Lecture 4: Model-Free Prediction

Hado van Hasselt

Outline

1. Introduction
2. Monte-Carlo Learning
3. Temporal-Difference Learning
4. TD(λ)

Reading: (Sutton & Barto Oct 2015) Chapters 5, 6, and 7 on prediction

Sample-based reinforcement learning

- Last lecture:
  - Planning by dynamic programming
  - Solve a *known* MDP
- This lecture:
  - Model-free prediction
  - Estimate the value function of an *unknown* MDP
- Next lecture:
  - Model-free control
  - Optimise the value function of an *unknown* MDP

Sample-based reinforcement learning

- Sample-based methods learn directly from episodes of experience
- We call this Monte Carlo
- MC is model-free: no knowledge of MDP required, only samples
- Simplest version:
  - Consider policy evaluation
  - Roll out a full trajectories until termination
  - Average the returns along these trajectories
  - (Caveat: only applies in episodic problems)

Sample-based reinforcement learning

- Simple example, multi-armed bandit:
  - m actions, goal is to estimate value of each action
  - select action A_t, receive reward R_{t+1}
  - 'value' = expected reward =

$$v(a) = \mathbb{E}\left[R_{t+1} | A_t = a\right]$$

  - approximation = average observed reward =

$$\frac{1}{n(a)} \sum_{t=0}^{n} \mathcal{I}(A_t = a) R_{t+1}$$

where $\mathcal{I}(\cdot)$ is the indicator function and
$n(a) = \sum_{t=0}^{n} \mathcal{I}(A_t = a)$ is the number of times action a was selected

Contextual bandits

- Imagine a bandit problem where the state might change
  - each episode still ends immediately
  - actions do not affect the states
  - e.g., different visitors to a website
- Then, we want to estimate

$$v(s, a) = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right]$$

- v could be a neural network, could use loss

$$l_t(\theta) = (v_\theta(S_t, A_t) - R_{t+1})^2$$

Monte-Carlo Policy Evaluation

- Now consider sequential decision problems
- Goal: learn v_π from episodes of experience under policy π

$$S_1, A_1, R_2, ..., S_k \sim \pi$$

- The *return* is the total discounted reward (for an episode ending at time T, where $T > t$):

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-t-1} R_T$$

- The value function is the expected return:

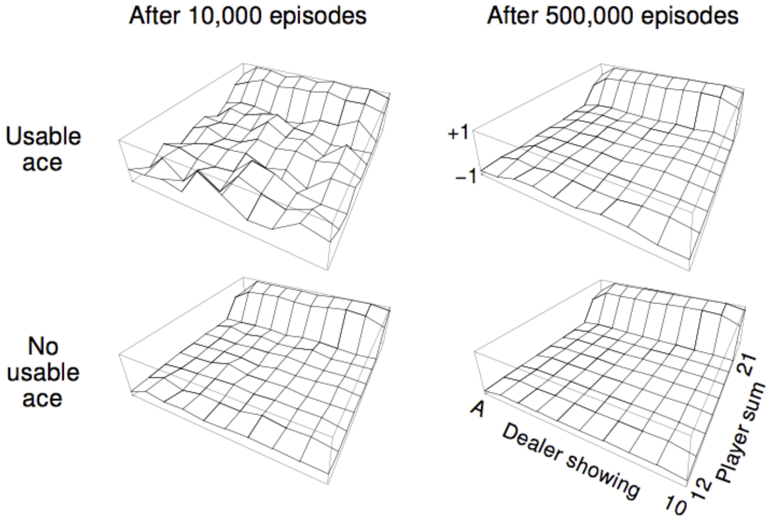
$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

- Monte-Carlo policy evaluation uses *empirical mean* return instead of *expected* return

Blackjack Example

- States (200 of them):
  - Current sum (12-21)
  - Dealer's showing card (ace-10)
  - Do I have a "useable" ace? (yes-no)
- Action stick: Stop receiving cards (and terminate)
- Action draw: Take another card (random, no replacement)
- Reward for stick:
  - $+1$ if sum of cards $>$ sum of dealer cards
  - 0 if sum of cards $=$ sum of dealer cards
  - -1 if sum of cards $<$ sum of dealer cards
- Reward for draw:
  - -1 if sum of cards > 21 (and terminate)
  - 0 otherwise
- Transitions: automatically draw if sum of cards < 12

Blackjack Value Function after Monte-Carlo Learning



Policy: stick if sum of cards ≥ 20, otherwise twist

Previous lecture: approximate dynamic programming

- Use a *function approximator* $v_\theta(s)$, with parameters $\theta \in \mathbb{R}^n$.
- Use dynamic programming to compute θ_{k+1} from θ_k.
  - Sample states $\tilde{\mathcal{S}} \subseteq \mathcal{S}$ (more simply, let $\tilde{S} = S$)
  - For each sample state $s \in \tilde{\mathcal{S}}$, compute target value using Bellman optimality equation,
  - Then

$$\tilde{v}_k(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_{\theta_k}(s') \right)$$

$$\theta_{k+1} = \underset{\theta}{\operatorname{argmin}} \sum_{s \in \tilde{\mathcal{S}}} \left(v_\theta(s) - \tilde{v}_k(s) \right)^2$$

Approximate dynamic programming

- Main idea: approximate $\tilde{v}_k \approx v_\pi$, then update

$$\theta_{k+1} = \underset{\theta}{\operatorname{argmin}} \sum_{s \in \tilde{\mathcal{S}}} \left(\tilde{v}(s) - v_\theta(s) \right)^2$$

- Extension 1: do not find the full minimum. Instead, follow gradient:

$$\theta_{k+1} = \alpha \sum_{s \in \tilde{\mathcal{S}}} \left(\tilde{v}(s) - v_\theta(s) \right) \nabla_\theta v_\theta(s)$$

- Extension 2: sample a single state S_t:

$$\theta_{t+1} = \alpha \left(\tilde{v}(S_t) - v_\theta(S_t) \right) \nabla_\theta v_\theta(S_t)$$

Temporal difference learning

- In (approximate) DP: we use model and compute

$$\tilde{v}_k(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_{\theta_k}(s') \right)$$

- Instead we could sample. In Monte Carlo, we use:

$$\tilde{v}_k(S_t) = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$$

- Alternatively, we could bootstrap, and use

$$\tilde{v}_k(S_t) = R_{t+1} + \gamma v_{\theta_k}(S_{t+1}) .$$

- This is called temporal-difference learning

Temporal-Difference Learning

- TD methods learn directly from experience
- TD is *model-free*: no knowledge of MDP transitions / rewards
- TD also learns from *incomplete* episodes, by *bootstrapping*
- TD updates a guess towards a guess

MC and TD

- Goal: learn v_π online from experience under policy π
- Incremental Monte-Carlo
  - Update value $v(S_t)$ towards sampled return G_t

$$v(S_t) \leftarrow v(S_t) + \alpha \left(G_t - v(S_t) \right)$$

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $v(S_t)$ towards *estimated* return $R_{t+1} + \gamma v(S_{t+1})$

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

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

MC and TD, with function approximation

- Goal: learn v_π online from experience under policy π
- Incremental Monte-Carlo
  - Update value $v(S_t)$ towards sampled return G_t

$$\theta_{t+1} = \theta_t + \alpha \left(R_{t+1} + \gamma v(S_{t+1}) - v_{\theta_t}(S_t) \right) \nabla_\theta v_\theta(S_t)$$

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $v(S_t)$ towards *estimated* return $R_{t+1} + \gamma v(S_{t+1})$

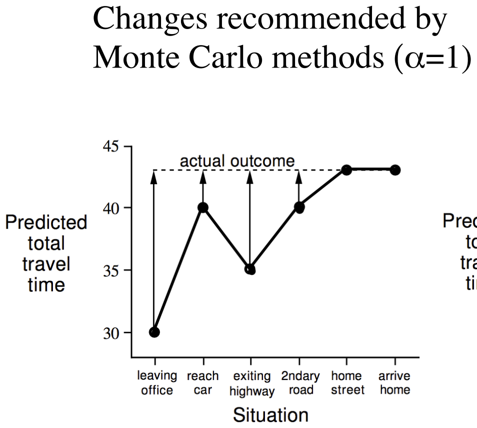
$$\theta_{t+1} = \theta_t + \alpha \left(R_{t+1} + \gamma v(S_{t+1}) - v_{\theta_t}(S_t) \right) \nabla_\theta v_\theta(S_t)$$

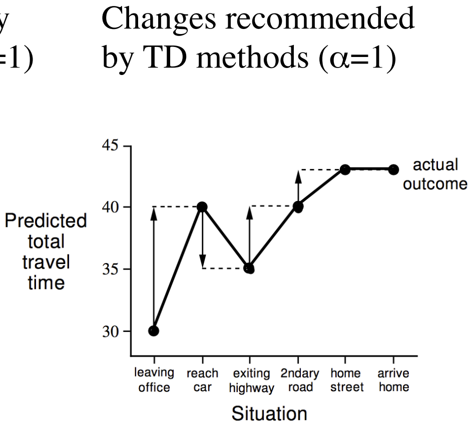
  - $\delta_t = R_{t+1} + \gamma v(S_{t+1}) - v(S_t)$ is called the *TD error*

Driving Home Example

State	Elapsed Time (minutes)	Predicted Time to Go	Predicted Total Time
leaving office	0	30	30
reach car, raining	5	35	40
exit highway	20	15	35
behind truck	30	10	40
home street	40	3	43
arrive home	43	0	43

Driving Home Example: MC vs. TD





Advantages and Disadvantages 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 *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

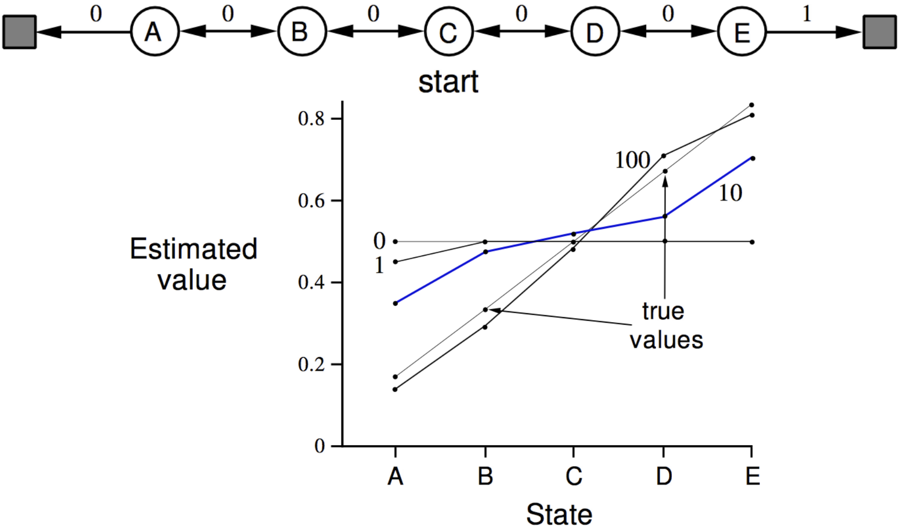
Bias/Variance Trade-Off

- Return $G_t = R_{t+1} + \gamma R_{t+2} + \ldots$ is an unbiased estimate of $v_\pi(S_t)$
- TD target $R_{t+1} + \gamma v(S_{t+1})$ is a biased estimate of $v_\pi(S_t)$
  - Unless $v(S_{t+1}) = v_\pi(S_{t+1})$
- But the TD target has much lower variance:
  - Return depends on *many* random actions, transitions, rewards
  - TD target depends on *one* random action, transition, reward

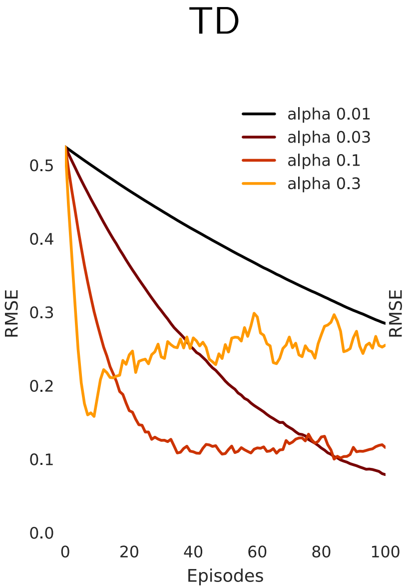
Advantages and Disadvantages of MC vs. TD (2)

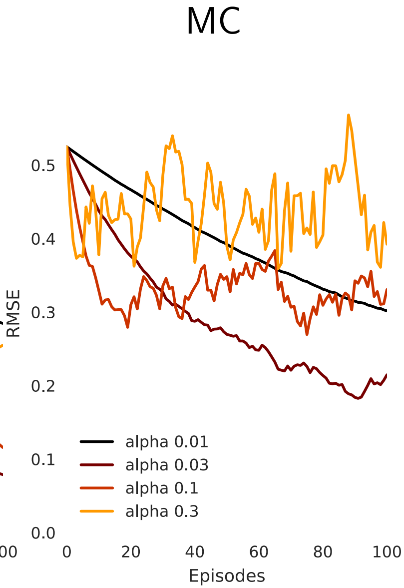
- MC has high variance, zero bias
  - Good convergence properties
  - (even with function approximation)
  - Very simple to understand/analyse
- TD has low variance, some bias
  - Usually more efficient than MC
  - TD(0) converges to $v_\pi(s)$
  - (but not always with function approximation)
  - More sensitive to initial value

Random Walk Example



Random Walk: MC vs. TD





Batch MC and TD

- TabularMC and TD converge: $v(s) \rightarrow v_\pi(s)$ as experience $\rightarrow \infty$ and $\alpha \rightarrow 0$
- But what about finite experience?

$$\begin{array}{ll} \text{episode 1:} & s_1^1, a_1^1, r_2^1, ..., s_{T_1}^1 \\ & \vdots \\ \text{episode K:} & s_1^K, a_1^K, r_2^K, ..., s_{T_K}^K \end{array}$$

  - e.g. Repeatedly sample episode $k \in [1, K]$
  - Apply MC or TD(0) to episode k
  - $=$ sampling from an *empirical model*

AB Example

Two states A, B; no discounting; 8 episodes of experience

$\mathrm{A}, 0, \mathrm{B}, 0$

$\mathrm{B}, 1$

$\mathrm{B}, 1$

$\mathrm{B}, 1$

$\mathrm{B}, 1$

$\mathrm{B}, 1$

$\mathrm{B}, 1$

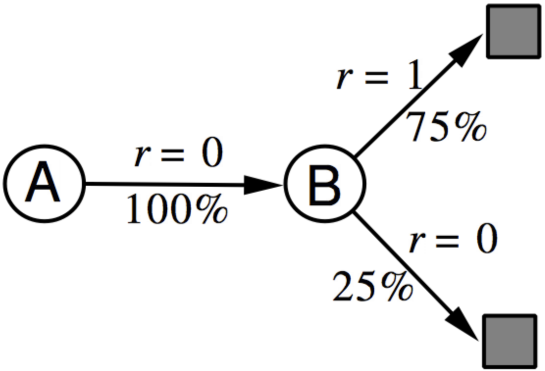
$\mathrm{B}, 0$

What is $v(A), v(B)$?

AB Example

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 $v(A), v(B)$?

Certainty Equivalence

- MC converges to solution with minimum mean-squared error
  - Best fit to the observed returns

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

  - In the AB example, $v(A) = 0$
- TD(0) converges to solution of max likelihood Markov model
  - Solution to the empirical MDP $\langle \mathcal{S}, \mathcal{A}, \hat{\mathcal{P}}, \hat{\mathcal{R}}, \gamma \rangle$ that best fits the data

$$\hat{\mathcal{P}}_{s,s'}^{a} = \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k}\mathbf{1}(s_t^k, a_t^k, s_{t+1}^k = s, a, s')$$

$$\hat{\mathcal{R}}_{s}^{a} = \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k}\mathbf{1}(s_t^k, a_t^k = s, a) r_t^k$$

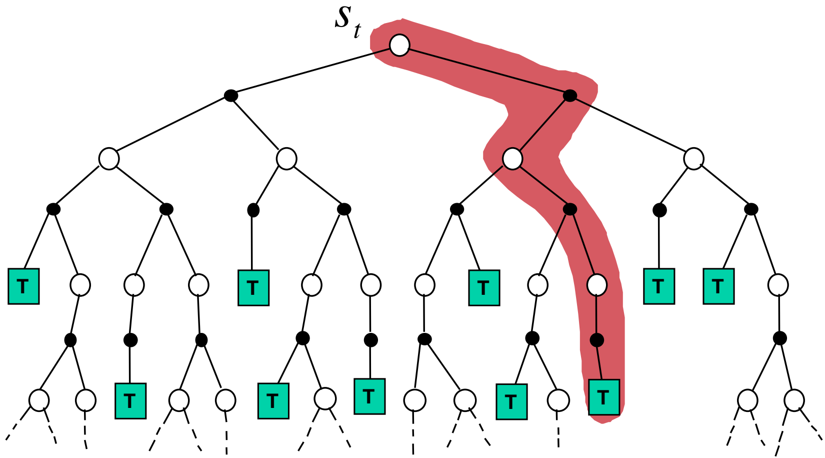
  - In the AB example, $v(A) = 0.75$

Advantages and Disadvantages of MC vs. TD (3)

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

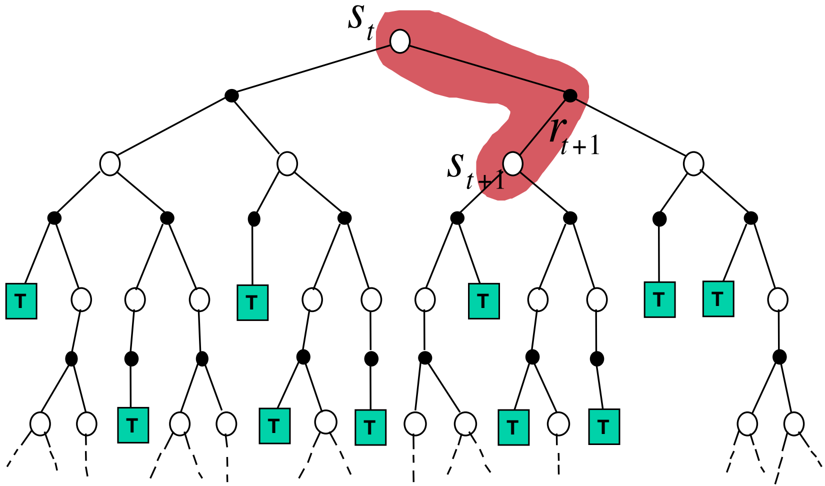
Monte-Carlo Backup

$$v(S_t) \leftarrow v(S_t) + \alpha\left(G_t - v(S_t)\right)$$



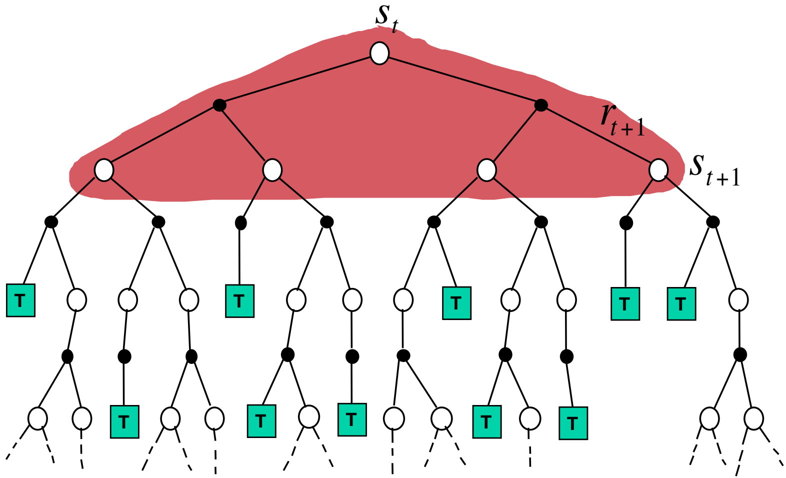
Temporal-Difference Backup

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



Dynamic Programming Backup

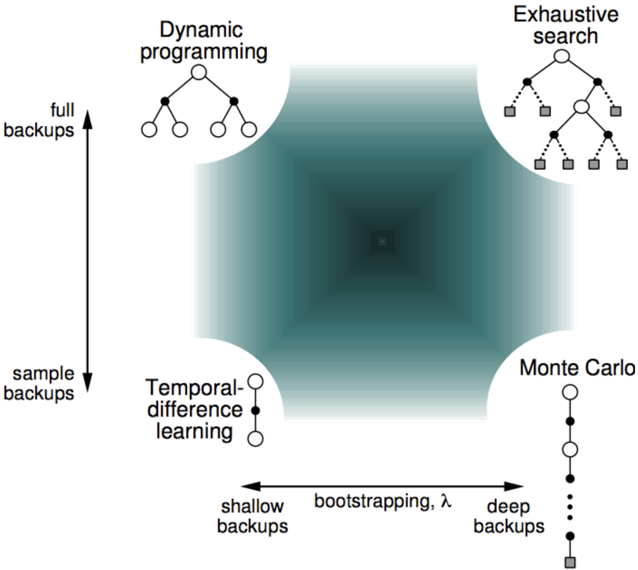
$$v(S_t) \leftarrow \mathbb{E}_\pi\left[R_{t+1} + \gamma v(S_{t+1})\right]$$



Bootstrapping and Sampling

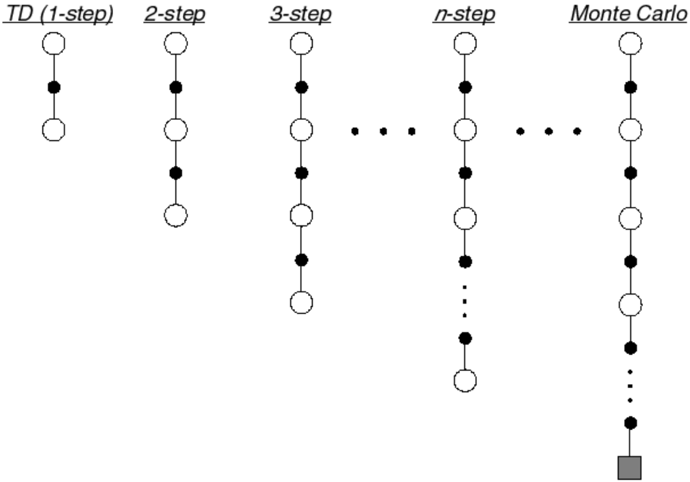
- Bootstrapping: update involves an estimate
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- Sampling: update samples an expectation
  - MC samples
  - DP does not sample
  - TD samples

Unified View of Reinforcement Learning



n-Step Prediction

- Let TD target look n steps into the future



n-Step Return

- Consider the following n-step returns for $n = 1, 2, \infty$:

$$\begin{array}{lll}
n = 1 & (TD) & G_t^{(1)} = R_{t+1} + \gamma v(S_{t+1}) \\
n = 2 & & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 v(S_{t+2}) \\
\vdots & & \vdots \\
n = \infty & (MC) & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-t-1} R_T
\end{array}$$

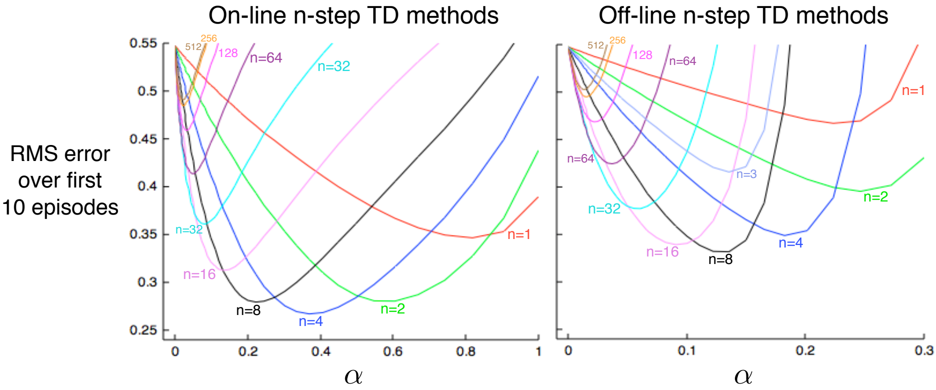
- Define the n-step return

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n v(S_{t+n})$$

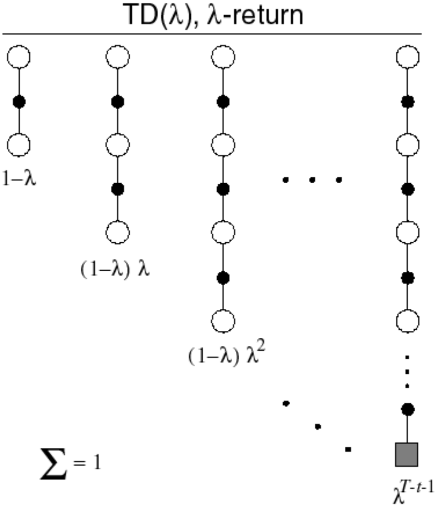
- n-step temporal-difference learning

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

Large Random Walk Example



λ-return



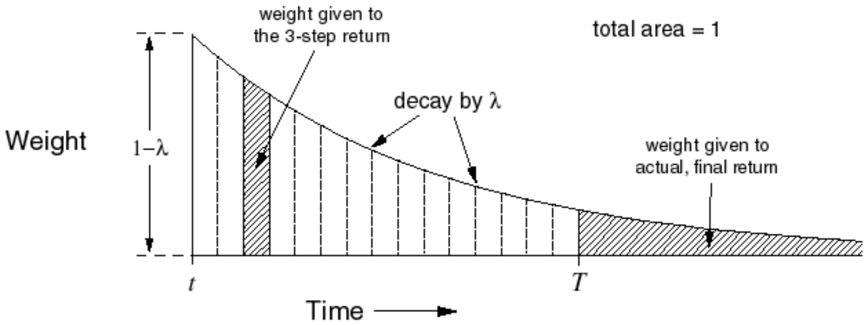
- The *λ-return* G_t^λ combines all n-step returns $G_t^{(n)}$
- Using weight $(1-\lambda)\lambda^{n-1}$

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

- Forward-view TD(λ)

$$v(S_t) \leftarrow v(S_t) + \alpha \left(G_t^\lambda - v(S_t) \right)$$

TD(λ) Weighting Function



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

TD(λ)

- Equivalence:

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

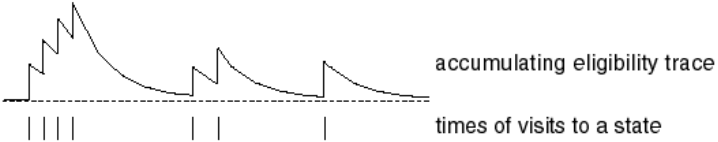
$$G_t^\lambda = R_{t+1} + \gamma\left((1-\lambda)v(S_{t+1}) + \lambda G_{t+1}^\lambda\right)$$

- Interpretation:
  - Observe reward
  - Continue with weight/probability γ
  - Bootstrap with weight $(1-\lambda)$, continue with remaining weight λ
  - Observe next reward, and continue
- Forward-view looks into the future to compute G_t^λ
- Like MC, can only be computed from complete episodes

Eligibility Traces

- *Eligibility traces* allow us to implement TD(λ) online

$$\mathbf{e}_0(s) = 0$$
$$\mathbf{e}_t = \gamma\lambda\mathbf{e}_{t-1}(S_t) + \nabla_\theta v_\theta(S_t)$$



Background reading:
van Hasselt & Sutton 2015, Chapter 12 of Sutton & Barto 2017

TD(λ) and TD(1)

- TD(1) is roughly equivalent to Monte-Carlo
- (Can be made exact for tabular or linear function approximation)

TD(λ) with function approximation

- With tabular or linear function approximation, TD(n) and TD(λ) converge
  - Solution depends on n or λ, and on the function class (i.e., on the features)
  - Typically, the solution will have lower error if either the function is more flexible (better/more features), and if n or λ is higher

Experience replay

- In dynamic programming, we query a model
- With samples, we can do something similar:
  - Store sampled transitions $\{(S_n, A_n, R_{n+1}, S_{n+1})\}$
  - Replay these to the algorithm
- Useful if data is expensive
- Useful for more diverse updates (e.g., closer to i.i.d.)
- Can be interpreted as using an empirical model

Using stationary targets

- In approximate dynamic programming, we often update multiple states towards targets depending on v_k to obtain v_{k+1}
- Similarly, in TD we can keep the bootstrap value function fixed for a while
  - E.g., copy $\theta_t^- = \theta_t$, and then keep θ_t^- fixed for several time steps

$$
\begin{aligned}
\theta_{t+1}^- &= \theta_t^- \\
\theta_{t+1} &= \theta_t + \alpha \left(R_{t+1} + \gamma v_{\theta_t^-}(S_{t+1}) - v_{\theta_t}(S_t) \right) \nabla_{\theta_t} v_{\theta_t}(S_t)
\end{aligned}
$$

  - Periodically make a fresh copy
- Experience replay and stationary targets were important to stabilize learning in DQN on Atari

Deep Q-networks

- We can apply the same algorithms to action values $q(s, a)$, updating towards

$$R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) q(S_{t+1}, a)$$

- Additionally, we can learn about policies other than current π. For instance, greedy policy:

$$R_{t+1} + \gamma \max_a q(S_{t+1}, a)$$

- We can combine this with 1) deep convolutional networks to represent q_θ, 2) experience replay, and 3) stationary targets q_{θ^-}
- This gives the DQN algorithm (more in next lecture)

Background reading:
Mnih, Kavukcuoglu, Silver, et al. 2015, Nature