can be shown that $\hat{Q}$ converges to Q

Nondeterministic Case

- Alter training rule to
$$\hat{Q}_n(s,a) \leftarrow (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_n \left[r + \max_{a'} \hat{Q}_{n-1}(s',a')\right]$$
- $s' \sim p(s'|s,a)$
- Learning rule
  - $\alpha_n = \frac{1}{1+visits_n(s,a)}$
  - Convergence of $\hat{Q}$ to Q still holds under mild assumptions

$$\hat{Q}_n(s,a) \leftarrow \hat{Q}_{n-1}(s,a) + \alpha_n \left[r + \max_{a'} \hat{Q}_{n-1}(s',a') - \hat{Q}_{n-1}(s,a)\right]$$

Q-learning



s'
s
$a \uparrow$
s'
s'
s'
s
$r + \gamma \max_{a'} \hat{Q}(s', a')$
New $\hat{Q}(s, a)$
Q-factor
$\hat{Q}(s, a)$
(s, a)

Algorithm

- Loop
  - Sample state s
  - $a = argmax_{\bar{a}} \hat{Q}(s, \bar{a})$
  - Sample $s' \sim p(s'|s, a)$
  - $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha_n \left[r + \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a) \right]$
- Can sample more than one s' and then average
- Note

$$\sum_i \max_{a'} \hat{Q}(s'_i, a') \neq \max_{a'} \sum_i \hat{Q}(s'_i, a')$$

- $\sum_i \max_{a'} \hat{Q}(s'_i, a')$ should be used

Generalization: True Q-learning

- Loop
  - Sample state s
  - Select any action a
  - Sample $s' \sim p(s'|s,a)$
  - $\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha_n \left[r + \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a) \right]$
- Two policies
  - The selection one behind step 2
  - The ‘optimal’ policy behind $\hat{Q}$

SARSA

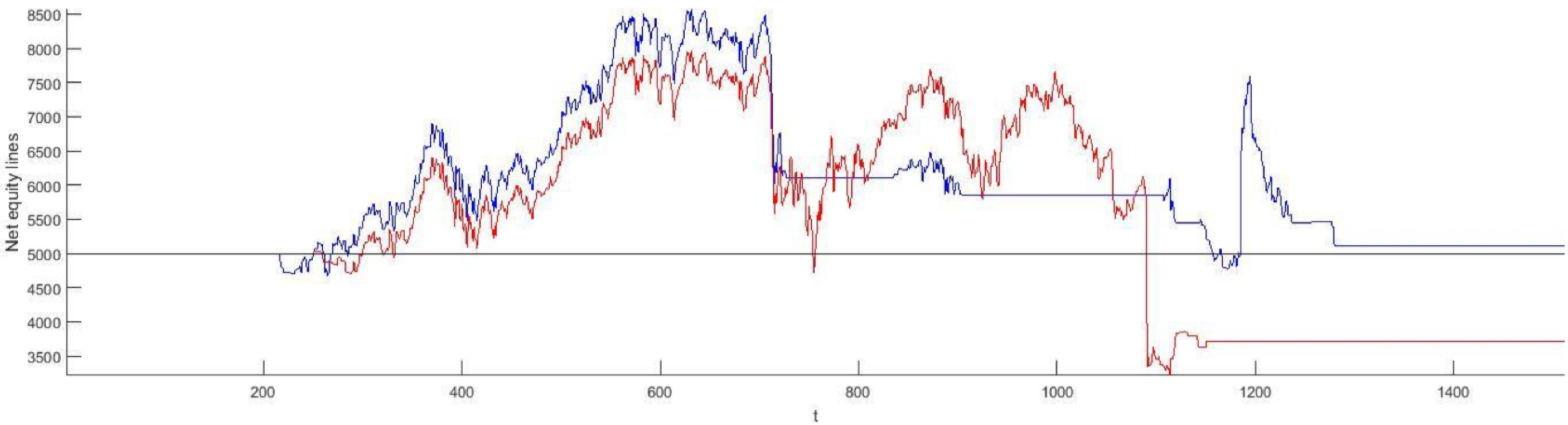
- Estimate value for the current action executed
- Loop
  - Execute action a at state s
  - Get reward r
  - Sample $s' \sim p(s'|s, a)$
  - $a' = \max_a \hat{Q}(s', a)$
  - $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha_n \left[r + \hat{Q}(s', a') - \hat{Q}(s, a) \right]$
  - $s = s', a = a'$
- Just one single policy
  - Agent follows ‘optimal’ policy

Off-policy vs On-policy

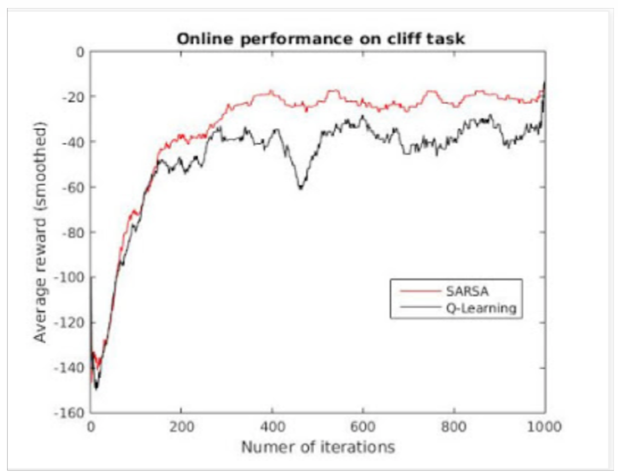
- On-policy algorithm
  - Learn the policy being executed by the agent
- Off-policy algorithm
  - Evaluate a policy from samples generated by a different policy
    - Target policy
  - Learn policy independent of policy taken by agent
    - Behavior policy
- Q-learning is off-policy
- SARSA is on-policy

SARSA vs Q-learning

- SARSA useful when you want to optimize the value of an agent that is exploring
  - Learn near optimal while exploring
  - More conservative – slower convergence
- Do offline learning, then use the policy in an agent that does not explore
  - Q-learning more appropriate
    - Directly learns optimal policy
- Practical experiments
  - Q-values are lower in SARSA than in Q-learning
    - Does not mean that SARSA superior
  - Q-learning higher per sample variance



http://tinyurl.com/y984s42k



https://sridhartee.blogspot.com/2016/09/qlearning-and-sarsa-cliff-task.html

Value Iteration for Q-factor

- Deterministic case
- Use Bellman equation as an iterative update

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a')|s,a\right]$$

  - Compute this for each state-action pair
- Q_i converge to Q^* as $i \to \infty$
- In practice cannot loop through all state-action pairs
  - Functional approximations must be considered

TD-LEARNING

Monte-Carlo Policy Evaluation

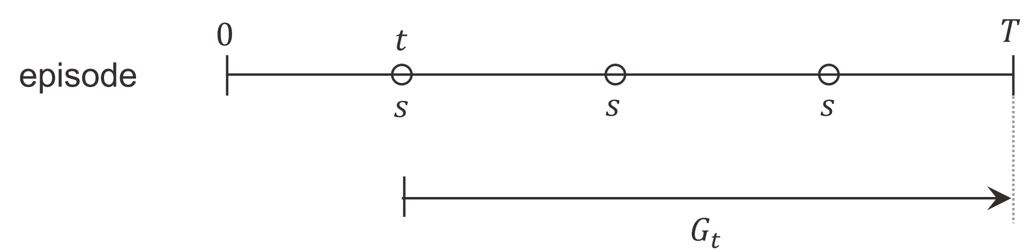
- Learn directly from episodes of experience
- Model-free
  - No knowledge of functions for transitions/rewards
- Learn from complete episodes
- Idea
  - Value = mean return
- Caveat
  - Applicable only to finite episodes
  - Cannot handle infinite time horizon settings

Basics

- Goal
  - Learn V^{π} from episodes of experience under policy π
  - $s_0, a_0, r_0, \dots, s_T \sim \pi$
- Total discounted reward
  - $G_t = r_t + \gamma r_{t+1} + \ \dots + \gamma^{T-1} r_T$
- Value of policy
  - $V^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$
- Monte-Carlo policy evaluation uses empirical mean return instead of expected return

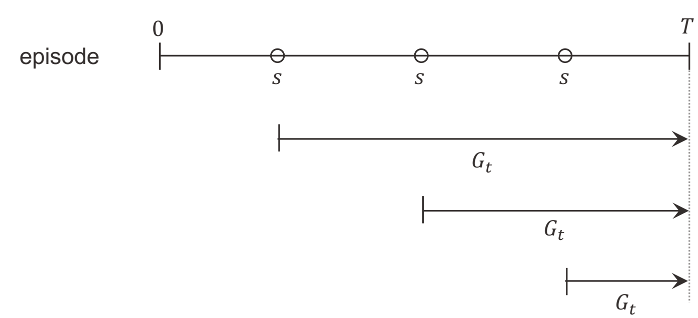
First-Visit Monte-Carlo Policy Evaluation

- To evaluate state s
  - Loop
    - Generate an episode based on π
    - The first time-step t that state s is visited in the episode
      - Increment counter $N(s) \leftarrow N(s) + 1$
      - Increment total return $S(s) \leftarrow S(s) + G_t$
  - Value is estimated by mean return $V(s) = S(s)/N(s)$
- Law of large numbers
  - $V(s) \rightarrow V^{\pi}(s)$ as $N(s) \rightarrow \infty$



Every-Visit Monte-Carlo Policy Evaluation

- To evaluate state s
  - Loop
    - Generate an episode based on π
    - Every time-step t that state s is visited in the episode
      - Increment counter $N(s) \leftarrow N(s) + 1$
      - Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- Law of large numbers
  - $V(s) \rightarrow \mathrm{V}^{\pi}(s)$ as $N(s) \rightarrow \infty$



Incremental Mean

- Mean $\mu_1, \mu_2, \ldots$ of sequence $x_1, x_2, \ldots$

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

Incremental Monte-Carlo Updates

- Update $V(s)$ incrementally after episode $s_0, a_0, r_0, \ldots, s_T$
- For each state s_t with return G_t

$$N(s_t) \leftarrow N(s_t) + 1$$

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

- Non-stationary problems
  - Forget old episodes

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

Temporal-Difference Learning

- Learn directly from episodes of experience
- Model-free
  - No knowledge of functions for transitions/rewards
- Learns from incomplete episodes
- Updates a guess towards a guess

MC and TD

- Goal
  - Learn V^{π} online from experience under policy π
- Incremental every-visit Monte-Carlo
  - 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 $r_{t+1} + \gamma V(s_{t+1})$
  - $V(s_t) \leftarrow V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$
  - TD target: $r_t + \gamma V(s_{t+1})$
  - TD error: $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$

TD Algorithm

- Loop
  - Generate an episode based on π: $s_0, a_0, r_0, \ldots, s_T$
  - For t = 0 to T
    - $V(s_t) \leftarrow V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$
- Recall from value function to policy
  - $\pi(s) = argmax_a r(s,a) + E_{s' \sim p(s'|s,a)} V(s')$
  - This holds for MC and TD

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 without the final outcome
  - TD can learn from incomplete sequences
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments

Bias/Variance Trade-Off

- Return $G_t = r_t + \gamma r_{t+1} + \ \ldots + \gamma^{T-1} r_T$ unbiased estimate of $\mathrm{V}^{\pi}(s_t)$
- True TD target $r_t + \gamma \mathrm{V}^{\pi}(s_{t+1})$ unbiased estimate of $\mathrm{V}^{\pi}(s_t)$
- TD target $\mathrm{r}_t + \gamma V(s_{t+1})$ biased estimate of $\mathrm{V}^{\pi}(s_t)$
- TD target much lower variance than the return
  - Return depends on many random actions, transitions, rewards
  - TD target depends on one random action, transition, reward

Advantages and Disadvantages of MC vs. TD

- MC has high variance, zero bias
  - Good convergence properties
  - Not very sensitive to initial value
  - Very simple to understand and use
- TD has low variance, some bias
  - Usually more efficient than MC
  - TD converges to$V^{\pi}(s)$
  - More sensitive to initial value

Batch MC and TD

- MC and TD converge
  - $V(s) \rightarrow V^{\pi}(s)$ as experience $\rightarrow \infty$
- Batch solution for finite experience?

$$s_0^1, a_0^1, r_0^1, \ldots, s_{T_1}^1$$

$$\vdots$$

$$s_0^K, a_0^K, r_0^K, \ldots, s_{T_K}^K$$

  - Repeatedly sample episode $k \in [1, K]$
  - Apply MC or TD to episode k

Certainty Equivalence

- MC converges to solution with minimum mean-squared error
  - Best fit to the observed returns
  - $\sum_{k=1}^{K} \sum_{t=0}^{T_k} \left(G_t^k - V(s_t^k)\right)^2$
- TD converges to solution of max likelihood Markov model
  - Solution to the MDP $\langle \mathcal{S}, \mathcal{A}, \hat{\mathcal{P}}, \hat{\mathcal{R}}, \gamma \rangle$ that best fits the data
  - $\hat{\mathcal{P}}_{s,s'}^{a} = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=0}^{T_k} \mathbf{1}\left(s_t^k, a_t^k, s_{t+1}^k = s, a, s'\right)$
  - $\hat{\mathcal{R}}_{s}^{a} = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=0}^{T_k} \mathbf{1}\left(s_t^k, a_t^k = s, a\right) r_t^k$

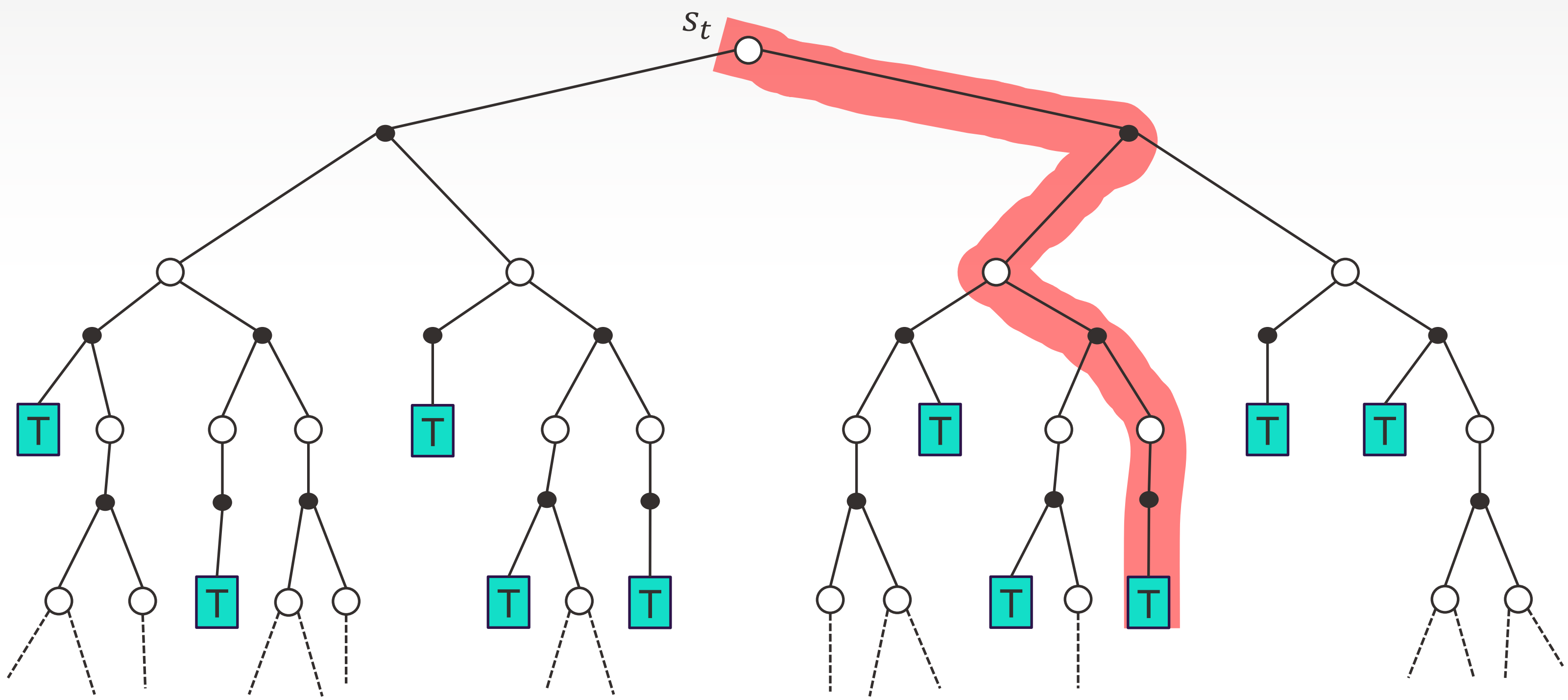
MC vs. TD

- TD exploits Markov property
  - Usually more efficient in Markov environments
- MC does not exploit Markov property
  - Usually more efficient in non-Markov environments

$$P\{X_{t+1} = j \mid X_0 = k_0, X_1 = k_1, \cdots, X_{t-1} = k_{t-1}, X_t = i\}$$
$$= P\{X_{t+1} = j \mid X_t = i\}$$

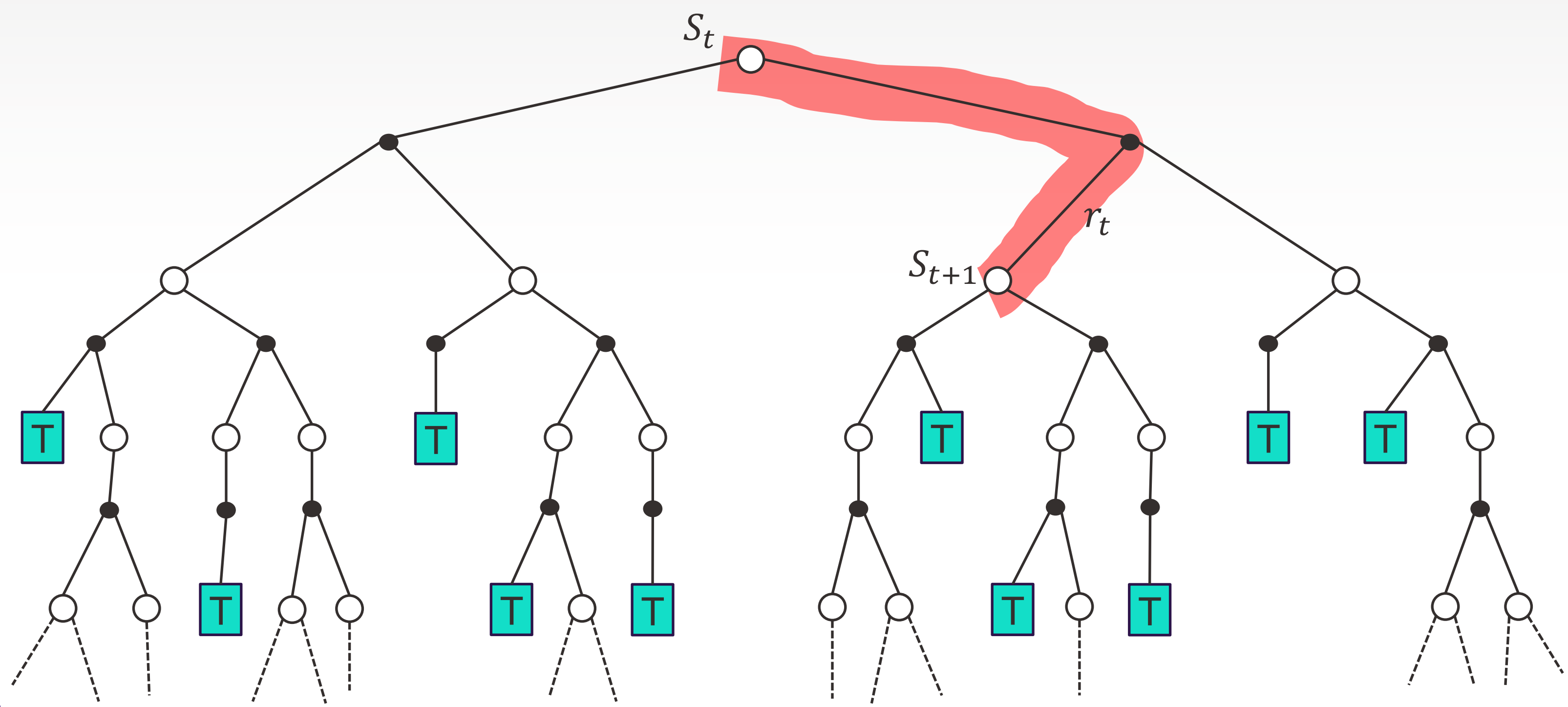
Monte-Carlo Backup

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



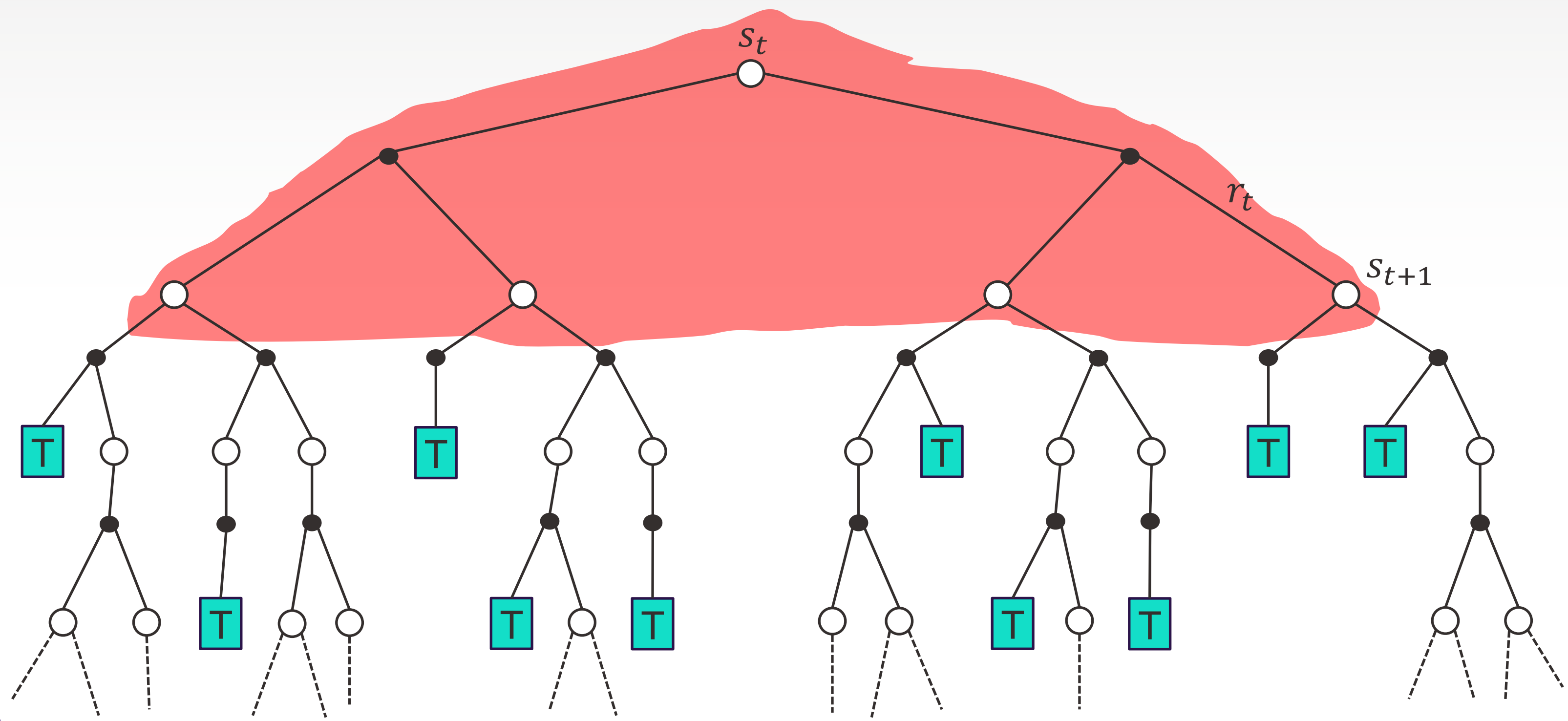
Temporal-Difference Backup

$$V(s_t) \leftarrow V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$$



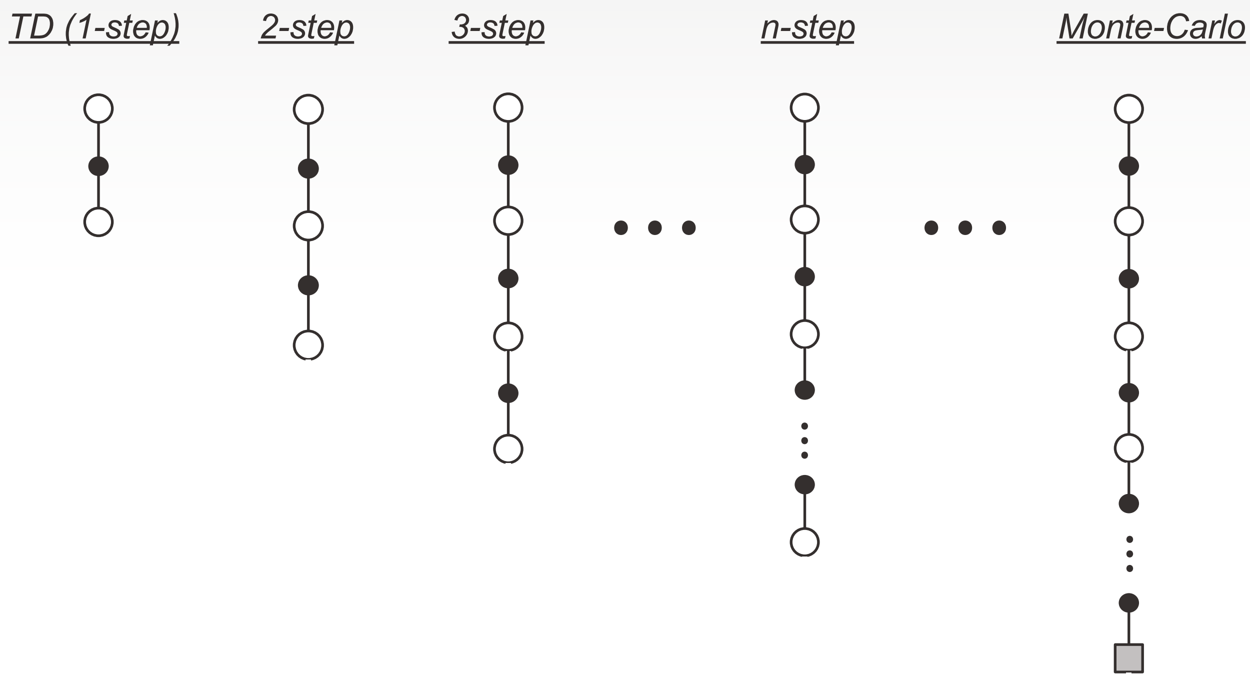
Dynamic Programming Backup

$$V(s_t) \leftarrow \mathbb{E}_\pi[r_t + \gamma V(s_{t+1})]$$



n-Step Prediction

- TD can look n steps into the future



n-Step Return

- Consider n-step returns for $n = 1,2,\ldots$

$$
\begin{array}{lll}
n = 1 & (TD) & G_t^{(1)} = r_t + \gamma V(s_{t+1}) \\
n = 2 & & G_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) \\
\vdots & & \vdots \\
n = \infty & (MC) & G_t^{(\infty)} = r_t + \gamma r_{t+1} + \ldots + \gamma^{T-1} r_T
\end{array}
$$

- n-step return

$$G_t^{(n)} = r_t + \gamma r_{t+1} + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n V(s_{t+n})$$

- n-step temporal-difference learning

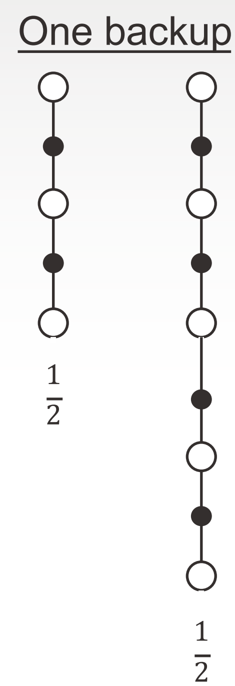
$$V(s) \leftarrow V(s_t) + \alpha\left(G_t^{(n)} - V(s_t)\right)$$

Averaging n-Step Returns

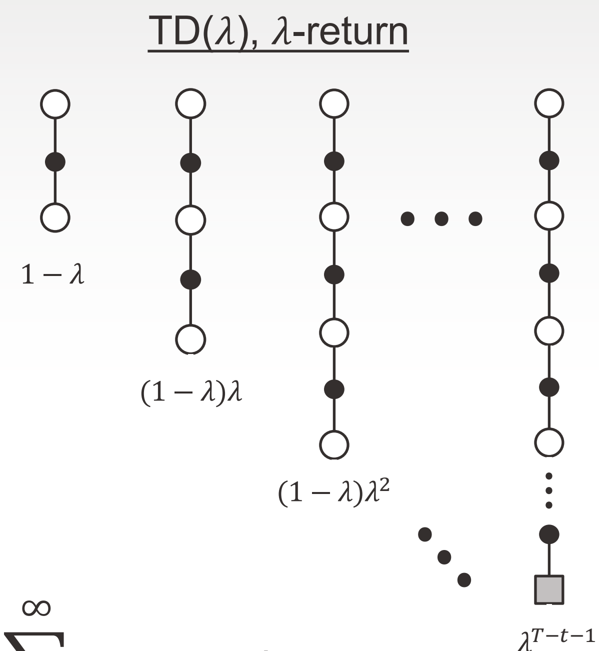
- Average n-step returns over different n
  - Average 2-step and 4-step returns

$$\frac{1}{2}G^{(2)} + \frac{1}{2}G^{(4)}$$

- Combines information from two different time-steps
- All of them
  - Weights should sum to 1



λ-return



$$\sum_{i=0}^{\infty}(1-\lambda)\lambda^i = 1$$

- λ-return G_t^λ combines all n-step returns $G_t^{(n)}$
- Using weight $(1-\lambda)\lambda^{n-1}$

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

- Forward-view TD(λ)

$$V(s_t) \leftarrow V(s_t) + \alpha\left(G_t^\lambda - V(s_t)\right)$$

- Original version is TD(0)

The Credit Assignment Problem

State	Reward	Action
21	0	3
40	0	2
39	0	2
38	0	2
45	0	1
44	0	1
43	100	

- Have great reward at the end
- What state-action led to high reward
- The credit assignment problem

Real-world

- Is this a purely fictitious example?
- Robots
  - Reward 0 except when hitting an object
  - Drone hitting wall otherwise reward 0
- Lose/win (zero-sum) games
  - Tic-tac-toe: reward 0 except when you have 3 in row/column/diagonal
  - Chess: reward 0 except at the end when you win or lose
  - Zero-sum: reward of player 1 = lose of player 2
- Clearly not always the case (inventory, autonomous cars)

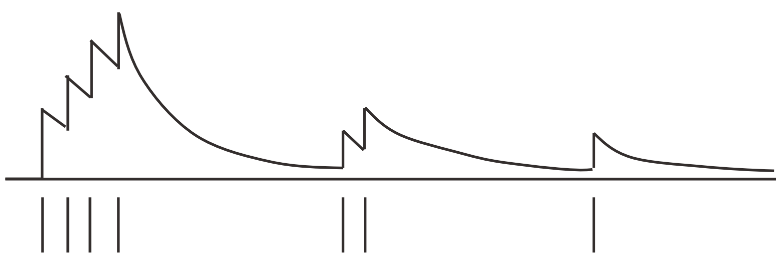
Exploration-Exploitation

- Exploration-Exploitation tradeoff
- Have visited part of the state space and found a reward of 100
  - Is this the best we can hope for?
  - Should we keep ‘pounding’ the visiting states and figure out actions that lead to 100?
  - Perhaps there are other states that we have not yet visited that lead to even higher reward
- Exploitation
  - Should we stick with what we know
  - Find a good policy with respect to this knowledge
    - Risk of missing out on a better reward somewhere else
- Exploration
  - Should we look for states with more reward
    - At risk of wasting time and getting some negative reward

Eligibility Traces

- Keep in mind the assignment problem
- Possible solutions
  - Frequency heuristic
    - Assign credit to most frequent states
  - Recency heuristic
    - Assign credit to most recent states
- Eligibility traces combine both heuristics

$$E_0(s) = 0$$
$$E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbf{1}(s_t = s)$$
γ =hyperparameter



accumulating eligibility trace

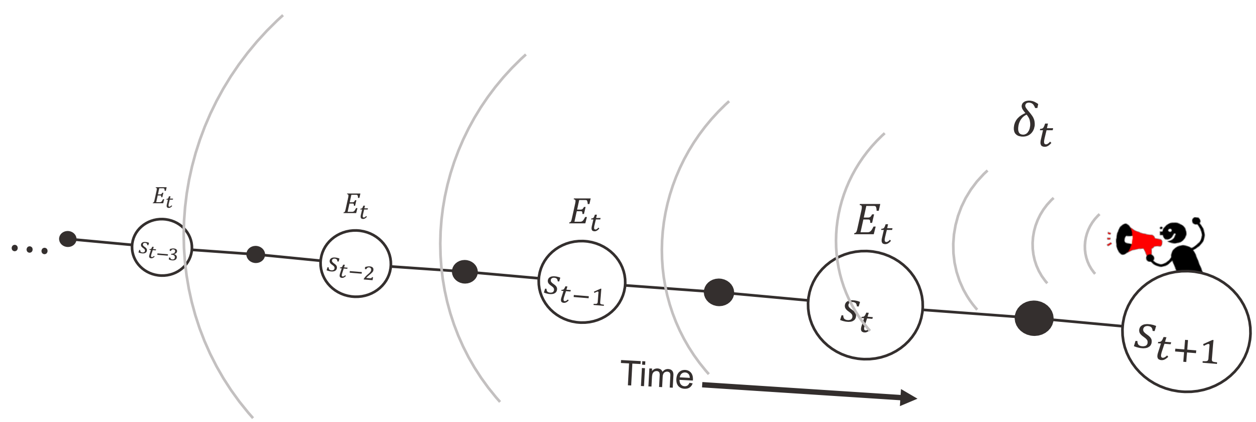
times of visits to a state

Backward View TD(λ)

- 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 $E_t(s)$

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$



TD(λ) and TD(0)

- When $\lambda = 0$, only current state is updated

$$E_t(s) = \mathbf{1}(S_t = s)$$

$$V(s) \leftarrow V(s) + \alpha\delta_t E_t(s)$$

- This is exactly equivalent to TD(0) update

$$V(s_t) \leftarrow V(s_t) + \alpha\delta_t$$

- Parametric value function approximations
  - Eligibility trace replaced with gradient

Algorithm

- Loop
  - For each episode $\tau = (s_0, a_0, r_0, \dots, s_T)$
    - $E = 0$
    - For $t = 0$ to T
      - For each state s
        - $E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbf{1}(s_t = s)$
      - $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$
      - $V(s_t) \leftarrow V(s_t) + \alpha\delta_t E_t(s_t)$
- Alternative
  - $\delta_t = G_t^\lambda - V(s_t)$
- Note that it does not deal with policies
  - We know how to get the final policy

- Parametric approximation to E
- Updated through gradients

Connection

- The sum of offline updates is identical for forward-view and backward-view TD(λ)

$$\sum_{t=0}^{T} \alpha \delta_t E_t(s) = \sum_{t=0}^{T} \alpha \left(G_t^{\lambda} - V(S_t) \right) \mathbf{1}(S_t = s)$$

- Not obvious
  - Can be seen with elementary mathematics

MC and TD(1)

- Consider an episode where s is visited once at time-step k in entire episode
- TD(1) eligibility trace discounts time since visit

$$E_t(s) = \gamma E_{t-1}(s) + 1(s_t = s)$$
$$= \begin{cases} 0 & if\ t < k \\ \gamma^{t-k} & if\ t \geq k \end{cases}$$

- TD(1) updates accumulate error online

$$\sum_{t=0}^{T-1} \alpha \delta_t E_t(s) = \alpha \sum_{t=k}^{T-1} \gamma^{t-k} \delta_t = \alpha(G_k - V(S_k))$$

- By end of episode it accumulates total error

$$\delta_k + \gamma \delta_{k+1} + \gamma^2 \delta_{k+2} + \ \ldots + \gamma^{T-1-k} \delta_{T-1}$$

TD(1)

- Assume $\lambda = 1$
  - Sum of TD errors telescopes into MC error

$$
\begin{aligned}
&\delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \ \dots + \gamma^{T-1-t}\delta_{T-1} \\
&= r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \\
&+ \gamma r_{t+2} + \gamma^2 V(s_{t+2}) - \gamma V(s_{t+1}) \\
&+ \gamma^2 r_{t+3} + \gamma^3 V(s_{t+3}) - \gamma^2 V(s_{t+2}) \\
&\quad \vdots \\
&+ \gamma^{T-1-t} r_T + \gamma^{T-t} V(s_T) - \gamma^{T-1-t} V(s_{T-1}) \\
&= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3\dots} + \gamma^{T-1-t} r_T - V(s_t) \\
&= G_t - V(s_t)
\end{aligned}
$$

  - This holds also if t replaced by k
    - Shows equation on previous slide

TD(1)

- TD(1) roughly equivalent to every-visit Monte-Carlo
- Error accumulated online
  - Step-by-step
- If value function only updated offline at end of episode
  - Total update is exactly the same as MC

TD(λ)

- General λ
- $$\begin{aligned} G_t^\lambda - V(s_t) = -V(s_t) &+ (1-\lambda)\lambda^0\big(r_t + \gamma V(s_{t+1})\big) \\ &+ (1-\lambda)\lambda^1\big(r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2})\big) \\ &+ (1-\lambda)\lambda^2\big(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V(s_{t+3})\big) \\ &+ \ \dots \\ = -V(s_t) &+ (\gamma\lambda)^0\,(r_t + \gamma V(s_{t+1}) - \gamma\lambda V(s_{t+1})) \\ &+ (\gamma\lambda)^1(r_{t+1} + \gamma V(s_{t+2}) - \gamma\lambda V(s_{t+2})) \\ &+ (\gamma\lambda)^2(r_{t+2} + \gamma V(s_{t+3}) - \gamma\lambda V(s_{t+3})) \\ &+ \ \dots \\ = \quad & \quad (\gamma\lambda)^0(r_t + \gamma V(s_{t+1}) - V(s_t)) \\ &+ (\gamma\lambda)^1(r_{t+1} + \gamma V(s_{t+2}) - V(s_{t+1})) \\ &+ (\gamma\lambda)^2(r_{t+2} + \gamma V(s_{t+3}) - V(s_{t+2})) \\ &+ \ \dots \\ = \delta_t + \gamma\lambda\delta_{t+1} &+ (\gamma\lambda)^2\delta_{t+2} + \ \dots \end{aligned}$$

Summary

- TD(0) is the original version
  - With eligibility traces
  - One step look ahead
- TD(1)
  - Look ahead all the way to the end of an episode
    - But from the first occurrence of state
  - Very similar to MC
- TD(λ)
  - Errors weighted by $\gamma\lambda$ all the way through the end of an episode