Applied Computational Mathematics Lab

TEDDY LAZEBNIK'S COURSE · LECTURE 5

Reinforcement Learning

Lecture 5 · Model-free prediction: Monte Carlo, a first look at function approximation, temporal-difference learning, multi-step returns and eligibility traces



Teddy Lazebnik · Applied Computational Mathematics Laboratory

BEFORE WE START

Recap, and what this lecture does

- RL is the science of learning to make decisions. Agents can learn a policy, a value function and/or a model.
- Decisions affect the reward, the agent state and the environment state.
- Lectures 3 and 4: **planning** by dynamic programming, in a **known** MDP.

LECTURES 5–8

Without a model

Model-free prediction (today) and control; function approximation; off-policy learning.

LATER

Beyond

Model-based learning and planning; policy gradients and actor-critic; more deep RL.



IN PLAIN WORDS

For two lectures, the agent knew every probability and every reward. Today we take that away. No model: the agent only gets to play.

THE GOAL

Sample the world forever and you'd know it exactly. In practice we have a finite pile of experience, so we need methods that learn well from it. Today's goal is **prediction**: given a policy, learn its values from samples.

BEFORE WE START

Today's map

PART 1

Monte Carlo

Play, see what happened, average. From bandits to whole episodes.

PART 2

A first look at function approximation

Features, linear values, and why a table is a special case.

PART 3

Temporal-difference learning

Learn a guess from a guess, after every step.

PART 4

Comparing MC and TD

Bias and variance, a random walk race, and batch learning.

PART 5

Multi-step returns and traces

Between TD and MC: n-step, the λ-return, eligibility traces.

READING

Sutton & Barto (2018)

Chapters 5, 6, 7, 9 and 12. The most important for now: 5 and 6.



IN PLAIN WORDS

By the end you should be able to compute Monte Carlo and TD updates by hand, explain when each is better, and say what λ and eligibility traces do.

Applied Computational Mathematics Lab

PART 1

Monte Carlo

Play. See what happened. Average.



Learning from samples

- We can use experience samples to learn **without a model**.
- Directly sampling whole episodes and averaging what happened is called **Monte Carlo**.
- Monte Carlo is model-free: no knowledge of the MDP is required, only samples.

TRY IT

Your commute. You don’t know the traffic model, but you time the trip: 31, 27, 35 and 29 minutes. Your estimate of the expected time is the average, 30.5 minutes. That is a Monte Carlo estimate.



IN PLAIN WORDS

The name comes from the casino in Monaco: rolling dice many, many times. That is the whole method.

WHY IT WORKS

The law of large numbers (Lecture 2): the average of independent samples converges to the expected value. No probabilities needed, just enough samples.

Monte Carlo for bandits

- A multi-armed bandit: for each action, average the reward samples:

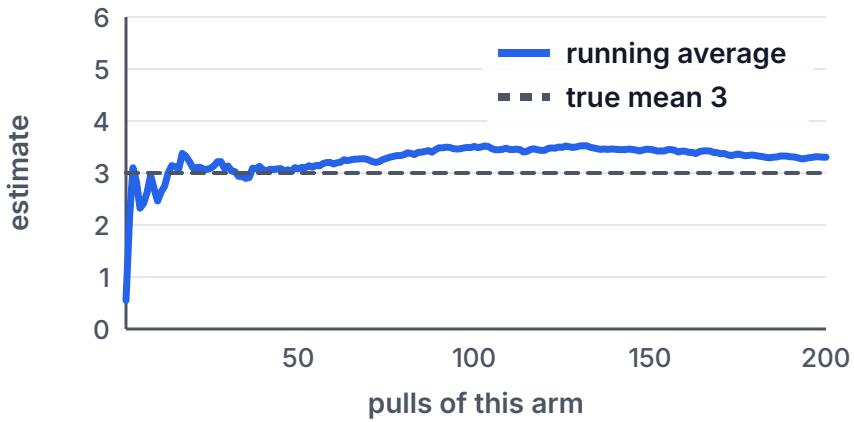
$$q_t(a) = \frac{\sum_{i=0}^{t} \mathbb{I}(A_i = a)\, R_{i+1}}{\sum_{i=0}^{t} \mathbb{I}(A_i = a)} \;\approx\; \mathbb{E}[R_{t+1} \mid A_t = a] = q(a)$$

- Equivalently, incrementally, with $\alpha_t = 1/N_t(A_t)$:

$$q_{t+1}(A_t) = q_t(A_t) + \alpha_t\big(R_{t+1} - q_t(A_t)\big), \qquad q_{t+1}(a) = q_t(a) \;\; \forall a \neq A_t$$

NOTATION

From now on the reward after action A_t is called R_{t+1}: in an MDP it arrives on the next time step.



Our run: rewards with mean 3 and standard deviation 2. The average wobbles early, then settles.

TRY IT

One arm pays 4, 0, 2, 6.
Averages after each pull: 4, 2, 2, 3.
Incremental: $4 + \frac{1}{2}(0 - 4) = 2$, then $2 + \frac{1}{3}(2 - 2) = 2$, then $2 + \frac{1}{4}(6 - 2) = 3$.

Bandits with states

- Now give the bandit different states:
  - episodes still last one step;
  - actions do not affect the next state;
  - so there are no long-term consequences.
- We want to estimate

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

- These are called **contextual bandits**. Each state is its own little bandit.

USER (STATE)	AD 1	AD 2	AD 3
student, morning	**0.08**	0.03	0.05
student, evening	0.02	**0.07**	0.04
retired, morning	0.01	0.02	**0.06**

Showing ads: the user is the state, the ad is the action, a click is the reward. Estimated click rates, averaged per pair. Illustrative numbers.



IN PLAIN WORDS

With millions of users you can’t keep one number per user. That needs function approximation: Part 2.

Applied Computational Mathematics Lab

PART 2

A first look at function approximation

Features, linear values, and why a table is a special case.



Value function approximation

- So far, mostly **lookup tables**: every state s has an entry $v(s)$, or every pair s, a an entry $q(s, a)$.
- Problems with large MDPs:
  - too many states and/or actions to store in memory;
  - too slow to learn the value of each state separately;
  - individual states are often not fully observable.
- Solution: estimate the values with a function with parameters $\mathbf{w}$, and update $\mathbf{w}$ (for example with MC or TD):

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

EXAMPLES

Chess has over 10^{40} positions. A robot camera gives continuous pixels: no two images are the same. No table could hold either.



IN PLAIN WORDS

The big win is **generalisation**: what you learn in one state carries over to similar states you have never seen.

TRY IT

A problem with a million states, described by 3 features. A table needs a million numbers; a linear value function needs 3 weights.

The agent state

- If the environment state is not fully observable, use the **agent state**:

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

- with parameters ω (typically $\omega \in \mathbb{R}^n$).
- From now on, S_t denotes the agent state.
- Think of it as a vector inside the agent, or, in the simplest case, just the current observation: $S_t = O_t$.
- For now we won't discuss how to learn the update u_ω. Feel free to read S_t as an observation.

IN

Observation

What the agent sees now.

KEPT

Agent state

A summary of the past.

OUT

Value

A prediction from that summary.

EXAMPLE

A poker player can't see the other hands. Her agent state: her cards, the table, and what she remembers about how each opponent bet.

Features and linear values

- Represent a state by a **feature vector** $\mathbf{x}(s) = (x_1(s), \dots, x_m(s))^\top$, a fixed map from the agent state to $\mathbb{R}^m$. Short-hand: $\mathbf{x}_t = \mathbf{x}(S_t)$.
- Examples: distances of a robot from landmarks; trends in the stock market; piece and pawn configurations in chess.
- A linear value function, and its loss over the state distribution d:

$$v_{\mathbf{w}}(s) = \mathbf{w}^\top \mathbf{x}(s) = \sum_{j=1}^{m} x_j(s)\, w_j, \qquad L(\mathbf{w}) = \mathbb{E}_{S\sim d}\big[(v_\pi(S) - \mathbf{w}^\top \mathbf{x}(S))^2\big]$$

- L is quadratic in $\mathbf{w}$, so stochastic gradient descent converges to the global optimum. Since $\nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t) = \mathbf{x}_t$:

$$\Delta\mathbf{w} = \alpha\big(v_\pi(S_t) - v_{\mathbf{w}}(S_t)\big)\,\mathbf{x}_t$$



IN PLAIN WORDS

Update = step size × prediction error × feature vector. Features that were on get the blame, or the credit.

TRY IT

$\mathbf{w} = (1, 2)$, $\mathbf{x} = (3, 0.5)$, so $v = 4$. The target is 6, $\alpha = 0.05$.
$\Delta\mathbf{w} = 0.05 \times 2 \times (3, 0.5) = (0.3, 0.05)$.
New $v = 1.3 \times 3 + 2.05 \times 0.5 = 4.925$: closer to 6.

A table is a special case

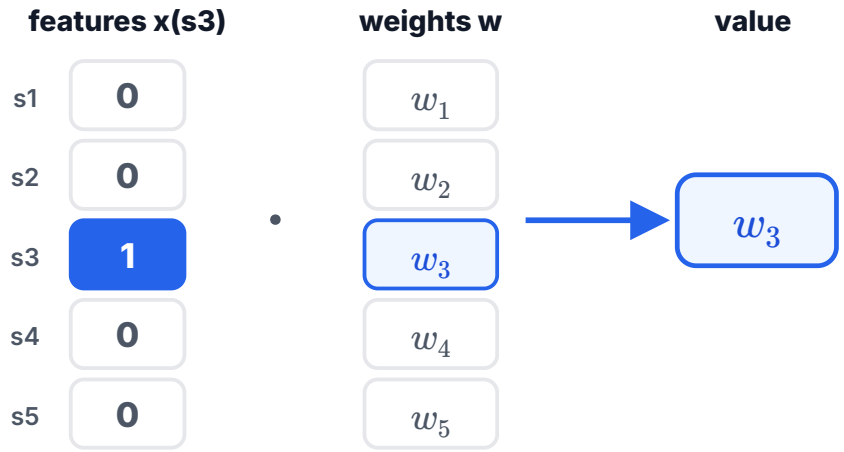
- Let the n states be $\mathcal{S} = \{s_1, \dots, s_n\}$. Use one-hot features:

$$\mathbf{x}(s) = \big(\mathbb{I}(s = s_1), \dots, \mathbb{I}(s = s_n)\big)^\top$$

- Then the weights simply hold one value per state:

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

- And the linear update changes only that state's entry: the incremental average from Lecture 2. Same formula, new clothes.



one feature per state: the weight is that state's table entry



IN PLAIN WORDS

A linear model sounds weak. With good features, it goes surprisingly far.

Monte Carlo with a learned q

- For a contextual bandit, q can be any parametric function, for example a neural network, trained on the loss

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

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

- Sample the expectation: a stochastic gradient update. This is just regression, and it works for large, even continuous, state spaces.

MODEL	GRADIENT	UPDATE
table	one-hot	only the cell $[S_t, A_t]$ moves
linear, $\mathbf{w}^\top \mathbf{x}(s,a)$	$\mathbf{x}(s,a)$	step × error × features
non-linear (a network)	$\nabla_{\mathbf{w}} q_{\mathbf{w}}$	step × error × gradient



IN PLAIN WORDS

Same idea as supervised learning, with the observed reward as the label.

Monte Carlo policy evaluation

- Now sequential 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 $T > t$, and the value is its expectation:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T, \qquad v_\pi(s) = \mathbb{E}[G_t \mid S_t = s, \pi]$$

- Use the **sample average** of returns instead of the expected return: **Monte Carlo policy evaluation**.



IN PLAIN WORDS

For every state you visited, look at everything that came after it. That is its return.

LINK

Lecture 2 averaged rewards per action. Now we average returns per state: the same idea, over longer stretches of time.



return **5.536** | return **5.04** | return **5.6** | return **4**

returns computed backwards: each is its reward plus 0.9 × the next return

One episode, $\gamma = 0.9$. Our numbers: from S_2, $G = 2 + 0.9 \times 4 = 5.6$; from S_0, $G = 1 + 0.9 \times 5.04 = 5.536$.

The Monte Carlo algorithm

MONTE CARLO PREDICTION

1. Play an episode with π until it ends.
2. Compute the returns backwards: $G_T = 0$, $G_t = R_{t+1} + \gamma G_{t+1}$.
3. For each visited state S_t: $v(S_t) \leftarrow v(S_t) + \alpha\big(G_t - v(S_t)\big)$.
4. Repeat with a new episode.

CHOICE	OPTIONS
which visits count	**first visit** per episode, or **every visit**: both converge
step size	$\alpha = 1/N(S_t)$: the exact average
	constant α: forgets old episodes, good if the world changes



IN PLAIN WORDS

Nudge each state's value toward the return that really followed it. No p anywhere.

TRY IT

A state was visited in three episodes with returns 5, 1 and 6. With $\alpha = 1/N$, its value goes 5, then 3, then 4: the average.

Example: Blackjack

- Get closer to 21 than the dealer without going over. Every game is an episode, and it always ends.
- **States** (200): your sum (12–21), the dealer's showing card (ace–10), and whether you hold a usable ace (yes/no).
- **Actions**: *stick* (stop and end) or *draw* (take another card). Below 12 you always draw automatically.
- **Rewards**: when you stick, +1 if your sum beats the dealer's, 0 if equal, −1 if lower. Drawing past 21: −1 and the game ends; otherwise 0.

STATE

3 numbers

sum, dealer card, usable ace

POLICY WE EVALUATE

stick on 20 or 21

otherwise always draw

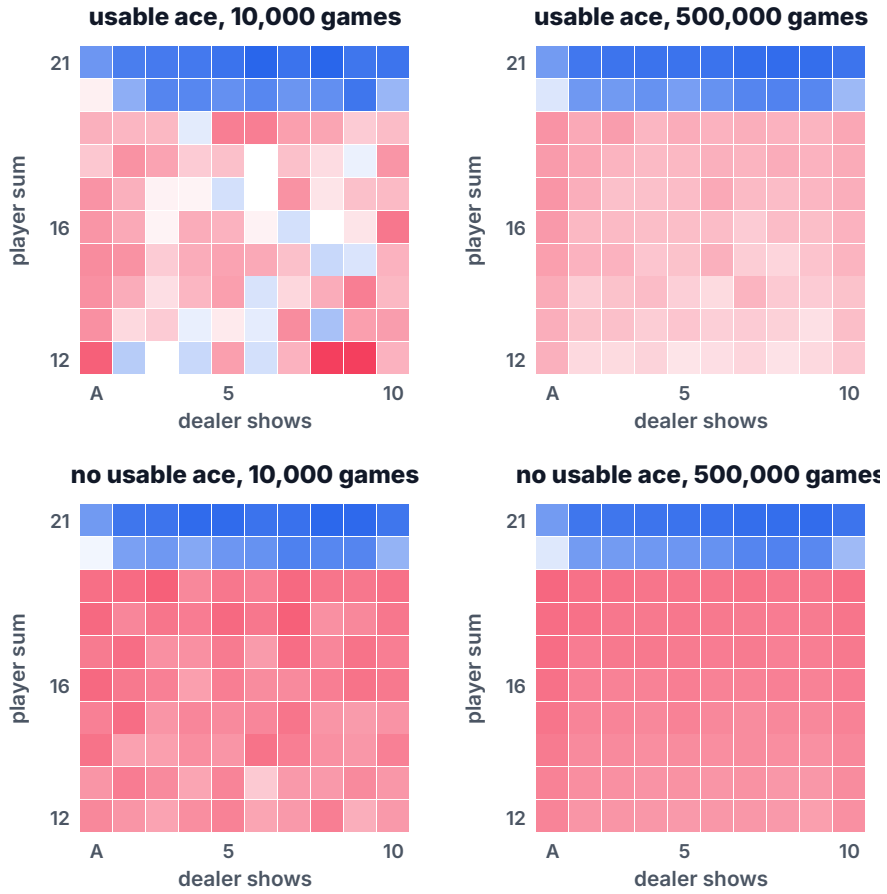


IN PLAIN WORDS

A perfect fit for Monte Carlo: the rules are a pain to write as probabilities, but games are cheap to play.

Example 5.1 of Sutton & Barto (2018). In our run the cards come from an infinite deck, and the dealer draws below 17.

Blackjack values after Monte Carlo



Our run, policy “stick on 20 or 21”. Blue: expected win; rose: expected loss; white: 0.

- After 10,000 games the values are bumpy, especially with a usable ace: those states are rare.
- After 500,000 games the surface is smooth.

YOUR SUM, DEALER SHOWS	VALUE
21, a 10	+0.89
20, a 10	+0.44
20, an ace	+0.15
19, a 10	−0.75



IN PLAIN WORDS

Drawing on 19 is not a great plan. Monte Carlo found that without knowing a single rule.

The catch with Monte Carlo

- Monte Carlo can learn value predictions. But:
  - when episodes are long, learning is slow: we must wait until an episode ends before we can learn;
  - returns can have high variance: each adds up many random steps.
- And episodes that never end give no learning at all.
- Are there alternatives? Yes: temporal-difference learning, Part 3.



IN PLAIN WORDS

Two hundred states, and 10,000 games still leave holes. Now imagine chess.

EXAMPLE

A thermostat or a trading system runs forever. There is no “end of episode” to wait for, so plain Monte Carlo cannot learn there.

TRY IT

An episode of 100 steps, $\gamma = 1$, each reward with independent noise of standard deviation 1. The return adds 100 noisy terms: its standard deviation is $\sqrt{100} = 10$, ten times that of one step.

Check yourself: Parts 1 and 2

CHECK YOURSELF

1. An episode gives rewards 0, 0, 1 and then ends, with $\gamma = 0.5$. What is G_0?

CHECK YOURSELF

2. A state's returns so far are 3, 5 and 1. What is its Monte Carlo estimate?

CHECK YOURSELF

3. With 4 states and one-hot features, what is $\mathbf{x}(s_2)$, and what is $v(s_2)$?

CHECK YOURSELF

4. $\mathbf{w} = (2, -1)$, $\mathbf{x} = (1, 1)$, target 3, $\alpha = 0.1$. What is the new $\mathbf{w}$?

CHECK YOURSELF

5. Why can't plain Monte Carlo learn in a task that never ends?





HINT

Answers are at the end of the deck.

Applied Computational Mathematics Lab

PART 3

Temporal-difference learning

Learning a guess from a guess, after every step.



Sampling the Bellman equation

- Lecture 3: the Bellman equation, and dynamic programming iterates it:

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

- We can **sample** this: $v_{t+1}(S_t) = R_{t+1} + \gamma v_t(S_{t+1})$. That is noisy, so take a small step instead:

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



IN PLAIN WORDS

Dynamic programming computes the expectation, using p. TD says: I don't have p , but I just took one step. Let me use it as a sample.

IN ONE LINE

It is Bellman's equation again, with one sample instead of the full average. The rest is details.

Monte Carlo and TD side by side

- Prediction: learn v_π online from experience under π.
- **Monte Carlo**: move $v(S_t)$ toward the sampled return G_t:

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

- **Temporal difference**: move $v(S_t)$ toward the estimated return $R_{t+1} + \gamma v(S_{t+1})$:

$$v(S_t) \leftarrow v(S_t) + \alpha\,\delta_t, \qquad \delta_t = R_{t+1} + \gamma v(S_{t+1}) - v(S_t)$$

- δ_t is the **TD error**. Next state better than expected: $\delta_t > 0$ and the value goes up.

TRY IT

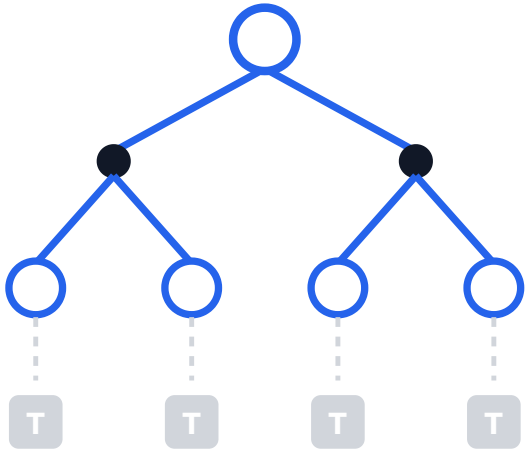
$v(s) = 0.5$, you get $r = 0$ and land in s' with $v(s') = 0.7$; $\gamma = 1$, $\alpha = 0.1$.
$\delta = 0 + 0.7 - 0.5 = 0.2$.
New $v(s) = 0.5 + 0.1 \times 0.2 = 0.52$.



IN PLAIN WORDS

Learning a guess from a guess. It sounds like cheating. It works: the reward is real, and each step a little more truth gets in.

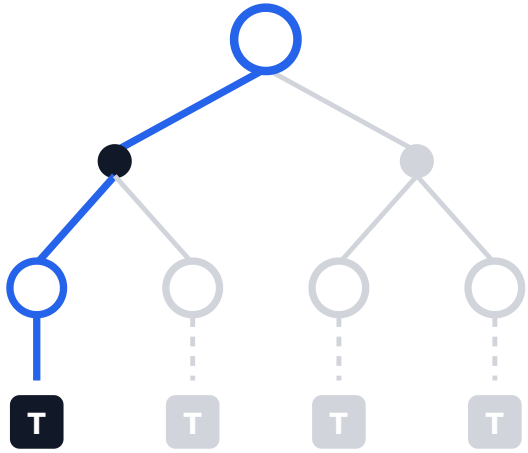
Three kinds of backup



Dynamic programming

one step, every branch

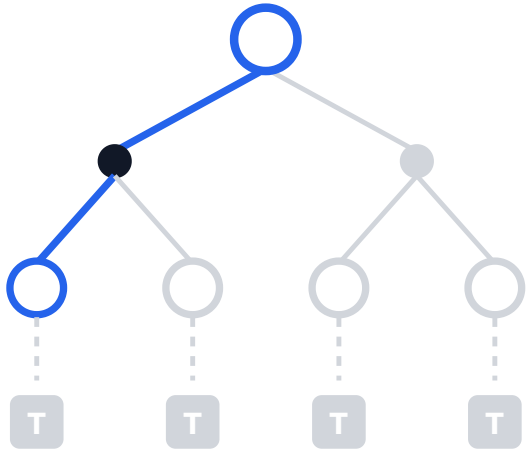
$$v(S_t) \leftarrow \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid A_t \sim \pi(S_t)\right]$$



Monte Carlo

one sampled branch, to the end

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



Temporal difference

one step, one sampled branch

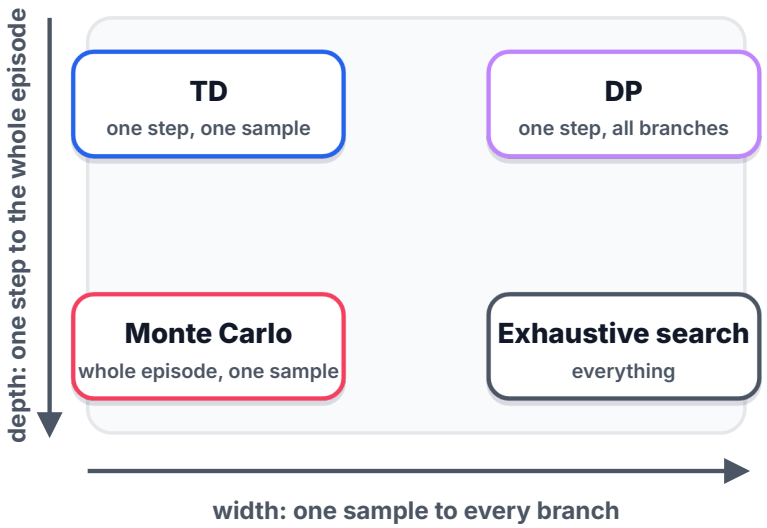
$$v(S_t) \leftarrow v(S_t) + \alpha\big(R_{t+1} + \gamma v(S_{t+1}) - v(S_t)\big)$$



IN PLAIN WORDS

TD and DP both look one step ahead. DP averages over every next state; TD uses the one that actually happened. MC follows one branch all the way to the end (T).

Bootstrapping and sampling



BOOTSTRAPPING

The update uses an **estimate** of the value. DP and TD bootstrap; MC does not.

SAMPLING

The update uses a **sample** instead of an expectation. MC and TD sample; DP does not.

	BOOTSTRAPS?	SAMPLES?
Dynamic programming	yes	no
Monte Carlo	no	yes
TD	yes	yes



IN PLAIN WORDS

TD is the cheap corner: one step, one sample. Everything else lives somewhere in between.

TD for action values: SARSA

- The same idea works for action values: move $q(S_t, A_t)$ toward $R_{t+1} + \gamma q(S_{t+1}, A_{t+1})$:

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha\big(R_{t+1} + \gamma q(S_{t+1}, A_{t+1}) - q(S_t, A_t)\big)$$

- Known as **SARSA**, after the five things it uses: $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$. We'll use it for control in Lecture 6.

1

Model-free

No knowledge of the MDP.

2

Incomplete episodes

Learns by bootstrapping.

3

Online

Learns during each episode.



IN PLAIN WORDS

Like tasting the soup every minute while you cook, instead of waiting until the guests have left.

TRY IT

$q(s, a) = 2$, reward 1, next pair has $q(s', a') = 3$, $\gamma = 0.9$, $\alpha = 0.5$. Target $1 + 2.7 = 3.7$; new $q(s, a) = 2 + 0.5 \times 1.7 = 2.85$.

Example: driving home

STATE	ELAPSED (MIN)	PREDICTED TIME TO GO	PREDICTED TOTAL
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**

Example 6.1 of Sutton & Barto (2018).



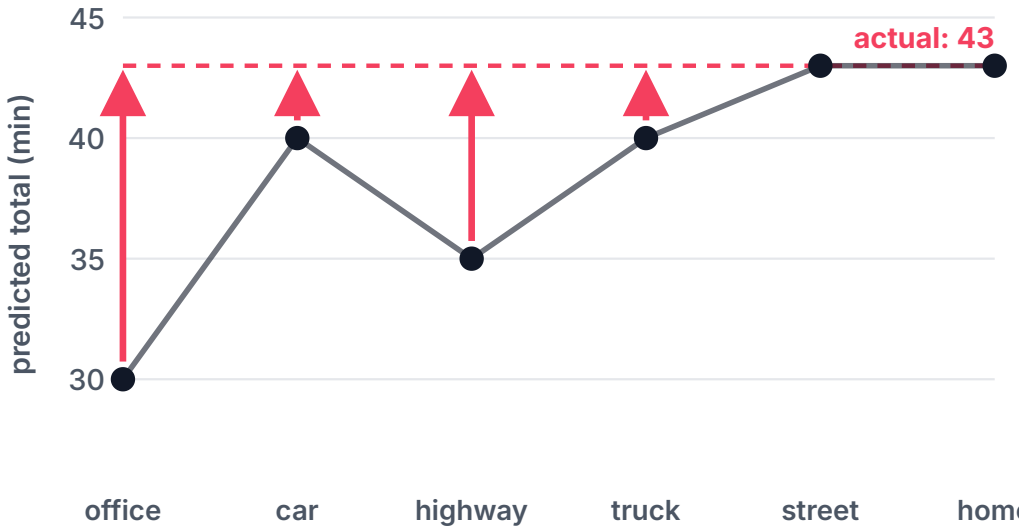
IN PLAIN WORDS

Each moment I predict the total trip time: elapsed time plus my guess of the time to go. Watch the predicted total move.

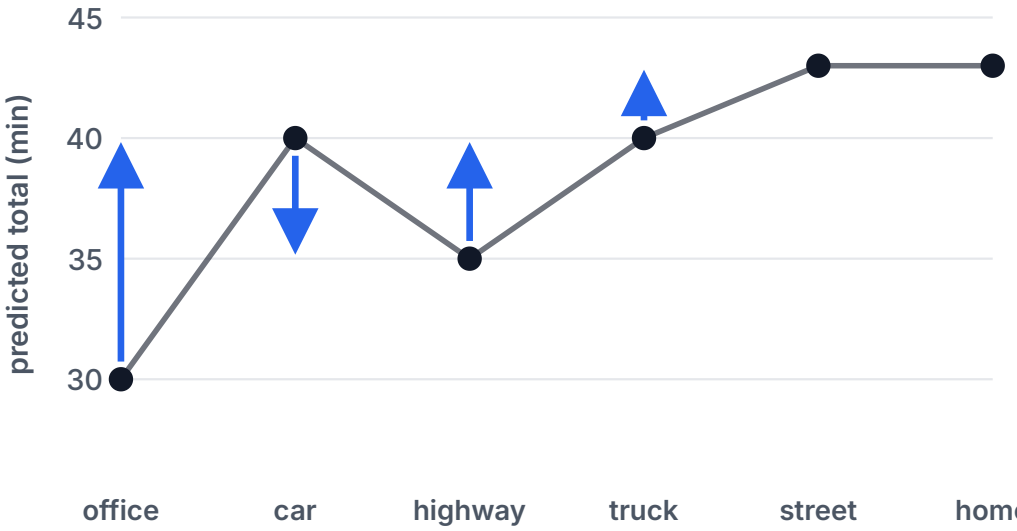
THE STORY

Leaving the office: 30 minutes. Rain at the car: 40. The highway goes well: 35. Stuck behind a truck: 40 again. My street: 43. Home after 43 minutes.

Driving home: Monte Carlo versus TD



Monte Carlo: every prediction moves to the final 43



TD: each prediction moves to the next one

- Monte Carlo ($\alpha = 1$) must wait until I'm home, then changes the five predictions by +13, +3, +8, +3, 0.
- TD ($\alpha = 1$) changes each prediction toward the next one as I drive: +10, −5, +5, +3, 0. Stuck behind the truck, I already know I was too optimistic on the highway.

Check yourself: Part 3

CHECK YOURSELF

6. $v(s) = 4$, reward -1, $v(s') = 6$, $\gamma = 0.5$, $\alpha = 0.5$. What is the TD error, and the new $v(s)$?

CHECK YOURSELF

7. Which of DP, MC and TD bootstrap? Which sample?

CHECK YOURSELF

8. In the driving example, which prediction does TD change the most?

CHECK YOURSELF

9. Why can TD learn in a task that never ends?



Applied Computational Mathematics Lab

PART 4

Comparing Monte Carlo and TD

Bias and variance, a race, and what happens with finite data.



Advantages and disadvantages

	TD	MONTE CARLO
Learns before the final outcome is known	yes, online after every step	no, waits until the episode ends
Learns from incomplete sequences	yes	no, only complete ones
Continuing (non-terminating) tasks	works	episodic tasks only
Memory for long episodes	single transitions are enough	must store the episode to update at the end
Needs reasonable value estimates	yes: it leans on them	no



IN PLAIN WORDS

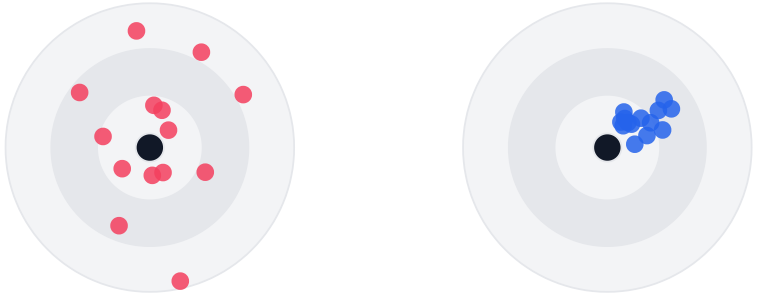
TD is independent of the time span of the prediction. Monte Carlo isn't: a long episode means a long wait.

EXAMPLE

A robot that runs for weeks never reaches an end. TD learns anyway.

The bias–variance trade-off

- The MC return $G_t = R_{t+1} + \gamma R_{t+2} + \cdots$ is an **unbiased** estimate of $v_\pi(S_t)$.
- The TD target $R_{t+1} + \gamma v_t(S_{t+1})$ is **biased**, unless $v_t(S_{t+1}) = v_\pi(S_{t+1})$.
- But the TD target has much lower **variance**: the return depends on many random actions, transitions and rewards; the TD target on only one of each.
- TD can keep an irreducible bias if the world is partially observable (MC implicitly accounts for hidden variables) or the function approximation fits poorly.
- In the tabular case, both converge: $v_t \to v_\pi$.



Monte Carlo
no bias, high variance

TD
some bias, low variance

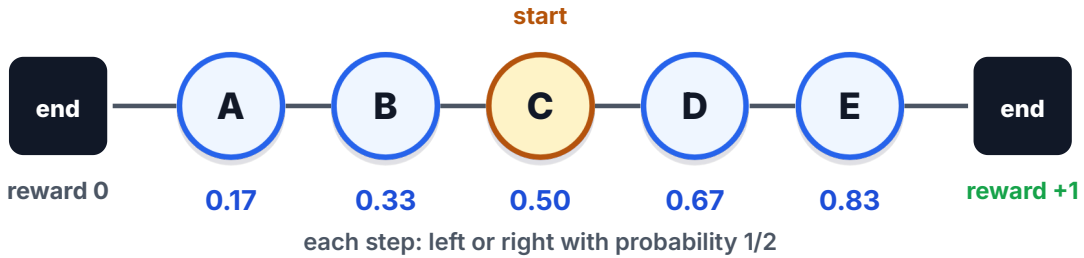
An illustration, not data: each dot is one update target, the bullseye is the true value.



IN PLAIN WORDS

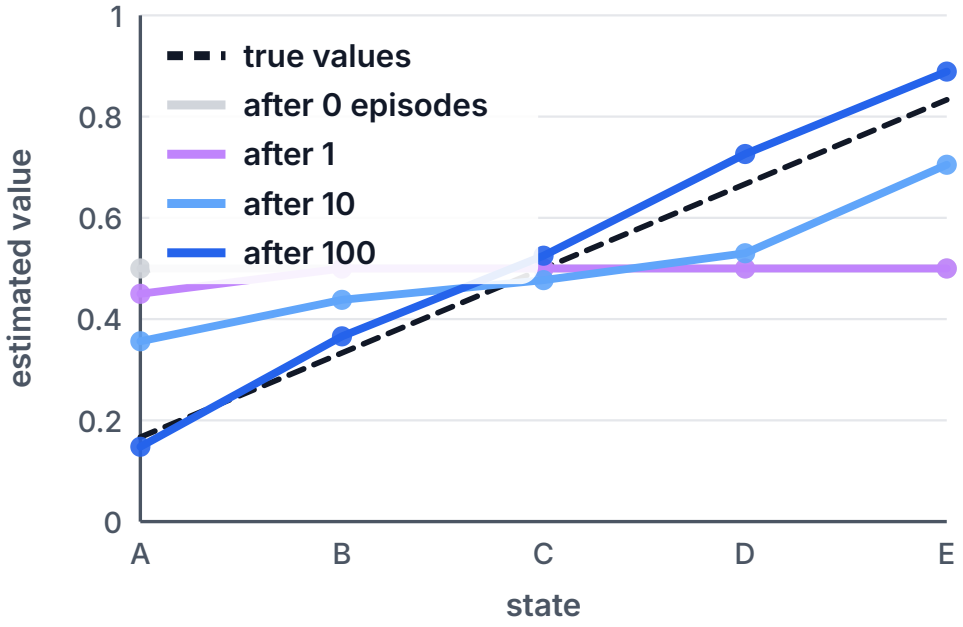
Lower variance means smoother learning and usually a larger step size you can afford.

Example: a random walk



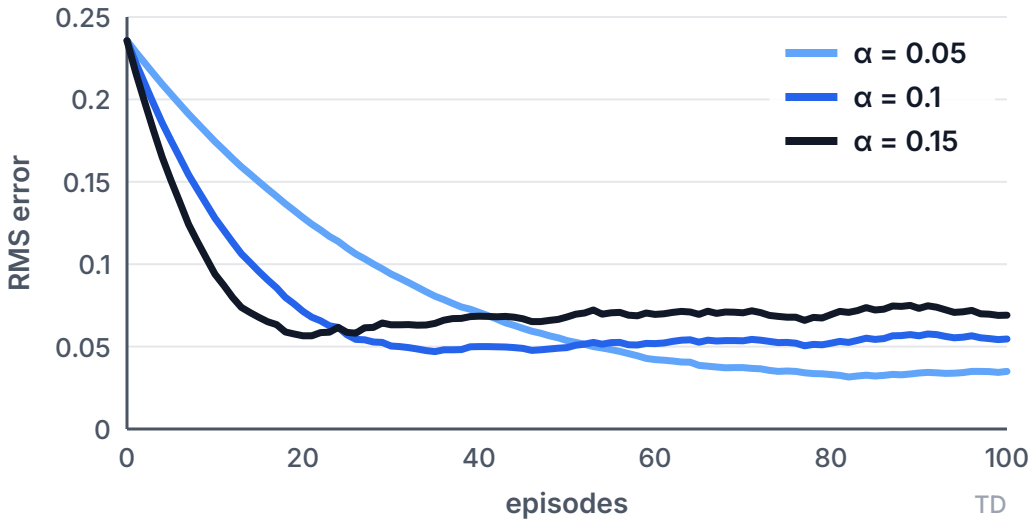
True values below the squares: the chance of finishing on the right, 1/6 to 5/6. Example 6.2 of Sutton & Barto (2018).

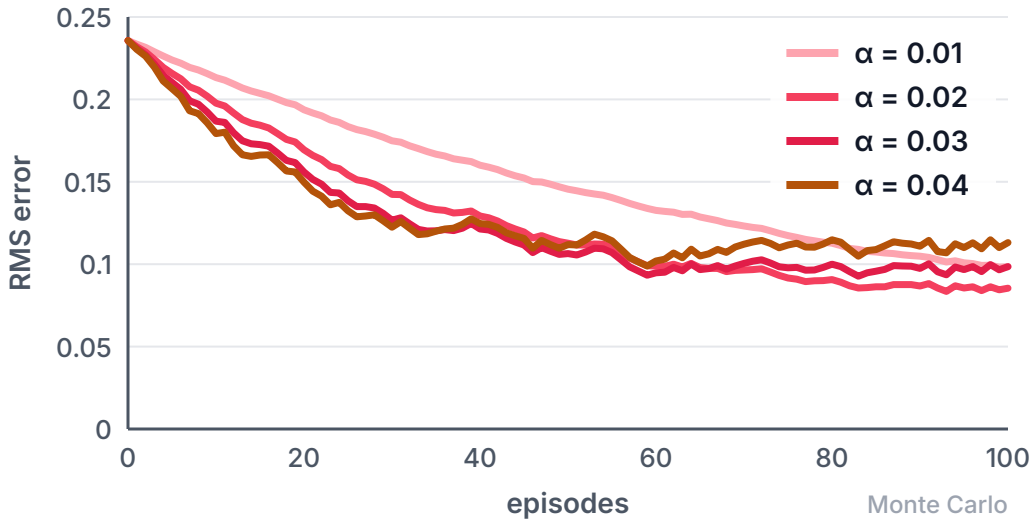
- Start every estimate at 0.5 and run TD with $\alpha = 0.1$.
- After the first episode only A moved: every other step compared 0.5 with 0.5, a TD error of 0.



Our run of TD, $\alpha = 0.1$. Information flows backwards from where the rewards are.

The race: Monte Carlo versus TD





Our run: 100 runs of 100 episodes, error averaged over the five states. After 100 episodes TD reaches 0.035 (α = 0.05); the best Monte Carlo run reaches 0.085.

- On this problem TD learns faster, for every step size we tried.
- The largest TD step is fastest at first, then stalls higher; the smallest is slow, but ends lowest. A common pattern, not a theorem.



IN PLAIN WORDS

Monte Carlo pays for its noise; TD pays for leaning on guesses that are still wrong.

Batch Monte Carlo 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 K episodes:

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

- Repeatedly sample an episode $k \in [1, K]$ and apply MC or TD(0), until the values stop moving. This is like sampling from an empirical model.



IN PLAIN WORDS

Many real projects look like this: a log of data, and that's all you'll get.

EXAMPLES

A log of driving data, a hospital's records, a history of games. Fixed. Squeeze everything out of it.

Batch learning in two states

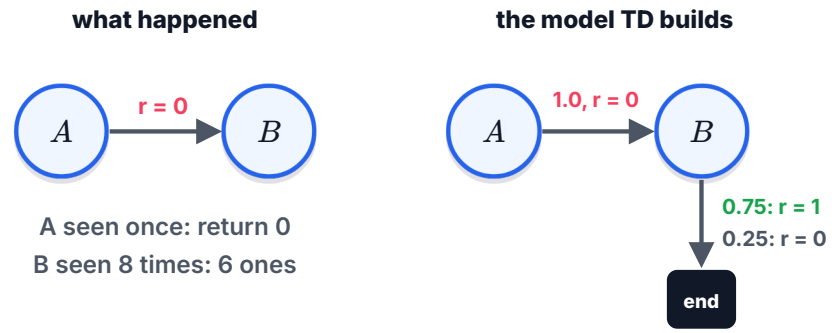
- Two states, A and B; no discounting; 8 episodes of experience:

EPISODES	WHAT HAPPENED
1	A, 0, B, 0, end
6	B, 1, end
1	B, 0, end

YOUR TURN

What are $v(A)$ and $v(B)$? Think about it before reading on.

- $v(B)$: 6 good endings out of 8, so 0.75. Everyone agrees.
- $v(A)$: **Monte Carlo says 0**: the one time we saw A, the return was 0. **TD says 0.75**: A always went to B with reward 0, and B is worth 0.75.



Example 6.4 of Sutton & Barto (2018). Our drawing.



IN PLAIN WORDS

Same data, two answers. Both right: they answer different questions.

What the batch solutions mean

- **Monte Carlo** converges to the best mean-squared fit of the observed returns:

$$\min_v \sum_{k=1}^{K} \sum_{t=1}^{T_k} \left(G_t^k - v(S_t^k)\right)^2 \qquad (\text{in the AB example, } v(A) = 0)$$

- **TD** converges to the solution of the maximum-likelihood Markov model given the data: the empirical MDP $(\mathcal{S}, \mathcal{A}, \hat{p}, \gamma)$ that best fits it.
- In the AB example $\hat{p}(S_{t+1} = B \mid S_t = A) = 1$, so $v(A) = v(B) = 0.75$.
- This is called the **certainty-equivalence** estimate: assume your estimated model is the truth, and solve it.

	TD	MONTE CARLO
Markov property	exploits it	does not use it
Helps when	the state is fully observable	the state is partially observable
Finite data or approximation	answers may differ	answers may differ



IN PLAIN WORDS

Trust your state? TD. Don't trust it? Monte Carlo. A camera only sees what is in front of it.

Check yourself: Part 4

CHECK YOURSELF

10. Which target is unbiased, MC or TD? Which has lower variance?

CHECK YOURSELF

11. In the AB example, suppose we also saw A five more times, each time "A, 0, B, 1, end". What is the Monte Carlo estimate of $v(A)$?

CHECK YOURSELF

12. In the random walk, why did only A change after the first episode?

CHECK YOURSELF

13. A robot sees the world through a narrow camera. Which batch answer would you trust more, MC or TD? Why?



Applied Computational Mathematics Lab

PART 5

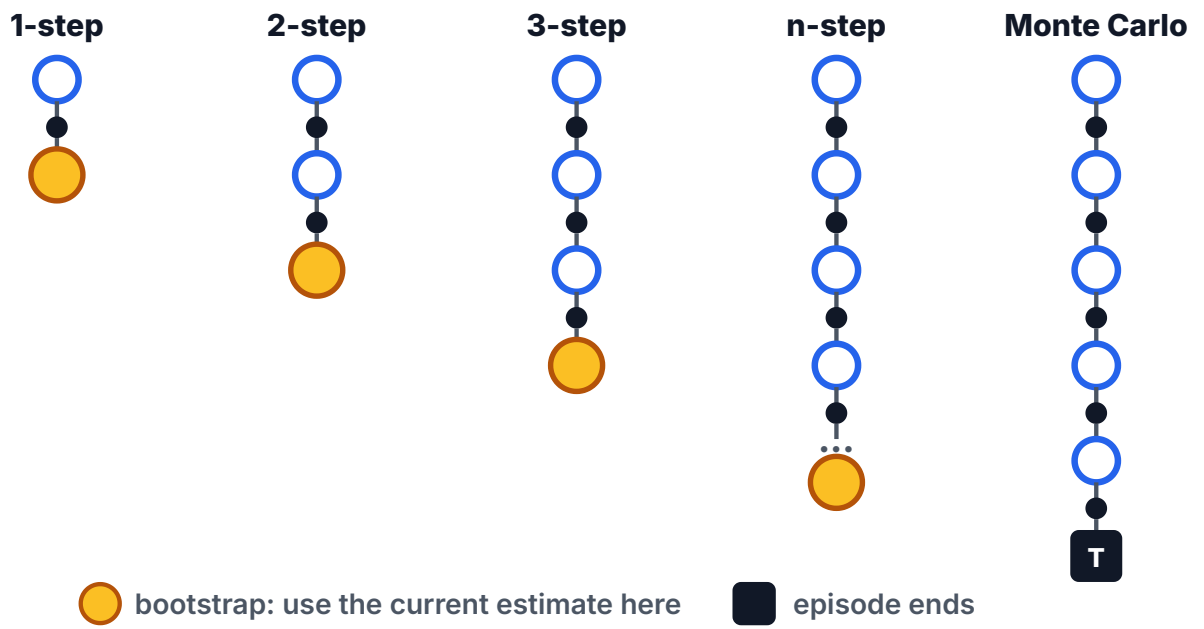
Multi-step returns and traces

Between TD and Monte Carlo: n-step returns, the λ-return and eligibility traces.



Between TD and Monte Carlo

- TD uses value estimates which might be inaccurate.
- And information propagates back slowly: one step per episode.
- In Monte Carlo information propagates faster, but the updates are noisier.
- We can go in between: let the TD target look n steps into the future.



Each column is one backup: real rewards along the way, then either an estimate (gold) or the end of the episode.

Multi-step returns

$$
\begin{aligned}
n = 1\ (\mathrm{TD}): &\quad G_t^{(1)} = R_{t+1} + \gamma v(S_{t+1}) \\
n = 2: &\quad G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 v(S_{t+2}) \\
n = \infty\ (\mathrm{MC}): &\quad G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T
\end{aligned}
$$

- In general, the n-step return:

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

- Multi-step TD: $v(S_t) \leftarrow v(S_t) + \alpha\big(G_t^{(n)} - v(S_t)\big)$.



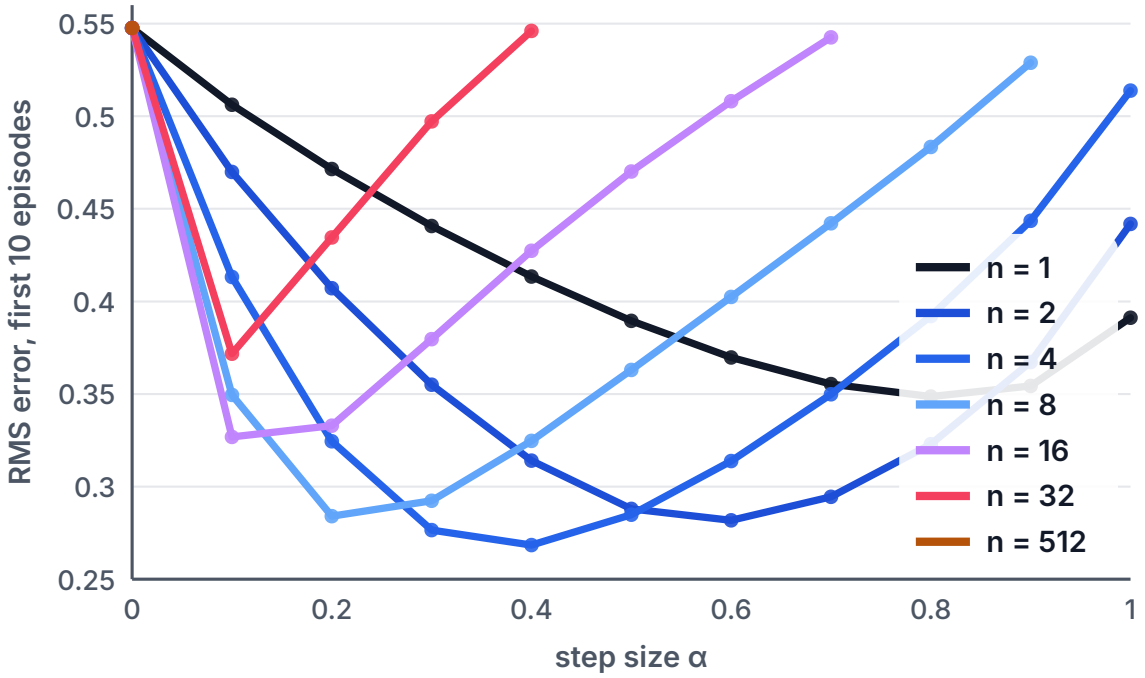
IN PLAIN WORDS

A dial: TD at one end, Monte Carlo at the other. Same update rule; only the target changes.

TRY IT

Rewards 1, 0, 2, then the episode ends; $v(S_1) = 3$, $v(S_2) = 5$, $\gamma = 0.9$.
$G^{(1)} = 1 + 0.9 \times 3 = 3.7$.
$G^{(2)} = 1 + 0 + 0.81 \times 5 = 5.05$.
$G^{(3)} = 1 + 0 + 0.81 \times 2 = 2.62$: the episode ended, so this is the MC return.

Which n is best?



Our run: a 19-state random walk (−1 off the left end, +1 off the right), 100 runs of 10 episodes for each n and α. Points above 0.56 are cut off. The setting of Figure 7.2 in Sutton & Barto (2018).

- The winner: $n = 4$ with $\alpha = 0.4$ (error 0.268). Not 1, not infinity: somewhere in the middle.
- Large n with a large step size is worse than not learning at all (the start, $\alpha = 0$, has error 0.548): too much noise.



IN PLAIN WORDS

Short steps lean on bad early guesses; long ones drown in noise. On another problem the best n will be different.

Mixing all n: the λ-return

- Multi-step returns bootstrap on one state, $v(S_{t+n})$. We can also bootstrap a little on every state:

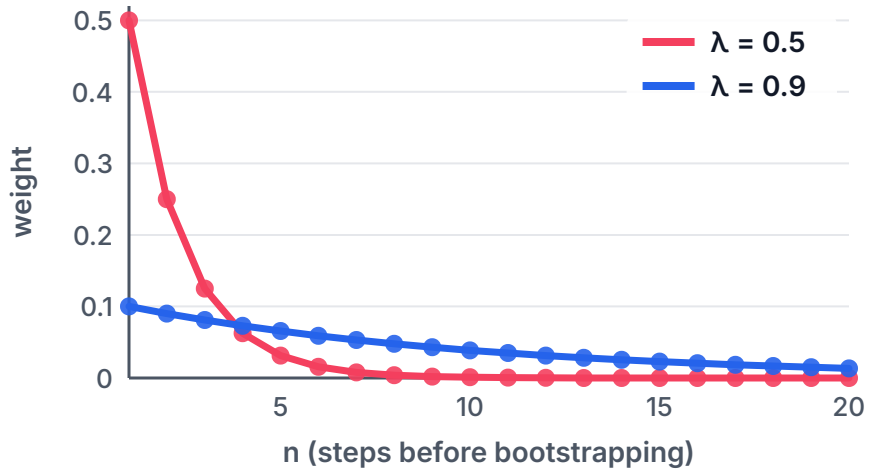
$$G_t^\lambda = R_{t+1} + \gamma\big((1-\lambda)\, v(S_{t+1}) + \lambda\, G_{t+1}^\lambda\big) \;=\; \sum_{n=1}^{\infty} (1-\lambda)\lambda^{n-1} G_t^{(n)}$$

- The weights $(1-\lambda)\lambda^{n-1}$ add up to 1: a proper weighted average of the n-step returns.
- Special cases: $G_t^{\lambda=0} = R_{t+1} + \gamma v(S_{t+1})$, which is TD, and $G_t^{\lambda=1} = R_{t+1} + \gamma G_{t+1}$, which is Monte Carlo.



IN PLAIN WORDS

One step gets weight $1-\lambda$; each further step, multiply by λ again.



Our computation of the weights $(1-\lambda)\lambda^{n-1}$.

The λ-return in practice

TRY IT

The same episode as before, $\lambda = 0.5$. Weights: 0.5 on $G^{(1)}$, 0.25 on $G^{(2)}$, and all the rest, 0.25, on the full return $G^{(3)}$ (the episode ended).

$G^{\lambda} = 0.5 \times 3.7 + 0.25 \times 5.05 + 0.25 \times 2.62 = 3.7675.$

- Intuition: $1/(1-\lambda)$ is the "horizon", just like $1/(1-\gamma)$.
- Multi-step returns have benefits from both TD and MC: bootstrapping can have issues with bias, Monte Carlo with variance.
- Typically, intermediate values are good, for example $n = 10$ or $\lambda = 0.9$.

λ	HORIZON $1/(1-\lambda)$	BEHAVES LIKE
0	1	TD
0.5	2	about 2 steps
0.9	10	about 10 steps
0.99	100	about 100 steps
1	∞	Monte Carlo

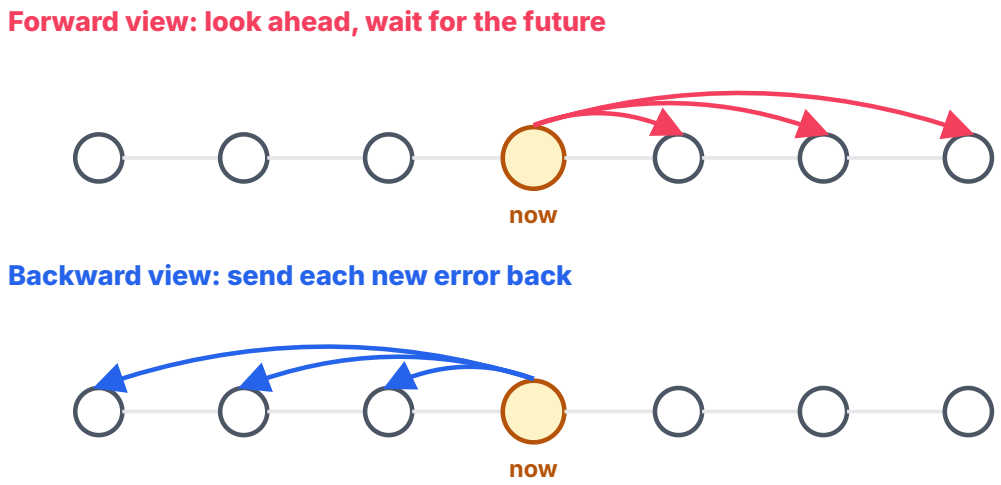


IN PLAIN WORDS

λ is a dial too, but a smooth one: no need to pick a whole number of steps.

Can we have it both ways?

- MC and multi-step returns are not independent of the span of the predictions: to update values in a long episode, you have to wait.
- TD updates immediately, and is independent of the span.
- Can we get both: the targets of MC (or the λ-return) and the immediacy of TD?



The forward view stands in a state and looks ahead. The backward view stands in the present and asks: who deserves the credit for this new error?



IN PLAIN WORDS

Yes. The trick is to look back instead of forward.

The MC error is a sum of TD errors

- With the values held fixed during an episode:

$$
\begin{aligned}
G_t - v(S_t) &= R_{t+1} + \gamma G_{t+1} - v(S_t) \\
&= \underbrace{R_{t+1} + \gamma v(S_{t+1}) - v(S_t)}_{=\delta_t} + \gamma\big(G_{t+1} - v(S_{t+1})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\big(G_{t+2} - v(S_{t+2})\big) = \cdots = \sum_{k=t}^{T} \gamma^{k-t}\delta_k
\end{aligned}
$$

- Each TD error δ_k is known at its own time step k, so this sum can be built up while the episode unfolds, instead of waiting for G_t.

t	R_{t+1}	$v(S_t)$	$v(S_{t+1})$	δ_t
0	1	2	3	$1 + 2.7 - 2 = 1.7$
1	0	3	5	$0 + 4.5 - 3 = 1.5$
2	2	5	0 (end)	$2 + 0 - 5 = -3$

TRY IT

Our episode: $v(S_0) = 2$, $v(S_1) = 3$, $v(S_2) = 5$; rewards 1, 0, 2; $\gamma = 0.9$.
TD errors: 1.7, 1.5, -3.
$1.7 + 0.9 \times 1.5 + 0.81 \times (-3) = 0.62$.
MC error: $2.62 - 2 = 0.62$. The same.



IN PLAIN WORDS

So the Monte Carlo update can be paid out in instalments: one TD error at a time.

Eligibility traces

- With linear values $v_{\mathbf{w}}(s) = \mathbf{w}^\top \mathbf{x}(s)$, Monte Carlo updates all states of an episode at once: $\Delta \mathbf{w} = \sum_{t=0}^{T-1} \alpha (G_t - v(S_t)) \mathbf{x}_t$.
- Substitute the sum of TD errors and swap the order of summation:

$$\Delta \mathbf{w} = \sum_{t=0}^{T-1} \alpha \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k \mathbf{x}_t = \sum_{k=0}^{T-1} \alpha \, \delta_k \underbrace{\sum_{t=0}^{k} \gamma^{k-t} \mathbf{x}_t}_{\equiv \mathbf{e}_k}, \qquad \mathbf{e}_t = \gamma \, \mathbf{e}_{t-1} + \mathbf{x}_t$$

- The vector $\mathbf{e}_t$ is the **eligibility trace**. Every step it decays by γ, then the current features are added. Per step: $\Delta \mathbf{w}_t = \alpha \, \delta_t \, \mathbf{e}_t$, applied at the end of the episode.



IN PLAIN WORDS

One update reaches every past state, in proportion to its trace. No need to store the episode or recompute old values.

	$\delta_0 \mathbf{e}_0$	$\delta_1 \mathbf{e}_1$	$\delta_2 \mathbf{e}_2$
$\mathbf{x}_0$	δ_0	$\gamma \delta_1$	$\gamma^2 \delta_2$
$\mathbf{x}_1$		δ_1	$\gamma \delta_2$
$\mathbf{x}_2$			δ_2

Rows: each state's MC error, spread over time.
Columns: what each step's trace pays out.

Traces with λ

- For the λ-return the same algebra holds, with $\gamma\lambda$ instead of γ:

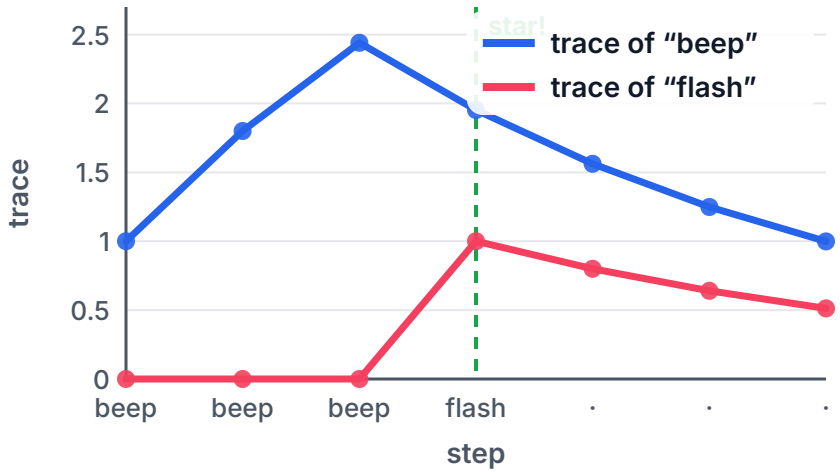
$$G_t^\lambda - v(S_t) = \sum_{k=0}^{T-t} (\lambda\gamma)^k \delta_{t+k} \quad \Longrightarrow \quad \mathbf{e}_t = \gamma\lambda\, \mathbf{e}_{t-1} + \mathbf{x}_t, \qquad \Delta\mathbf{w}_t = \alpha\, \delta_t\, \mathbf{e}_t$$

- An **accumulating trace** with decay $\gamma\lambda$. Exact for updates batched over an episode (“offline”); similar traces exist for fully online updating.
- $\lambda = 0$: traces vanish at once, plain TD. $\lambda = 1$: they fade only by γ, Monte Carlo.



IN PLAIN WORDS

A beep, a beep, a beep, then a flash, then a star. What earned the star? Recency says the flash; frequency says the beep. The trace blends both, and many episodes sort it out.



Our computation, $\gamma\lambda = 0.8$. When the star arrives, beep’s trace is 1.95 (frequent) and flash’s is 1.0 (recent): the credit is shared.

WRAP-UP

Summary

- **Monte Carlo**: average the returns that really followed. Model-free, unbiased, but noisy and must wait for the end.
- **Function approximation**: $v_{\mathbf{w}}(s) = \mathbf{w}^\top \mathbf{x}(s)$; update = step × error × features. A table is the one-hot case.
- **TD**: move toward $R_{t+1} + \gamma v(S_{t+1})$. Samples and bootstraps; learns online, from incomplete episodes; lower variance, some bias.
- **Batch**: MC fits the returns; TD solves the most likely Markov model.
- **n-step and λ** sit in between; **eligibility traces** give the λ-return's updates without waiting.

$$\text{MC: } v \leftarrow v + \alpha(G_t - v)$$

$$\text{TD: } v \leftarrow v + \alpha\big(R_{t+1} + \gamma v(S_{t+1}) - v\big)$$

$$\mathbf{e}_t = \gamma\lambda\, \mathbf{e}_{t-1} + \mathbf{x}_t, \quad \Delta\mathbf{w} = \alpha\delta_t \mathbf{e}_t$$

Answers to check yourself

- **1.** $G_0 = 0 + 0.5 \times 0 + 0.25 \times 1 = 0.25$.
- **2.** $(3 + 5 + 1)/3 = 3$.
- **3.** $\mathbf{x}(s_2) = (0, 1, 0, 0)$ and $v(s_2) = w_2$.
- **4.** $v = 1$, error 2, so $\mathbf{w} = (2, -1) + 0.2(1, 1) = (2.2, -0.8)$.
- **5.** The return is never complete, so there is nothing to average.
- **6.** $\delta = -1 + 3 - 4 = -2$; new $v(s) = 4 - 1 = 3$.
- **7.** DP and TD bootstrap; MC and TD sample.
- **8.** The first one, leaving the office: +10 (30 to 40).
- **9.** It updates after every step, using the next estimate; no end is needed.
- **10.** MC is unbiased; TD has lower variance.
- **11.** Returns from A: 0 and five 1s, so $5/6 \approx 0.83$.
- **12.** All other steps had TD error $0 + 0.5 - 0.5 = 0$; the episode ended off the left end, from A.
- **13.** Monte Carlo: with a narrow camera the state isn't Markov, and MC doesn't assume it is.

CREDITS

Sources and credits

- The order of topics follows *Lecture 5: Model-Free Prediction* of the DeepMind × UCL Reinforcement Learning Lecture Series (2021), by Hado van Hasselt. This deck is not an official DeepMind or UCL material.
- The text, figures, examples and simulations were written, drawn and run for ACML. Numbers labelled “our run” come from our own code; examples taken from books or papers are credited where they appear.
- Sutton & Barto (2018), *Reinforcement Learning: An Introduction*, chapters 5, 6, 7, 9 and 12: the Blackjack (5.1), driving home (6.1), random walk (6.2) and batch AB (6.4) examples and the 19-state walk of Figure 7.2. All runs re-done by us.
- Companion videos: **ACML Explains**, season 1, episodes 17–20. More from the lab: acml.teddylazebnik.com



Applied Computational Mathematics Lab

END OF LECTURE 5

Next: model-free control

Lecture 6 uses these prediction methods to improve policies without a model: Monte Carlo control, SARSA and Q-learning.

