Lecture 4: Model-Free Prediction and Control

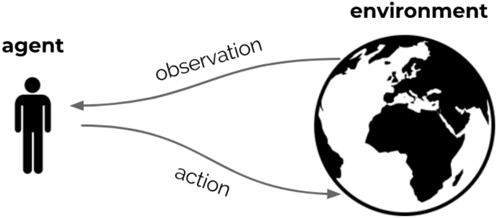
Hado van Hasselt

January 22, 2020, UCL

Background

Sutton & Barto 2018, Chapters $5 + 6$ $(+ 7 + 12)$

Recap



- Reinforcement learning is the science of learning to make decisions
- Agents can learn a **policy**, **value function** and/or a **model**
- The general problem involves taking into account **time** and **consequences**
- Decisions affect the **reward**, the **agent state**, and **environment state**

Sample-based reinforcement learning

- Last lecture:
  - **Planning** by **dynamic programming** to solve a known MDP
- This lecture & next lecture:
  - **Model-free prediction** to **estimate** values in an **unknown** MDP
  - **Model-free control** to **optimise** values in an **unknown** MDP
- Not yet:
  - Learning policies directly in sequential problems (policy gradients)
  - Continuous MDPs
  - Deep reinforcement learning

Sample-based reinforcement learning

- We can use experience **samples** to learn without a model
- We call direct sampling of episodes **Monte Carlo**
- MC is **model-free**: no knowledge of MDP required, only samples

Sample-based reinforcement learning

- Simple example, **multi-armed bandit**:
  - For each action, average reward samples

$$q_t(a) \;=\; \frac{\sum_{i=0}^{t} \mathcal{I}(A_i = a) R_{i+1}}{\sum_{i=0}^{t} \mathcal{I}(A_i = a)} \quad\approx\quad \mathbb{E}\left[R_{t+1} | A_t = a\right] \;=\; q(a)$$

  - Equivalently:

$$\begin{aligned} q_t(A_t) &= q_{t-1}(A_t) + \alpha_t(R_t - q_{t-1}(A_t)) \\ q_t(a) &= q_{t-1}(a) & \forall a \neq A_t \end{aligned}$$

with $\alpha_t = \frac{1}{N_t(A_t)} = \frac{1}{\sum_{i=0}^{t} \mathcal{I}(A_i = a)}$

Note: we changed notation $R_t \rightarrow R_{t+1}$ for the reward after A_t
In MDPs, the reward is said to arrive on the time step after the action

Contextual bandits

- Consider bandits with different states ('context')
  - episodes are still one step
  - actions do not affect states
  - no long-term consequences
- Then, we want to estimate

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

- q could be a parametric function, e.g., neural network, and we could use loss

$$L(w) = \frac{1}{2}\mathbb{E}\left[(R_{t+1} - q_w(S_t, A_t))^2\right]$$

- Also works for large (continuous) state spaces $\mathcal{S}$ — this is just **regression**

Contextual bandits (continued)

- We want to minimise

$$L(w) = \frac{1}{2}\mathbb{E}\left[(R_{t+1} - q_w(S_t, A_t))^2\right]$$

- Then the gradient update is

$$\begin{aligned} w_{t+1} &= w_t - \alpha \nabla_{w_t} L(w_t) \\ &= w_t - \alpha \nabla_{w_t} \frac{1}{2}\mathbb{E}\left[(R_{t+1} - q_{w_t}(S_t, A_t))^2\right] \\ &= w_t + \alpha \mathbb{E}\left[(R_{t+1} - q_w(S_t, A_t))\nabla_{w_t} q_{w_t}(S_t, A_t)\right] . \end{aligned}$$

  We can sample this to get a **stochastic gradient update** (SGD)
- The tabular case is a special case (only updates the value in cell $[S_t, A_t]$)

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 $T > t$):

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

- The value function is the expected return:

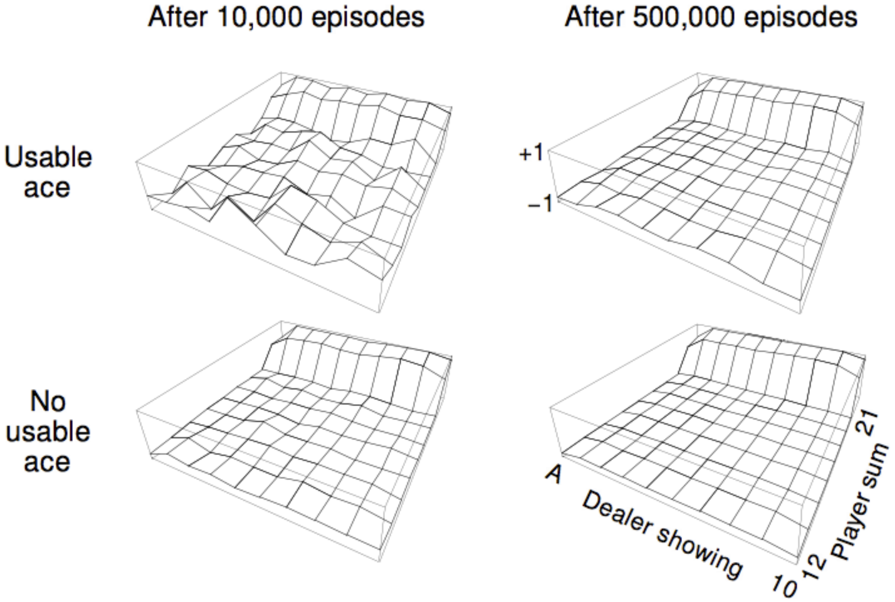
$$v_\pi(s) = \mathbb{E}\left[G_t \mid S_t = s, \pi\right]$$

- We can just use **sample average** return instead of **expected** return
- We call this **Monte Carlo policy evaluation**

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



Temporal Difference Learning by Sampling Bellman Equations

- Previous lecture: Bellman equations,

$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t \sim \pi(S_t)\right]$$

- Previous lecture: Approximate by iterating,

$$v_{k+1}(s) = \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t \sim \pi(S_t)\right]$$

- We can sample this!

$$v_{t+1}(S_t) = R_{t+1} + \gamma v_t(S_{t+1})$$

- This is likely quite noisy — better to average:

$$v_{t+1}(S_t) = v_t(S_t) + \alpha_t \left(\underbrace{R_{t+1} + \gamma v_t(S_{t+1})}_{\text{target}} - v_t(S_t) \right)$$

Temporal difference learning

- In dynamic programming (DP) we use one step of the model and **bootstrap**

$$\text{target} \quad = \quad \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, \pi\right] \tag{DP}$$

- In Monte Carlo (MC) we **sample**, and use:

$$\text{target} \quad = \quad G_t \quad (= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots) \tag{MC}$$

- Alternatively, we could **sample and bootstrap**, and use

$$\text{target} \quad = \quad R_{t+1} + \gamma v_t(S_{t+1})\,. \tag{TD}$$

- This is called temporal-difference learning (TD)

Temporal-Difference Learning

- TD is **model-free** (no knowledge of MDP) and learn directly from experience
- TD can learn from **incomplete** episodes, by **bootstrapping**
- TD can learn **during** each episode

MC and TD

- Consider **prediction**: learn v_π online from experience under policy π
- Incremental Monte-Carlo
  - Update value $v_n(S_t)$ towards sampled return $\boldsymbol{G_t}$

$$v_{n+1}(S_t) = v_n(S_t) + \alpha \left(\boldsymbol{G_t} - v_n(S_t) \right)$$

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

$$v_{t+1}(S_t) \leftarrow v_t(S_t) + \alpha \left(\overbrace{\underbrace{\boldsymbol{R_{t+1} + \gamma v_t(S_{t+1})}}_{\text{target}} - v_t(S_t)}^{\text{TD error}} \right)$$

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

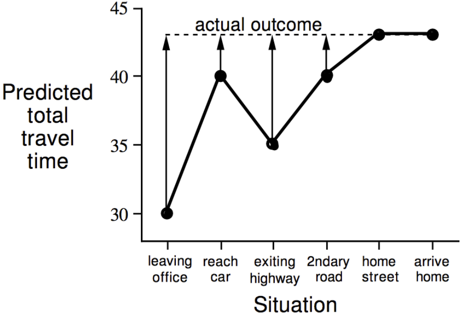
Driving Home Example

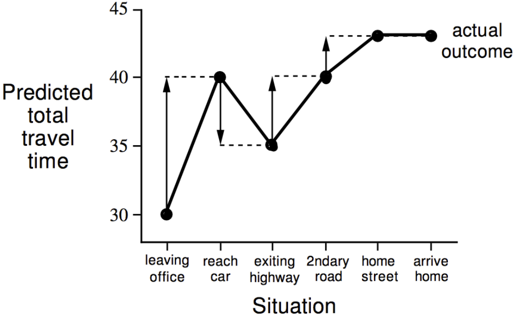
State	Elapsed Time (minutes)	Predicted Time to Go	Predicted Total Time
leaving office	0	30	30
reach car, raining	5	35	40
exit highway	20	15	35
behind truck	30	10	40
home street	40	3	43
arrive home	43	0	43

Driving Home Example: MC vs. TD

Changes recommended by Monte Carlo methods (α=1)

Changes recommended by TD methods (α=1)





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
- TD needs reasonable value estimates

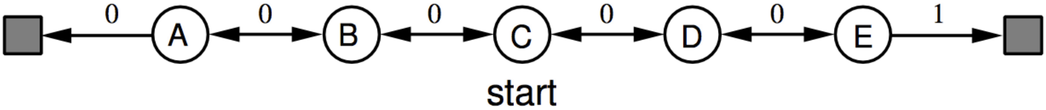
Bias/Variance Trade-Off

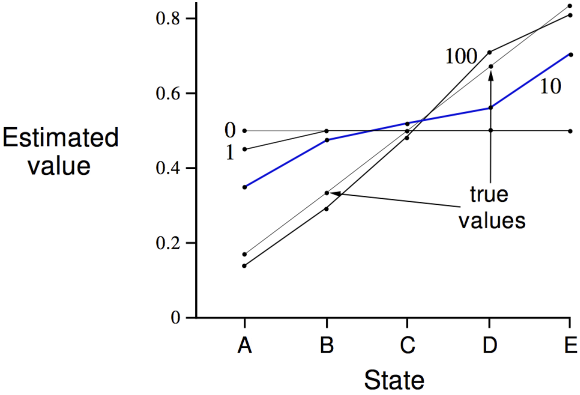
- MC return $G_t = R_{t+1} + \gamma R_{t+2} + \ldots$ is an **unbiased** estimate of $v_\pi(S_t)$
- TD target $R_{t+1} + \gamma v_t(S_{t+1})$ is a **biased** estimate of $v_\pi(S_t)$
  (Unless $v_t(S_{t+1}) = v_\pi(S_{t+1})$)
- But the TD target has **lower variance**:
  - Return depends on **many**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

Bias/Variance Trade-Off

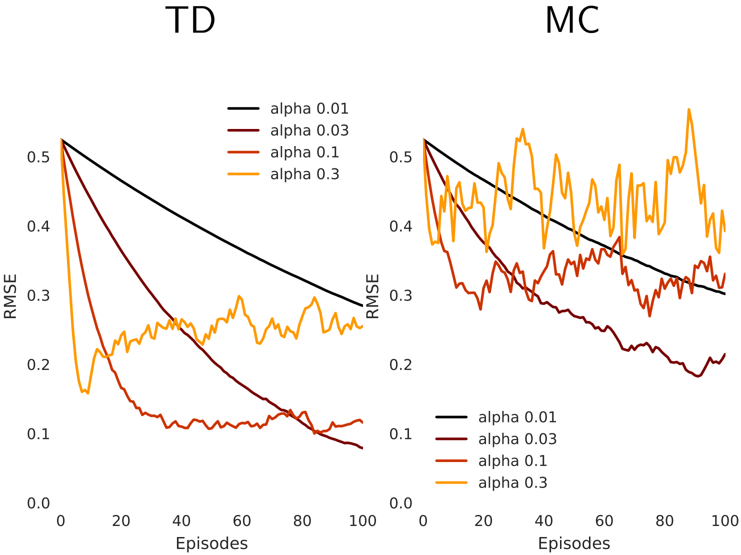
- In some cases, TD can have irreducible bias
- The world may be partially observable
  - MC would implicitly account for all the latent variables
- The function to approximate the values may fit poorly
- In the tabular case, both MC and TD will converge: $v_t \rightarrow v_\pi$

Random Walk Example





Random Walk: MC vs. TD



Batch MC and TD

- Tabular MC and TD converge: $v_t \to v_\pi$ as experience $\to \infty$ and $\alpha_t \to 0$
- But what about finite experience?
- Consider a fixed batch of experience:

$$
\begin{array}{ll}
\text{episode 1:} & S_1^1, A_1^1, R_2^1, ..., S_{T_1}^1 \\
 & \vdots \\
\text{episode K:} & S_1^K, A_1^K, R_2^K, ..., S_{T_K}^K
\end{array}
$$

- Repeatedly sample each episode $k \in [1, K]$ and apply MC or TD(0)
  - $=$ sampling from an **empirical model**

AB Example

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

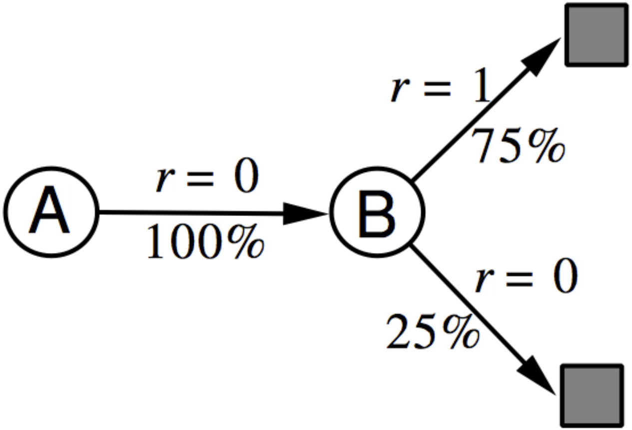
$$\mathrm{A}, 0, \mathrm{B}, 0$$
$$\mathrm{B}, 1$$
$$\mathrm{B}, 1$$
$$\mathrm{B}, 1$$
$$\mathrm{B}, 1$$
$$\mathrm{B}, 1$$
$$\mathrm{B}, 1$$
$$\mathrm{B}, 0$$

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

AB Example

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

$$\mathrm{A}, 0, \mathrm{B}, 0$$
$$\mathrm{B}, 1$$
$$\mathrm{B}, 1$$
$$\mathrm{B}, 1$$
$$\mathrm{B}, 1$$
$$\mathrm{B}, 1$$
$$\mathrm{B}, 1$$
$$\mathrm{B}, 0$$



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

Differences in batch solutions

- MC converges to best mean-squared fit for the observed returns

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

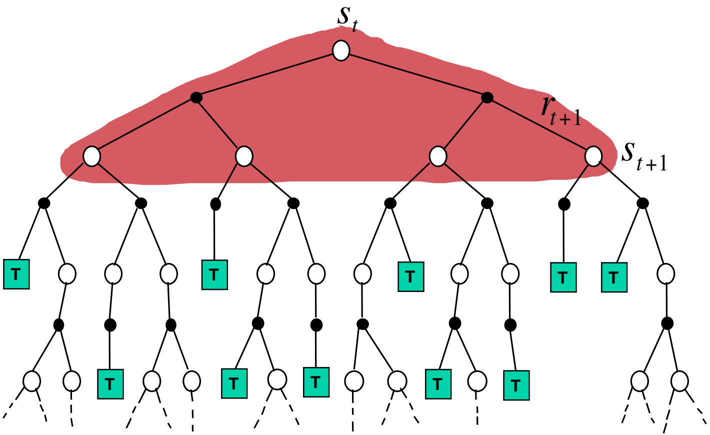
- In the AB example, $\boldsymbol{v(A) = 0}$
- TD converges to solution of max likelihood Markov model, given the data
  - Solution to the empirical MDP $(\mathcal{S}, \mathcal{A}, \hat{p}, \gamma)$ that best fits the data
  - In the AB example: $\hat{p}(S_{t+1} = B \mid S_t = A) = 1$, and therefore $\boldsymbol{v(A) = v(B) = 0.75}$

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
- With finite data, or with function approximation, the solutions may differ

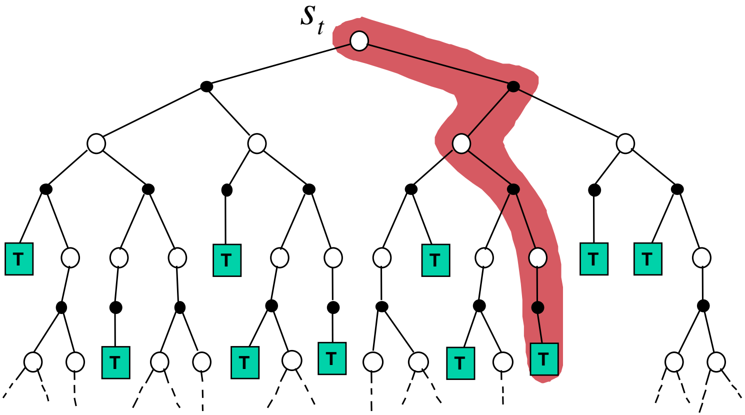
Dynamic Programming Backup

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



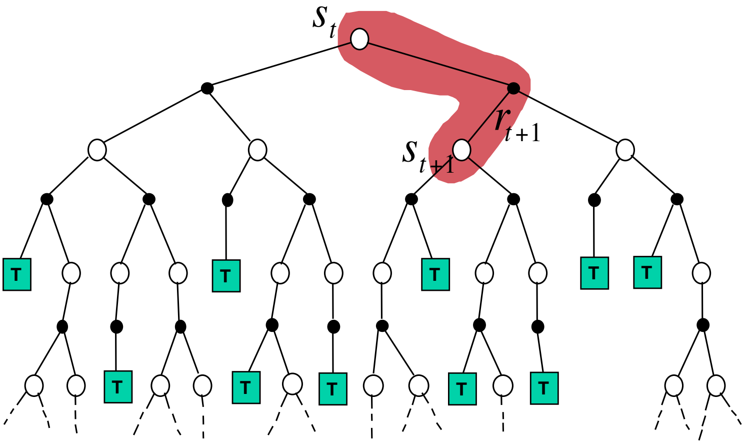
Monte-Carlo Backup

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



Temporal-Difference Backup

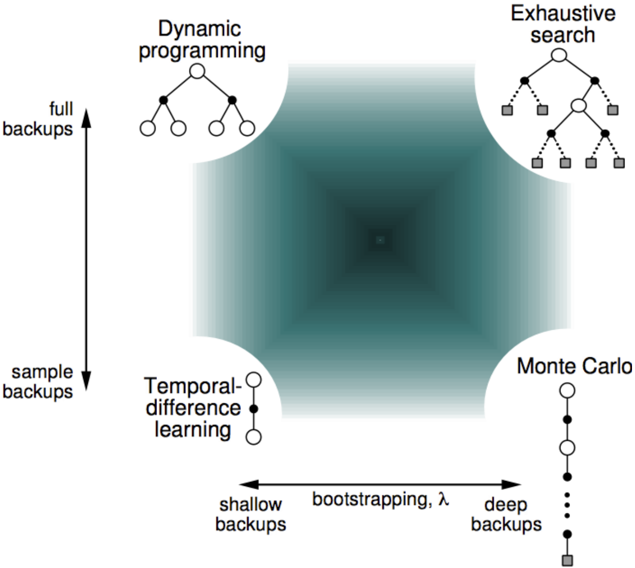
$$v(S_t) \leftarrow v(S_t) + \alpha \left(R_{t+1} + \gamma v(S_{t+1}) - v(S_t) \right)$$



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

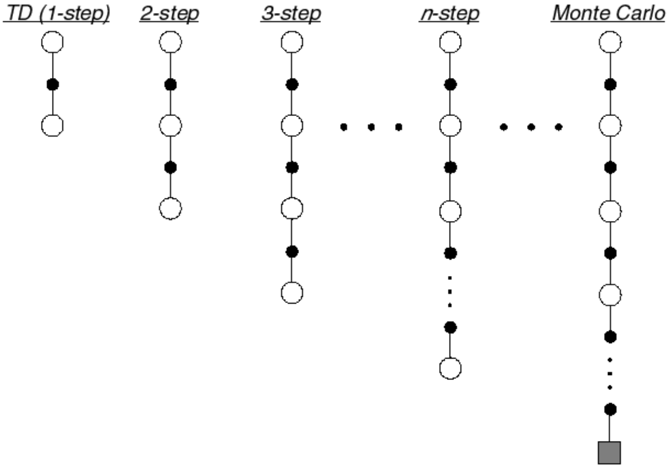


Multi-step updates

- When we bootstrap, updates use old estimates
- Information can propagate back quite slowly
- In MC information propagates faster, but the updates are noisier
- We can go in between TD and MC

n-Step Prediction

- Let TD target look n steps into the future



n-Step Return

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

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

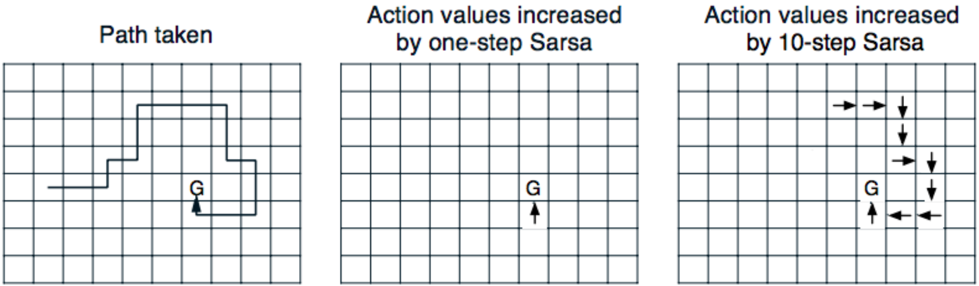
- In general, the n-step return is defined by

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

- Multi-step temporal-difference learning

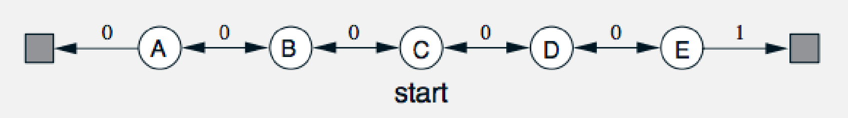
$$v(S_t) \leftarrow v(S_t) + \alpha \left(G_t^{(n)} - v(S_t) \right)$$

Multi-step Return



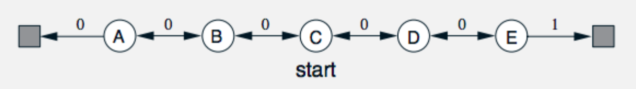
(SARSA is TD for action values $q(s, a)$ — more on that later)

Random Walk Example

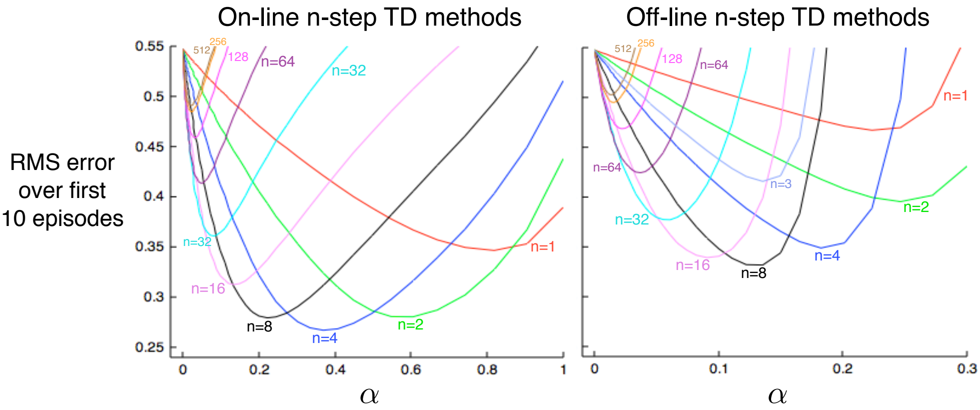


- Uniform random transitions (50% left, 50% right)
- Initial values are $v(s) = 0.5$, for all s
- True values happen to be
  $v(A) = \frac{1}{6}$, $v(B) = \frac{2}{6}$, $v(C) = \frac{3}{6}$, $v(D) = \frac{4}{6}$, $v(E) = \frac{5}{6}$

Large Random Walk Example



..., but with 19 states, rather than 5



Benefits of multi-step returns

- Multi-step returns have benefits from both TD and MC
- Bootstrapping can have issues with **bias**
- Monte Carlo can have issues with **variance**
- Typically, intermediate values of n are good

Independence of temporal span

- MC and multi-step returns are not **independent of span**:
  To update values in a long episode, you have to wait
- TD can update immediately, and is independent of span
- Can we get both?

Eligibility traces

- Consider $\boldsymbol{v}$ to be a vector with components v_s
- Let $\boldsymbol{x}_s$ be a one-hot vector: $\boldsymbol{x}_s = [0, 0, \ldots, 0, 1, 0, \ldots, 0, 0]^\top$
  with the s^{th} element equal to one: $\boldsymbol{x}_s[s] = 1$ and $\boldsymbol{x}_s[s'] = 0$ for all $s' \neq s$
- The Monte Carlo update to $\boldsymbol{v}$ for a state S_t is then

$$\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + \alpha(G_t - v(S_t))\boldsymbol{x}_{S_t} \,.$$

Note, this only updates the relevant entry in $\boldsymbol{v}$, as intended.
Note, this is the same update as before, just written differently

Eligibility traces

We can rewrite the MC error as a sum of TD errors:

$$
\begin{aligned}
\delta_t^{\mathrm{MC}} &= G_t - v(S_t) \\
&= R_{t+1} + \gamma G_{t+1} - v(S_t) \\
&= \underbrace{R_{t+1} + \gamma v(S_{t+1}) - v(S_t)}_{=\delta_t} + \gamma \underbrace{(G_{t+1} - v(S_{t+1}))}_{=\delta_{t+1}^{\mathrm{MC}}} \\
&= \delta_t + \gamma \delta_{t+1}^{\mathrm{MC}} \\
&= \delta_t + \gamma \delta_{t+1} + \gamma^2 \delta_{t+2}^{\mathrm{MC}} \\
&= \dots \\
&= \sum_{k=0}^{T-t} \gamma^k \delta_{t+k} \,.
\end{aligned}
$$

$$
\left(= \sum_{k=t}^{T} \gamma^{k-t} \delta_k \right)
$$

(used in the next slide)

Eligibility traces

- Now consider accumulating a whole episode (from time $t = 0$ to T) of updates:

$$
\begin{aligned}
\Delta \boldsymbol{v} &= \sum_{t=0}^{T} \alpha (G_t - v(S_t)) \boldsymbol{x}_{S_t} \\
&= \sum_{t=0}^{T} \alpha \sum_{k=t}^{T} \gamma^{k-t} \delta_k \boldsymbol{x}_{S_t} \\
&= \sum_{k=0}^{T} \alpha \sum_{t=0}^{k} \gamma^{k-t} \delta_k \boldsymbol{x}_{S_t} && \left(\text{using } \sum_{i=0}^{m} \sum_{j=i}^{m} z_{ij} = \sum_{j=0}^{m} \sum_{i=0}^{j} z_{ij}\right) \\
&= \sum_{k=0}^{T} \alpha \delta_k \underbrace{\sum_{t=0}^{k} \gamma^{k-t} \boldsymbol{x}_{S_t}}_{= \mathbf{e}_k} && = \sum_{t=0}^{T} \alpha \delta_t \mathbf{e}_t \,.
\end{aligned}
$$

Eligibility traces

$$\begin{array}{cccc}
\delta_1 \boldsymbol{x}_1 & \delta_2 \boldsymbol{x}_1 & \delta_3 \boldsymbol{x}_1 & \delta_4 \boldsymbol{x}_1 \\
 & \delta_2 \boldsymbol{x}_2 & \delta_3 \boldsymbol{x}_2 & \delta_4 \boldsymbol{x}_2 \\
 & & \delta_3 \boldsymbol{x}_3 & \delta_4 \boldsymbol{x}_3 \\
 & & & \delta_4 \boldsymbol{x}_4
\end{array}$$

Eligibility traces

$$\begin{array}{cccc} \delta_1 \boldsymbol{x}_1 & \delta_2 \boldsymbol{x}_1 & \delta_3 \boldsymbol{x}_1 & \delta_4 \boldsymbol{x}_1 \\ & \delta_2 \boldsymbol{x}_2 & \delta_3 \boldsymbol{x}_2 & \delta_4 \boldsymbol{x}_2 \\ & & \delta_3 \boldsymbol{x}_3 & \delta_4 \boldsymbol{x}_3 \\ & & & \delta_4 \boldsymbol{x}_4 \end{array}$$

$$= \sum_{k=2}^{4} \delta_k \boldsymbol{x}_2 = (G_2 - v(S_2))\boldsymbol{x}_2$$

Eligibility traces

$$
\begin{array}{cccccc}
 & & = \delta_3 \mathbf{e}_3 & & & \\
\delta_1 \boldsymbol{x}_1 & \delta_2 \boldsymbol{x}_1 & \delta_3 \boldsymbol{x}_1 & \delta_4 \boldsymbol{x}_1 & & \\
 & \delta_2 \boldsymbol{x}_2 & \delta_3 \boldsymbol{x}_2 & \delta_4 \boldsymbol{x}_2 & = (G_2 - v(S_2))\boldsymbol{x}_2 & \\
 & & \delta_3 \boldsymbol{x}_3 & \delta_4 \boldsymbol{x}_3 & & \\
 & & & \delta_4 \boldsymbol{x}_4 & &
\end{array}
$$

Eligibility traces

- Accumulating a whole episode of updates:

$$
\begin{aligned}
\Delta \boldsymbol{v} &= \sum_{t=0}^{T} \alpha \delta_t \mathbf{e}_t \\
\text{where} \qquad \mathbf{e}_t &= \sum_{j=0}^{t} \gamma^{t-j} \boldsymbol{x}_{S_j} \\
&= \gamma \sum_{j=0}^{t-1} \gamma^{t-1-j} \boldsymbol{x}_{S_j} + \boldsymbol{x}_{S_t} \\
&= \gamma \mathbf{e}_{t-1} + \boldsymbol{x}_{S_t} \,.
\end{aligned}
$$

This is called an **eligibility trace**
Every step, it decays (according to γ) and then the current one-hot is added

Eligibility traces

- Accumulating a whole episode of updates:

$$\Delta_t \boldsymbol{v} \equiv \alpha \delta_t \mathbf{e}_t$$
$$\Delta \boldsymbol{v} = \sum_{t=0}^{T} \Delta_t \boldsymbol{v}$$
$$\text{where} \qquad \mathbf{e}_t = \gamma \mathbf{e}_{t-1} + \boldsymbol{x}_{S_t} \,.$$

  (And then apply $\Delta \boldsymbol{v}$ at the end of the episode)
- Intuition: decay the eligibility of past states for the current TD error
- Intuition: the same TD error shows up in multiple MC errors—we grouped them so that we can applied it to all past states in one update
- This is kind of magical: we can reupdate all past states (to account for the new TD error) with a single update! No need to even recompute their values
  (You can't always do that, but here it works out)

Mixing multi-step returns

- Multi-step returns bootstrap on one state, $v(S_{t+n})$:

$$G_t^{(n)} = R_{t+1} + \gamma G_{t+1}^{(n-1)} \qquad \text{(while } n > 1\text{, continue)}$$
$$G_t^{(1)} = R_{t+1} + \gamma v(S_t)\,. \qquad \text{(truncate \& bootstrap)}$$

- You can also bootstrap a little bit on multiple states:

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

This turns out to be a weighted average of n-step returns:

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

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

Mixing multi-step returns

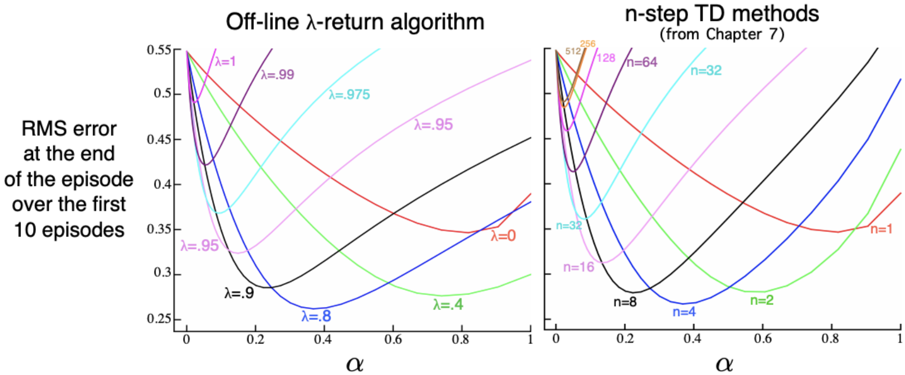
$$G_t^\lambda = R_{t+1} + \gamma \Big((1-\lambda) v(S_{t+1}) + \lambda G_{t+1}^\lambda \Big)$$

Special cases:

$$G_t^{\lambda=0} = R_{t+1} + \gamma v(S_{t+1}) \tag{TD}$$

$$G_t^{\lambda=1} = R_{t+1} + \gamma G_{t+1} \tag{MC}$$

Mixing multi-step returns



Intuition: $1/(1-\lambda)$ is the 'horizon'. E.g., $\lambda = 0.9 \approx n = 10$.

Mixing multi-step returns & traces

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

- The associated error and trace update are

$$G_t^\lambda = \sum_{k=0}^{T-t} \lambda^k \gamma^k \delta_{t+k} \qquad \text{(same as before, but with } \lambda\gamma \text{ instead of } \gamma\text{)}$$

$$\mathbf{e}_t = \gamma\lambda \mathbf{e}_{t-1} + \boldsymbol{x}_{S_t} \qquad \text{and} \qquad \Delta \boldsymbol{v}_t = \alpha \delta_t \mathbf{e}_t\,.$$

- This is called an **accumulating trace** with decay $\gamma\lambda$ (instead of γ for MC, or 0 for TD)
- It is exact for batched episodic updates, similar traces exist for online updating
- We can also derive traces for function approximation (in a later lecture)

Next lecture

Model-free control