Lecture 5: Model-Free Prediction

Hado van Hasselt

UCL, 2021

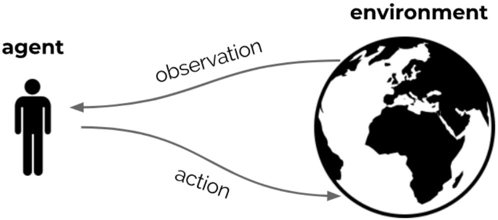
Background

Sutton & Barto 2018, Chapters 5 + 6 + 7 + 9 +12

Don’t worry about reading all of this at once!
Most important chapters, for now: 5 + 6
You can also defer some reading, e.g., until the reading week

Don’t forget to pause

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**

Lecture overview

- Last lectures (3+4):
  - **Planning** by **dynamic programming** to solve a known MDP
- This and next lectures (5→8):
  - **Model-free prediction** to **estimate** values in an **unknown** MDP
  - **Model-free control** to **optimise** values in an **unknown** MDP
  - **Function approximation** and (some) **deep reinforcement learning** (but more to follow later)
  - **Off-policy** learning
- Later lectures:
  - Model-based learning and planning
  - Policy gradients and actor critic systems
  - More deep reinforcement learning
  - More advanced topics and current research

Model-Free Prediction: Monte Carlo Algorithms

Monte Carlo Algorithms

- 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

Monte Carlo: Bandits

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

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

  - Equivalently:

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

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

no multi-arm vs RL

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

Monte Carlo: Bandits with States

- Consider bandits with different states
  - episodes are still one step
  - actions do not affect state transitions
  - $\implies$ no long-term consequences
- Then, we want to estimate

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

- These are called **contextual bandits**

$\rightsquigarrow$ multiple step this time.

Introduction Function Approximation

Value Function Approximation

- So far we mostly considered **lookup tables**
  - Every state s has an entry $v(s)$
  - Or every state-action pair s, a has an entry $q(s, a)$
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
  - Individual states are often **not fully observable**

Value Function Approximation

Solution for large MDPs:

- Estimate value function with **function approximation**

$$v_{\mathbf{w}}(s) \approx v_\pi(s) \qquad \text{(or } v_*(s)\text{)}$$
$$q_{\mathbf{w}}(s,a) \approx q_\pi(s,a) \qquad \text{(or } q_*(s,a)\text{)}$$

- Update parameter **w** (e.g., using MC or TD learning)
- Generalise from to unseen states

temporal difference.

Agent state update

Solution for large MDPs, if the environment state is not fully observable

- Use the **agent state**:

$$S_t = u_\omega(S_{t-1}, A_{t-1}, O_t)$$

reward part of this (or could be separate)

  with parameters ω (typically $\omega \in \mathbb{R}^n$)

- Henceforth, S_t denotes the agent state
- Think of this as either a vector inside the agent,
  or, in the simplest case, just the current observation: $S_t = O_t$
- For now we are **not** going to talk about how to learn the agent state update
- Feel free to consider S_t an observation

Linear Function Approximation

Feature Vectors

- A useful special case: **linear functions**
- Represent state by a **feature vector**

$$\mathbf{x}(s) = \begin{pmatrix} x_1(s) \\ \vdots \\ x_m(s) \end{pmatrix}$$

- $\mathbf{x} : \mathcal{S} \to \mathbb{R}^m$ is a fixed mapping from agent state (e.g., observation) to features
- Short-hand: $\mathbf{x}_t = \mathbf{x}(S_t)$
- For example:
  - Distance of robot from landmarks
  - Trends in the stock market
  - Piece and pawn configurations in chess

Linear Value Function Approximation

- Approximate value function by a linear combination of features

dot product

$$v_{\mathbf{w}}(s) = \mathbf{w}^\top \mathbf{x}(s) = \sum_{j=1}^{n} x_j(s) w_j$$

- Objective function ('loss') is quadratic in $\mathbf{w}$

we don't have this.

$$L(\mathbf{w}) = \mathbb{E}_{S \sim d}[(v_\pi(S) - \mathbf{w}^\top \mathbf{x}(S))^2]$$

- Stochastic gradient descent converges on global optimum

because "convex"

- Update rule is simple

$$\nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t) = \mathbf{x}(S_t) = \mathbf{x}_t \qquad \Longrightarrow \qquad \Delta \mathbf{w} = \alpha(v_\pi(S_t) - v_{\mathbf{w}}(S_t))\mathbf{x}_t$$

Update = **step-size** × **prediction error** × **feature vector**

Table Lookup Features

- Table lookup is a special case of linear value function approximation
- Let the n states be given by $\mathcal{S} = \{s_1, \ldots, s_n\}$.
- Use one-hot feature:

$\rightsquigarrow \delta$ (0 if not $s = s_1$)

$$\mathbf{x}(s) = \begin{pmatrix} \mathcal{I}(s = s_1) \\ \vdots \\ \mathcal{I}(s = s^n) \end{pmatrix}$$

- Parameters $\mathbf{w}$ then just contains value estimates for each state

$$v(s) = \mathbf{w}^\top \mathbf{x}(s) = \sum_j w_j x_j(s) = w_s \, .$$

Model-Free Prediction: Monte Carlo Algorithms

(Continuing from before...)

Monte Carlo: Bandits with States

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

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

- Then the gradient update is

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

  We can sample this to get a **stochastic gradient update** (SGD) → wtf is this.
- The tabular case is a special case (only updates the value in cell $[S_t, A_t]$)
- Also works for large (continuous) state spaces $\mathcal{S}$ — this is just **regression**

Monte Carlo: Bandits with States

- When using linear functions, $q(s, a) = \mathbf{w}^\top \mathbf{x}(s, a)$ and

$$\nabla_{\mathbf{w}_t} q_{\mathbf{w}_t}(S_t, A_t) = \mathbf{x}(s, a)$$

- Then the SGD update is

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha(R_{t+1} - q_{\mathbf{w}_t}(S_t, A_t))\mathbf{x}(s, a) .$$

- Linear update = **step-size** × **prediction error** × **feature vector**
- Non-linear update = **step-size** × **prediction error** × **gradient**

Monte-Carlo Policy Evaluation

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

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

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

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

- The value function is the expected return:

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

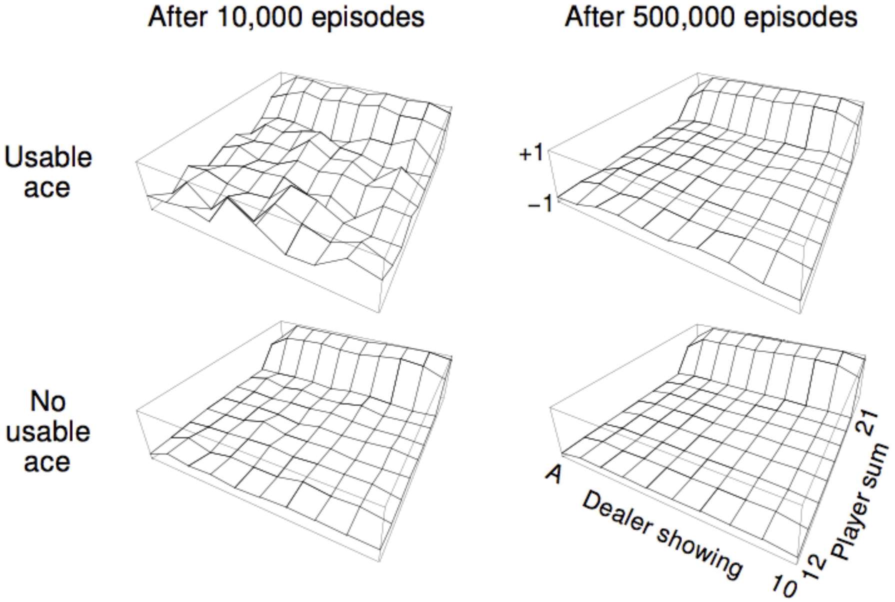
- We can just use **sample average** return instead of **expected** return
- We call this **Monte Carlo policy evaluation**

Example: Blackjack

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



Disadvantages of Monte-Carlo Learning

- We have seen MC algorithms can be used to learn value predictions
- But when episodes are long, learning can be slow
  - ...we have to wait until an episode ends before we can learn
  - ...return can have high variance
- Are there alternatives? (Spoiler: yes)

Temporal-Difference Learning

important

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! → estimate of v_π

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

- This is likely quite noisy — better to take a small step (with parameter α):

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

(Note: tabular update)

Temporal difference learning

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

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

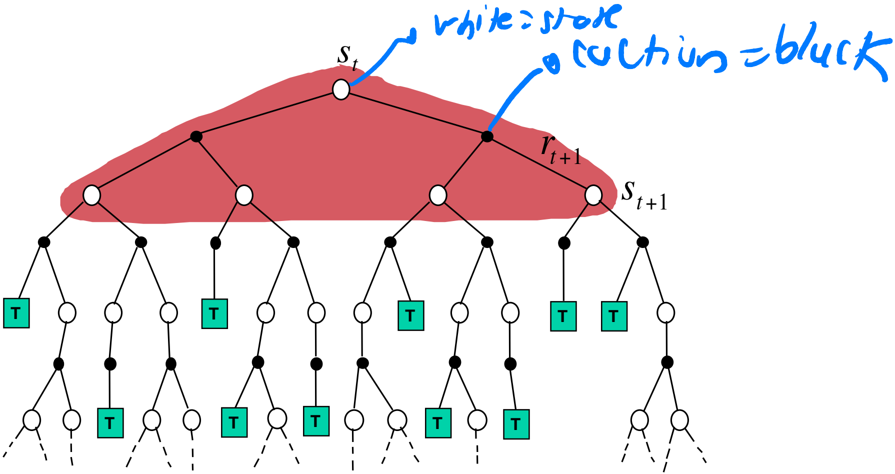
- Temporal-difference learning:
  - Update value $v_t(S_t)$ towards estimated return $\boldsymbol{R_{t+1} + \gamma v(S_{t+1})}$

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

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

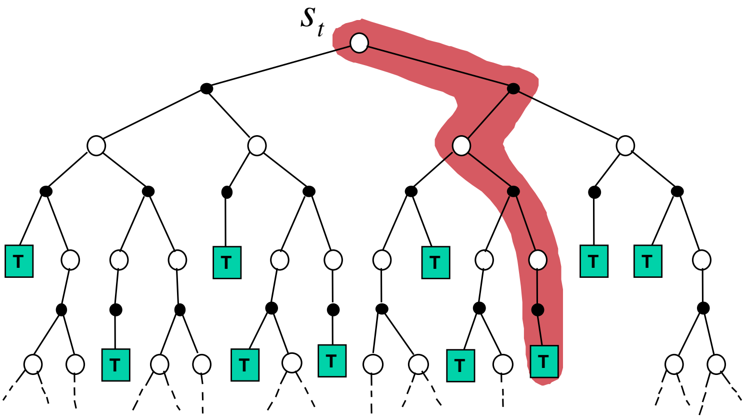
Dynamic Programming Backup

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



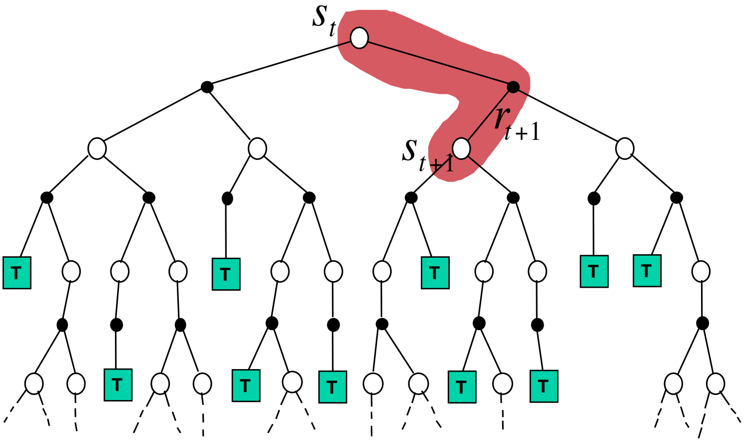
Monte-Carlo Backup

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



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

Temporal difference learning

- We can apply the same idea to **action values**
- Temporal-difference learning for action values:
  - Update value $q_t(S_t, A_t)$ towards estimated return $\boldsymbol{R_{t+1} + \gamma q(S_{t+1}, A_{t+1})}$

$$q_{t+1}(S_t, A_t) \leftarrow q_t(S_t, A_t) + \alpha \left(\overbrace{\underbrace{R_{t+1} + \gamma q_t(S_{t+1}, A_{t+1})}_{\text{target}} - q_t(S_t, A_t)}^{\text{TD error}} \right)$$

  - This algorithm is known as **SARSA**, because it uses $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$

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

→ don't have to wait until end of episode.

Example: Driving Home

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

reward

+15

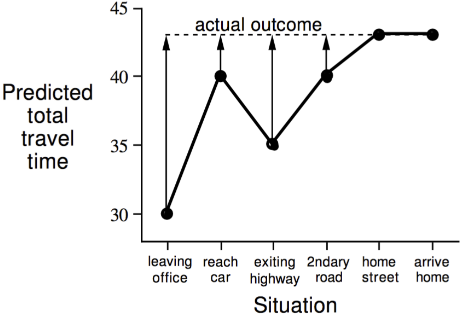
+10

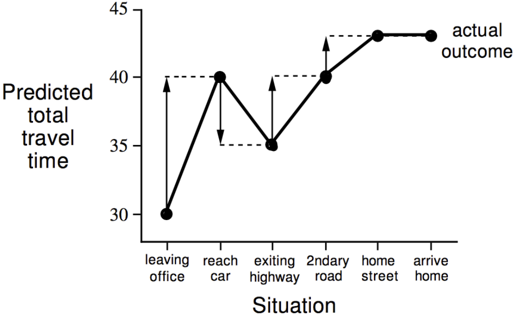
+10

Driving Home Example: MC vs. TD

Changes recommended by Monte Carlo methods (α=1)

Changes recommended by TD methods (α=1)





Comparing MC and 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
- TD is **independent of the temporal span** of the prediction
  - TD can learn from single transitions ↳ constant on every time-step.
  - MC must store all predictions (or states) to update at the end of an episode ↳ memory required grows if episode is longer
- TD needs reasonable value estimates

Bias/Variance Trade-Off

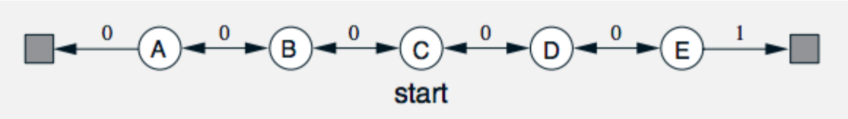
- MC return $G_t = R_{t+1} + \gamma R_{t+2} + \ldots$ is an **unbiased** estimate of $v_\pi(S_t)$
- TD target $R_{t+1} + \gamma v_t(S_{t+1})$ is a **biased** estimate of $v_\pi(S_t)$ (unless $v_t(S_{t+1}) = v_\pi(S_{t+1})$)
- But the TD target has **lower variance**:
  - Return depends on **many** random 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 → may not fit states.
- In the tabular case, both MC and TD will converge: $v_t \longrightarrow v_\pi$

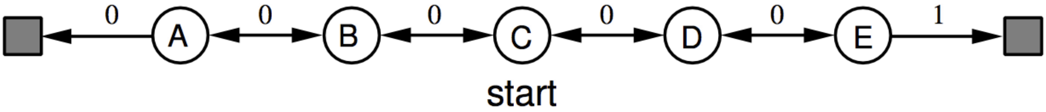
Example: Random Walk

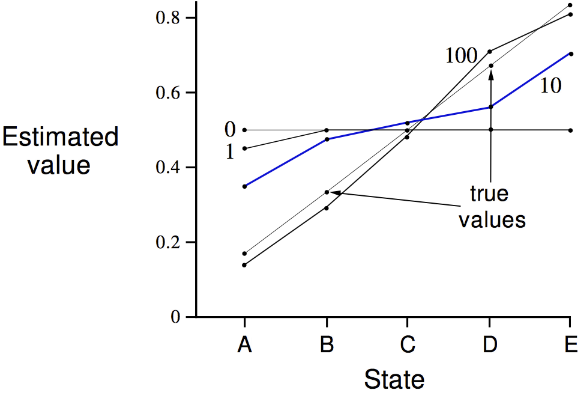
Random Walk Example



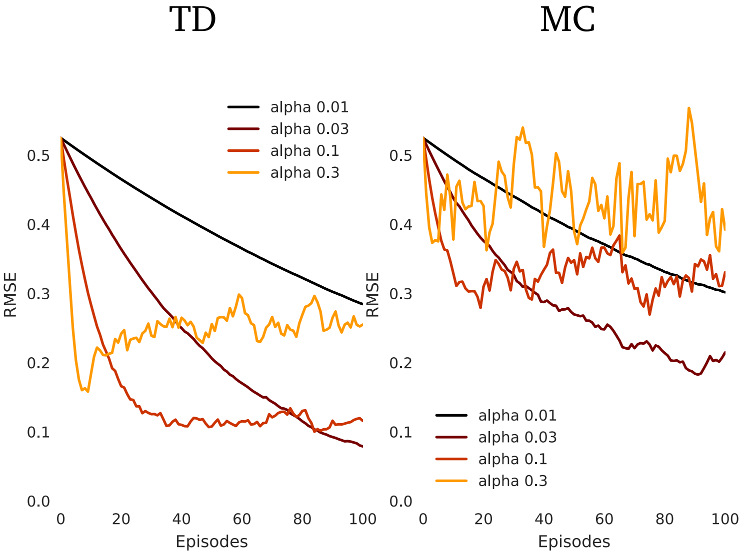
- Uniform random transitions (50% left, 50% right)
- Initial values are $v(s) = 0.5$, for all s
- True values happen to be
  $v(A) = \frac{1}{6}, v(B) = \frac{2}{6}, v(C) = \frac{3}{6}, v(D) = \frac{4}{6}, v(E) = \frac{5}{6}$

Random Walk Example





Random Walk: MC vs. TD



Batch MC and TD

Batch MC and TD

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

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

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

Example:
Batch Learning in Two States

Example: Batch Learning in Two States

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

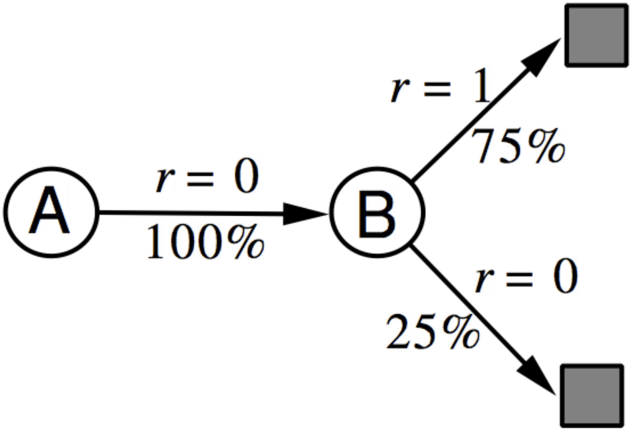
terminates

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

Example: Batch Learning in Two States

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

Differences in batch solutions

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

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

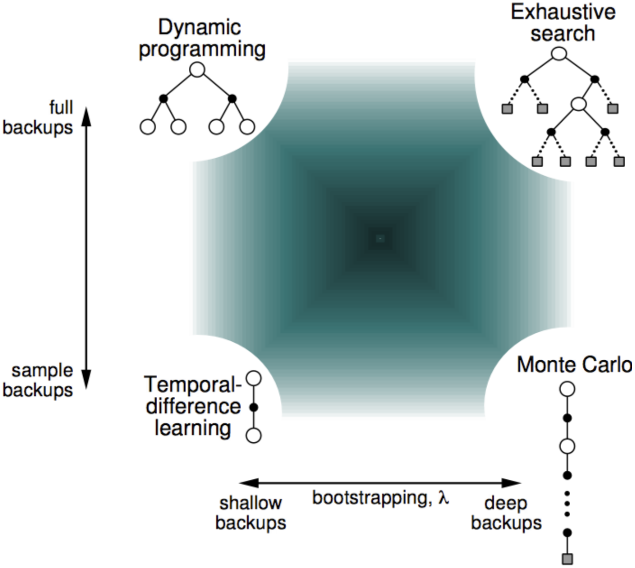
- In the AB example, $\boldsymbol{v(A) = 0}$
- TD converges to solution of max likelihood Markov model, given the data
  - Solution to the empirical MDP $(\mathcal{S}, \mathcal{A}, \hat{p}, \gamma)$ that best fits the data
  - In the AB example: $\hat{p}(S_{t+1} = B \mid S_t = A) = 1$, and therefore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

Advantages and Disadvantages of MC vs. TD

- TD exploits Markov property
  - Can help in fully-observable environments
- MC does not exploit Markov property
  - Can help in partially-observable environments
- With finite data, or with function approximation, **the solutions may differ**

Between MC and TD: Multi-Step TD

Unified View of Reinforcement Learning



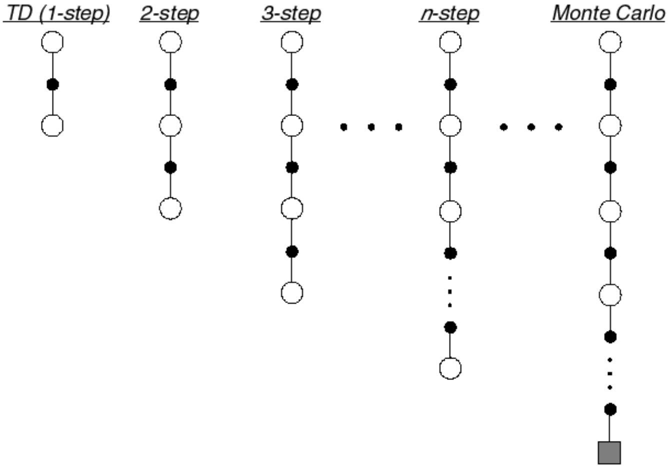
Multi-Step Updates

- TD uses value estimates which might be inaccurate
- In addition, information can propagate back quite slowly
- In MC information propagates faster, but the updates are noisier
- We can go in between TD and MC

because propagates only 1 state in front.

Multi-Step Prediction

- Let TD target look n steps into the future



Multi-Step Returns

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

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

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

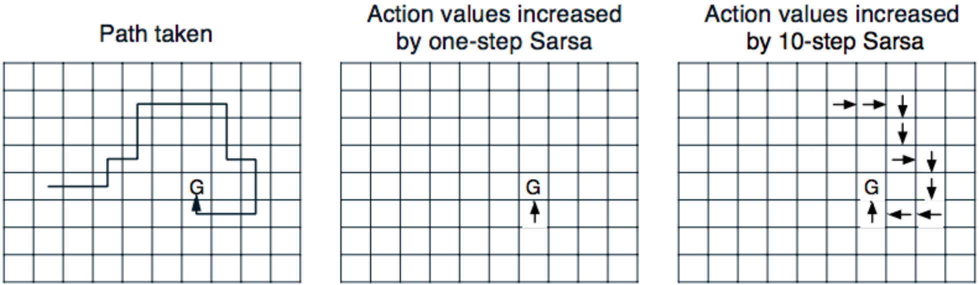
$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n v(S_{t+n})$$

- Multi-step temporal-difference learning

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

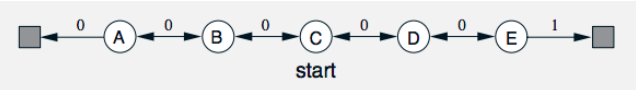
Multi-Step Examples

Grid Example

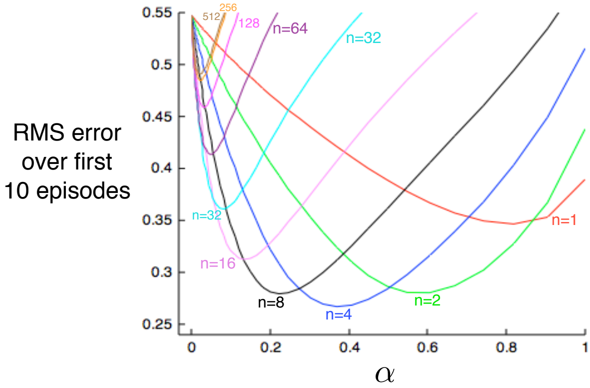


(Reminder: SARSA is TD for action values $q(s, a)$)

Large Random Walk Example



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



Mixed Multi-Step Returns

Mixing multi-step returns

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

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

- 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 gives 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

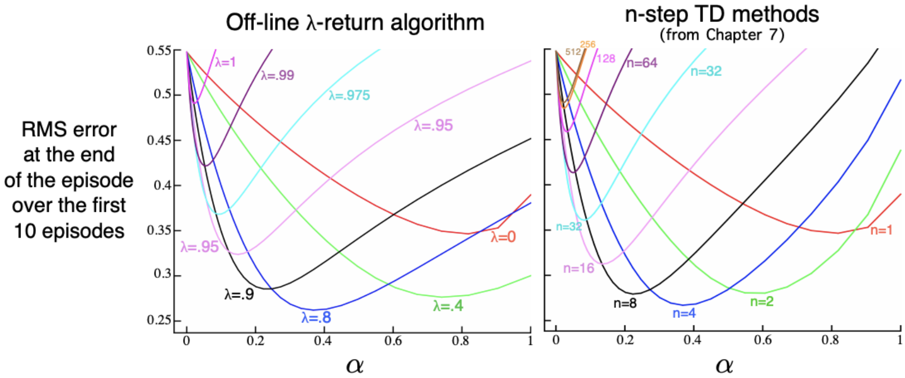
$$G_t^\lambda = R_{t+1} + \gamma\left((1-\lambda)v(S_{t+1}) + \lambda G_{t+1}^\lambda\right)$$

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., $\lambda = 0.9 \approx n = 10$.

Benefits of Multi-Step Learning

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 or λ are good (e.g., $n = 10$, $\lambda = 0.9$)

Eligibility Traces

↳ more advanced.

Independence of temporal span

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

Eligibility traces

- Recall **linear function approximation**
- The Monte Carlo and TD updates to $v_{\mathbf{w}}(s) = \mathbf{w}^\top \mathbf{x}(s)$ for a state $s = S_t$ is

$$\Delta \mathbf{w}_t = \alpha(G_t - v(S_t))\mathbf{x}_t \tag{MC}$$
$$\Delta \mathbf{w}_t = \alpha(R_{t+1} + \gamma v(S_{t+1}) - v(S_t))\mathbf{x}_t \tag{TD}$$

- MC updates all states in episode k at once:

$$\Delta \mathbf{w}_{k+1} = \sum_{t=0}^{T-1} \alpha(G_t - v(S_t))\mathbf{x}_t$$

where $t \in \{0, \ldots, T-1\}$ enumerate the time steps in this specific episode

- Recall: tabular is a special case, with one-hot vector $\mathbf{x}_t$

Eligibility traces

one step TD error

- Accumulating a whole episode of updates:

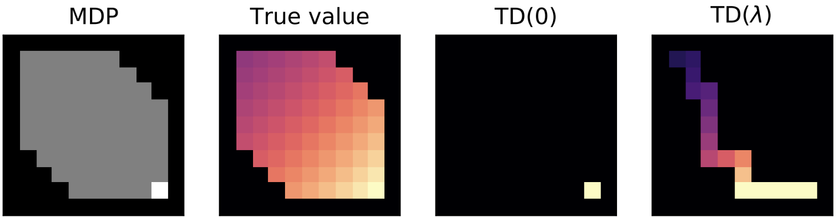
$$\Delta \mathbf{w}_t \equiv \alpha \delta_t \mathbf{e}_t \qquad \text{(one time step)}$$

$$\text{where} \qquad \mathbf{e}_t = \gamma \lambda \mathbf{e}_{t-1} + \boldsymbol{x}_t$$

decaying it.

- Note: if $\lambda = 0$, we get one-step TD
- Intuition: decay the **eligibility** of past states for the current TD error, then add it
- This is kind of magical: we can update **all past states** (to account for the new TD error) with a single update! No need to recompute their values.
- This idea extends to function approximation: $\mathbf{x}_t$ does not have to be one-hot

Eligibility traces



Eligibility traces

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

$$\begin{aligned}
G_t - v(S_t) &= R_{t+1} + \gamma G_{t+1} - v(S_t) \\
&= \underbrace{R_{t+1} + \gamma v(S_{t+1}) - v(S_t)}_{= \delta_t} + \gamma(G_{t+1} - v(S_{t+1})) \\
&= \delta_t + \gamma(G_{t+1} - v(S_{t+1})) \\
&= \ldots \\
&= \delta_t + \gamma \delta_{t+1} + \gamma^2(G_{t+2} - v(S_{t+2})) \\
&= \ldots \\
&= \sum_{k=t}^{T} \gamma^{k-t} \delta_k \qquad \text{(used in the next slide)}
\end{aligned}$$

Eligibility traces

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

$$\begin{aligned}
\Delta \mathbf{w}_k &= \sum_{t=0}^{T-1} \alpha (G_t - v(S_t)) \boldsymbol{x}_t \\
&= \sum_{t=0}^{T-1} \alpha \left(\sum_{k=t}^{T-1} \gamma^{k-t} \delta_k \right) \boldsymbol{x}_t && \text{(Using result from previous slide)} \\
&= \sum_{k=0}^{T-1} \alpha \sum_{t=0}^{k} \gamma^{k-t} \delta_k \boldsymbol{x}_t && \left(\text{Using } \sum_{i=0}^{m} \sum_{j=i}^{m} z_{ij} = \sum_{j=0}^{m} \sum_{i=0}^{j} z_{ij}\right) \\
&= \sum_{k=0}^{T-1} \alpha \delta_k \underbrace{\sum_{t=0}^{k} \gamma^{k-t} \boldsymbol{x}_t}_{\equiv \mathbf{e}_k} && = \sum_{k=0}^{T-1} \alpha \delta_k \mathbf{e}_k = \underbrace{\sum_{t=0}^{T-1} \alpha \delta_t \mathbf{e}_t}_{\substack{\text{renaming} \\ k \longrightarrow t}} \, .
\end{aligned}$$

Eligibility traces

Accumulating a whole episode of updates:

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

The vector $\mathbf{e}_t$ is called an **eligibility trace**
Every step, it decays (according to γ) and then the current feature $\mathbf{x}_t$ is added

Eligibility traces

- Accumulating a whole episode of updates:

$$\Delta \mathbf{w}_t \equiv \alpha \delta_t \mathbf{e}_t \qquad \text{(one time step)}$$

$$\Delta \mathbf{w}_k = \sum_{t=0}^{T-1} \Delta \mathbf{w}_t \qquad \text{(whole episode)}$$

$$\text{where} \qquad \mathbf{e}_t = \gamma \mathbf{e}_{t-1} + \boldsymbol{x}_t \; .$$

  (And then apply $\Delta \mathbf{w}$ at the end of the episode)

- Intuition: the same TD error shows up in multiple MC errors—grouping them allows applying it to all past states in one update

Eligibility Traces: Intuition

Eligibility traces

Consider a batch update on an episode with four steps: $t \in \{0, 1, 2, 3\}$

$$
\begin{array}{rrrrr}
\Delta \mathbf{v} = & \delta_0 \mathbf{e}_0 & \delta_1 \mathbf{e}_1 & \delta_2 \mathbf{e}_2 & \delta_3 \mathbf{e}_3 \\
(G_0 - v(S_0))\mathbf{x}_0 & \delta_0 \mathbf{x}_0 & \gamma \delta_1 \mathbf{x}_0 & \gamma^2 \delta_2 \mathbf{x}_0 & \gamma^3 \delta_3 \mathbf{x}_0 \\
(G_1 - v(S_1))\mathbf{x}_1 & & \delta_1 \mathbf{x}_1 & \gamma \delta_2 \mathbf{x}_1 & \gamma^2 \delta_3 \mathbf{x}_1 \\
(G_2 - v(S_2))\mathbf{x}_2 & & & \delta_2 \mathbf{x}_2 & \gamma \delta_3 \mathbf{x}_2 \\
(G_3 - v(S_3))\mathbf{x}_3 & & & & \delta_3 \mathbf{x}_3
\end{array}
$$

Mixed Multi-Step Returns and Eligibility Traces

Mixing multi-step returns & traces

- Reminder: mixed multi-step return

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

- The associated error and trace update are

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

$$\implies \quad \mathbf{e}_t = \gamma\lambda\mathbf{e}_{t-1} + \mathbf{x}_t \qquad \text{and} \qquad \Delta\mathbf{w}_t = \alpha\delta_t\mathbf{e}_t \ .$$

- This is called an **accumulating trace** with decay $\gamma\lambda$
- It is exact for batched episodic updates ('offline'), similar traces exist for online updating

End of Lecture

Next lecture:
Model-free control