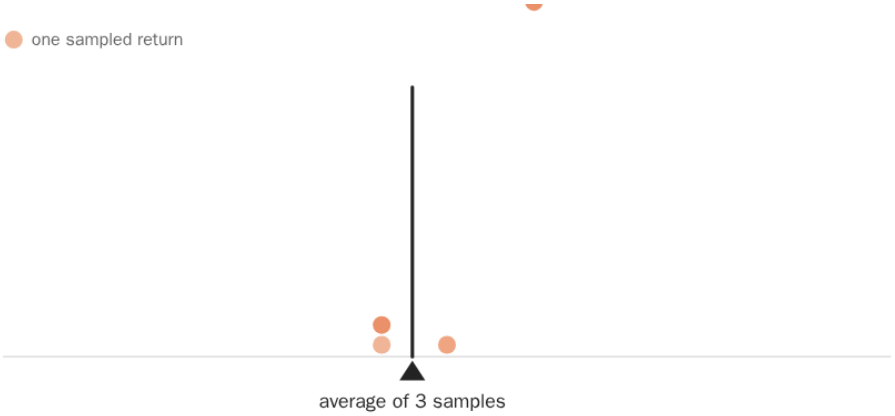
Learning from Experience



Reinforcement learning, part 3.1: prediction without a model, Monte Carlo and temporal differences

AUTHOR

Etienne Chassaing[1, 2]

AFFILIATIONS

1. Hugging Face
2. CentraleSupélec

PUBLISHED

Oct. 2026

Table of Contents

1 Introduction

- 1.1 Where Parts 1 and 2 left us in a few words
- 1.2 What if we don’t have the model?

2 1. Monte Carlo: learning from complete episodes

- 2.1 1.1 An average of returns
- 2.2 1.2 Evaluating VVV at the first visit or every visit
- 2.3 1.3 A constant step size

3 2. Temporal differences: learning from every step

- 3.1 2.1 Bootstrapping
- 3.2 2.2 TD(0) on a random walk

4 3. Monte Carlo and TD comparison

- 4.1 3.1 Estimating one the way : Driving home example
- 4.2 3.2 Bias and variance
- 4.3 3.3 RL in one plot: width and depth

5 4. Quick wrap-up before 3.2

Introduction

So far you discovered RL basics and how to solve problems as big as a tic-tac-toe but you might wonder how to solve larger problems for which we don't know the model. Welcome to this third part "Learning from Experience"

Where Parts 1 and 2 left us in a few words

Part 1, *Modeling the World* gave us the framework of RL: An agent in state S_t takes an action A_t, receives a reward R_{t+1} and lands in S_{t+1}; and it wants to maximize the return, the discounted sum of the rewards to come:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = R_{t+1} + \gamma\, G_{t+1}$$

The value functions score states and actions under a policy π: $v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s]$ and $q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$. Because the return is recursive, they satisfy the Bellman equations, one step plus the discounted value of where it leads:

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\, \big[r + \gamma\, v_\pi(s')\big]$$

When the model $p(s', r \mid s, a)$ is known, we can use dynamic programming to solve these equations by sweeps (Part 2, *Planning with a Model*): the Bellman equation is used as an update rule, applied to every state again and again:

$$v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\, \big[r + \gamma\, v_k(s')\big]$$

The key idea was that optimal policies are always found by algorithms belonging to generalized policy iteration (GPI) composed of two actions:

- an evaluation that makes the value table consistent with the policy ($V(s)$ converges to $V_\pi(s)$);
- an improvement that makes the policy greedy with respect to the value table, and the two meet at v_* and π_*.

Part 2 ended on the limit of this approach: even when the experience decides where to update, every update still needs the model to decide what the update is, which is rarely the case most cases.

What if we don't have the model?

All the methods we have seen so far compute $\sum_{s',r} p(s', r \mid s, a)\, [r + \gamma\, v(s')]$, which needs the model p. This session removes that requirement.

The key idea fits in one line:

> 💡 The key idea
>
> Replace the expectation over p by an average over experience we draw.
>
> $$\sum_{s',r} p(s', r \mid s, a)\, \big[r + \gamma\, v(s')\big] \quad \longrightarrow \quad \frac{1}{N} \sum_{i=1}^{N} \big[r_i + \gamma\, v(s'_i)\big]$$

See how we can estimate the model by using experiences only on the following example :

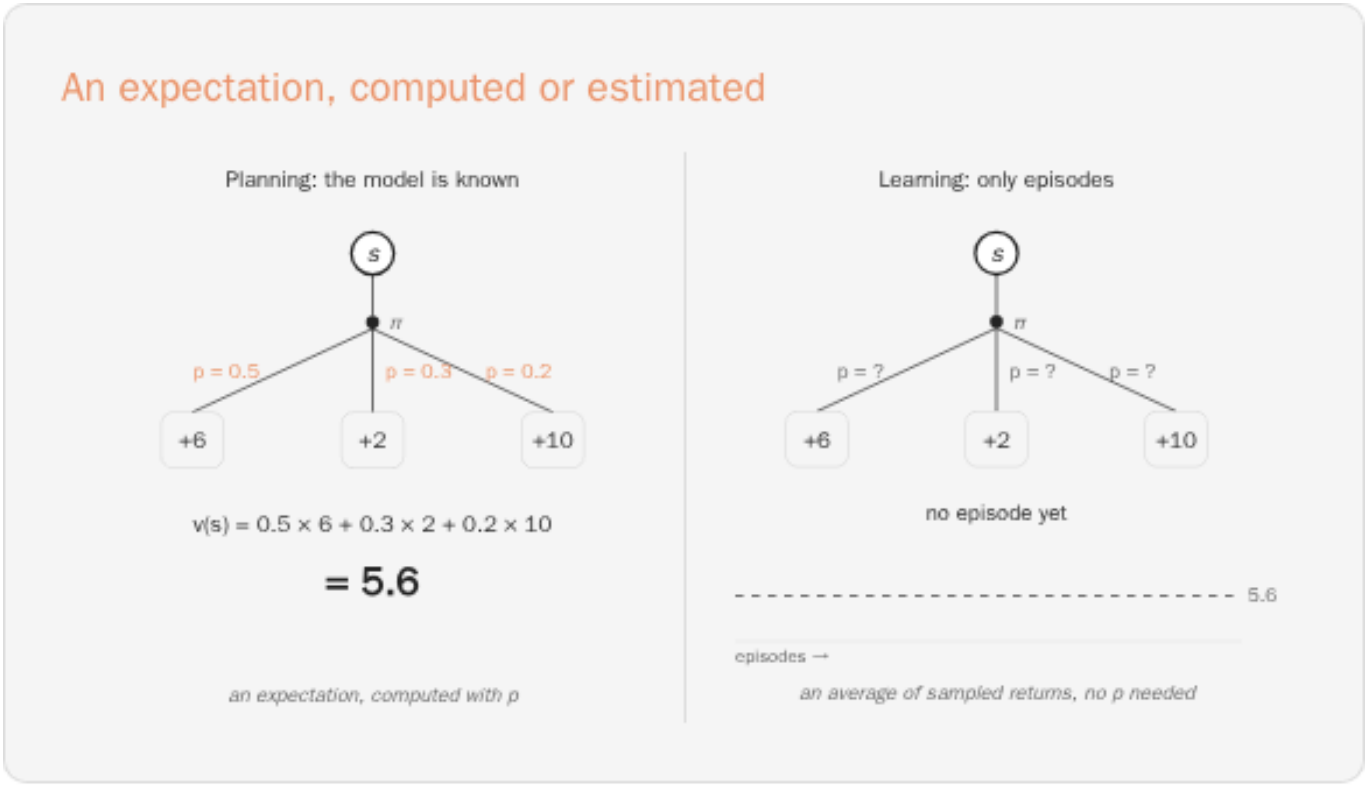


Figure 1 · One state, one action, three possible outcomes. Left, planning: the model gives the probability of each outcome, and the value is their weighted sum, computed exactly. Right, learning: the probabilities are unknown, but each episode draws one outcome; the average of the returns observed so far (orange) approaches the same value (dashed) as episodes accumulate.

When applying this idea to our problems, we could estimate p from a lot of interactions. But this would require a giantic amount of time to cross each (s, a) pair multiple times. So instead of finding the dynamics p and then solving the problem, we keep the steps from GPI, and each one slightly differ:

- Prediction, Part 3.1, this page: the input is a policy π and episodes played with π; the output is $V \approx v_\pi$. The expectation over p becomes an average of samples.
- Control, Part 3.2: the input is episodes of interaction; the output is $Q \approx q_*$ and $\pi \approx \pi_*$, the optimal policy is obtained from it.

One key idea for both: as we need many episodes, the environment must be fast to evaluate, we trade prior knowledge against fast interactions.

We will study:

1. Monte Carlo evaluation: the value of a state as the average of the returns that followed.
2. Temporal-difference evaluation: update after every step, from the next estimate.
3. Monte Carlo or TD? How these two give different answers with the same data.

The control problem, i.e. finding a good policy without the model, is the subject of Part 3.2, *Learning to Act*: Monte Carlo control, SARSA, and Q-learning.

1. Monte Carlo: learning from complete episodes

Monte Carlo methods aim at estimating a dynamics based on a set of observations drawn from a probability distribution.

1.1 An average of returns

Monte Carlo methods are not specific to reinforcement learning. Whenever an expectation can't be computed analytically, one can draw random samples and average them.

Formally, with $X_1, X_2, \dots$ independent copies of a random variable X,

$$\hat{\mu}_N = \frac{1}{N}\sum_{i=1}^{N} X_i \quad \xrightarrow[N\to\infty]{} \quad \mathbb{E}[X]$$

with probability one: this is the law of large numbers. The estimate is unbiased, $\mathbb{E}[\hat{\mu}_N] = \mathbb{E}[X]$, and its error shrinks like $\sigma/\sqrt{N}$, where σ is the standard deviation of X. It is used to price options, compute integrals in high dimension, or propagates uncertain inputs through finite-element models.

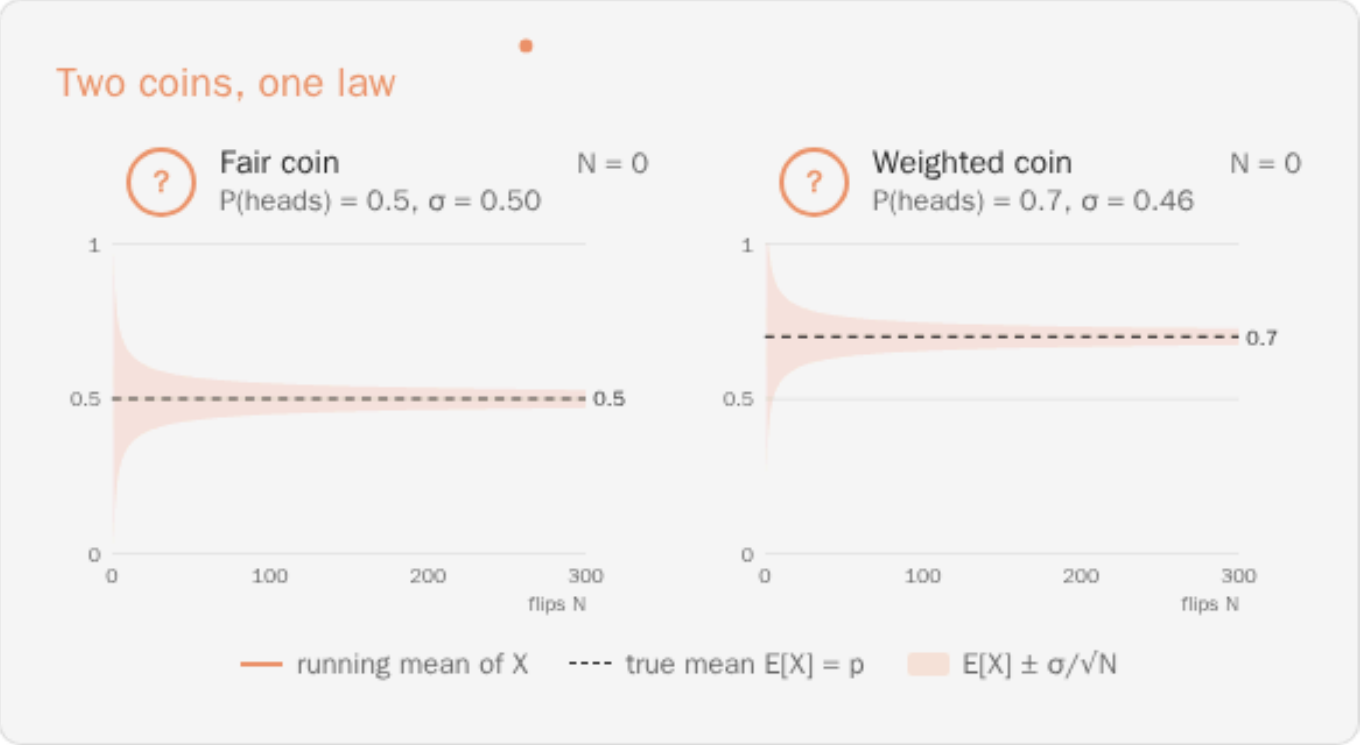


Figure 2 · A fair coin (P(heads) = 0.5) and a weighted one (P(heads) = 0.7), flipped live, with X = 1 for heads. The running mean wanders at first, then settles on E[X] (dashed), inside a funnel E[X] ± σ/√N that narrows like 1/√N.

In reinforcement learning, the expectation we want is the value $v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$, and the samples are the returns.

The recipe is thus to play episodes with π:

$$S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_T$$

and, every time the agent is in s, write down the return that followed, $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T$.

Monte Carlo policy evaluation estimates $v_\pi(s)$ by the average of these returns: the empirical mean instead of the true expected return. No model is needed, only episodes.

But the episodes must end: G_t is known only once the episode has terminated, so Monte Carlo works for episodic tasks only.

The average does not need to store the returns. After each new return G_t from s, it can be updated in place, as an average written step by step:

$$N(s) \leftarrow N(s) + 1, \qquad V(s) \leftarrow V(s) + \frac{1}{N(s)}\big[G_t - V(s)\big]$$

This incremental form is what the algorithms below use.

1.2 Evaluating V at the first visit or every visit

A state can come back several times in the same episode. To estimate its value fo a given policy, First-visit MC keeps only the return after the first visit to s in each episode; every-visit MC keeps them all. See on the example below :

Algorithm Monte Carlo prediction (first visit / every visit)

Input: a policy π to evaluate

$V(s) \leftarrow 0$ and $N(s) \leftarrow 0$ for every state s
repeat for each episode
  generate an episode with π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  for $t = T-1, T-2, \ldots, 0$, backward in time
    $G \leftarrow \gamma G + R_{t+1}$
    if S_t does not appear in $S_0, \ldots, S_{t-1}$
      $N(S_t) \leftarrow N(S_t) + 1$
      $V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}\left[G - V(S_t)\right]$

First visit | Every visit

γ = 1

Initialize V(s) = 0 and N(s) = 0 for every state

s	N(s)	V(s)	returns averaged
A	0	0	–
B	0	0	–

Pause | Step | Restart

Two episodes with γ = 1: A →(1)→ B →(0)→ A →(2)→ end, then B →(1)→ A →(2)→ end. The backward pass builds each return, $G \leftarrow \gamma G + R_{t+1}$. In first-visit mode, the return after the second visit to A fails the test and is skipped: V(A) = 2.5, the average of 3 and 2. Switch to every visit: the test is struck out, and V(A) averages 2, 3 and 2, about 2.33. V(B) = 2.5 in both.

Every-visit MC is the same algorithm without the if: switch the figure to *every visit* to see it run.

Running backward builds every return of the episode in a single pass (forward pass would require $O(n^2)$ steps), since $G_t = R_{t+1} + \gamma\, G_{t+1}$. The update with $\frac{1}{N}$ is the incremental form of section 1.1: the running average of the returns, computed without storing them.

For example, with $\gamma = 1$, on an episode that visits A twice, the two version give:

$$A \overset{R_1=1}{} \quad B \overset{R_2=0}{} \quad A \overset{R_3=2}{} \quad \text{end} \qquad \Rightarrow \qquad G_0 = 1 + 0 + 2 = 3, \quad G_2 = 2$$

First-visit MC adds only the return 3 to the average of $V(A)$. Every-visit MC adds 3 and 2. Both converge to $v_\pi(A)$ as the number of visits grows, but they don't behave the same on the way: the next exercise shows how.

Exercise 1.2 First visit or every visit? Reason Show ›

1.3 A constant step size

In the incremental form of section 1.1, $\frac{1}{N(s)}$ comes from the definition of the mean, but replace it by a constant step size α, and you can control how older returns are gradually forgotten, which is useful when the policy or the environment drifts:

$$V(s) \leftarrow V(s) + \alpha \left[G_t - V(s)\right]$$

Read it as: trust the new sample by a fraction α of the change.

On the equation G_t is called the target.

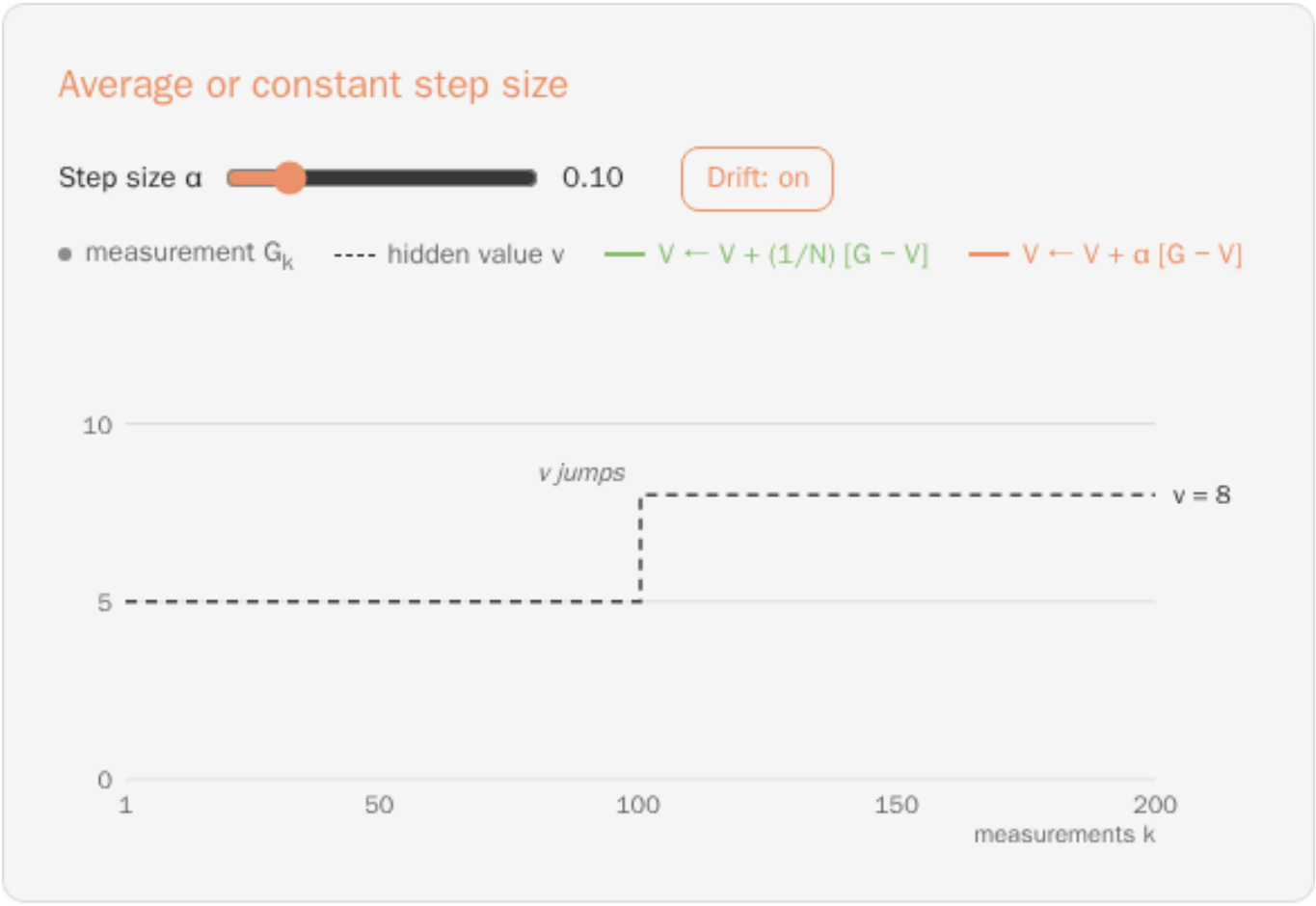


Figure 4 · Noisy measurements G_k ($\sigma = 2$) of a hidden value v, and two incremental estimates starting from 0. The exact average $V \leftarrow V + (1/N)[G - V]$ (green) is smooth, but after v jumps from 5 to 8 it barely moves, because it remembers every measurement. The constant step size $V \leftarrow V + \alpha[G - V]$ (orange) follows the jump, more noisily the larger α is.

This form general form of the equation will come back in every algorithm of this page and of Part 3.2; only the target changes.

Monte Carlo prediction has three properties to keep in mind for what follows:

- Unbiased: G_t is a true sample of $v_\pi(s)$.
- High variance: G_t depends on every random event of the episode, every action and every transition until the end.
- Offline: nothing is learned before the episode is over.

An example: blackjack. Writing $p(s', r \mid s, a)$ for every sequence of cards is tedious; simulating games is easy, which makes blackjack the classic Monte Carlo example. Evaluating the policy “stick on 20 or 21” over its 200 states, the value surface is still noisy after 10,000 games and smooth after 500,000: the high variance, in numbers.

WHAT TO LOOK AT Blackjack, evaluated by Monte Carlo: the value surfaces after 10,000 and 500,000 games ›

OPTIONAL Monte Carlo as an observer: a control-engineering reading of the update Reason ›

2. Temporal differences: learning from every step

The main drawback with Monte Carlo estimation is that one needs the fulln episode to compute the estimation. This section introduces Temporal Difference or TD that precisely aims at solving this issue with a technique called bootstrapping.

2.1 Bootstrapping

After each episode, Monte Carlo update pulls $V(S)$ towards the return,

$$V(S) \leftarrow V(S) + \alpha \left[G - V(S)\right] \tag{MC}$$

Its problem is that G is known only at the end of the episode. But recall that the return is recursive between two consecutive timesteps, and its second part has a name:

$$G_t = R_{t+1} + \gamma G_{t+1}, \qquad \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1}\right] = v_\pi(S_{t+1})$$

Taking the expectation of the first equation, the value of a state is written with the value of the next one:

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

So after a single step, we can replace the unknown G_{t+1} by our current estimate of its expectation, $V(S_{t+1})$. Updating an estimate from another estimate is called bootstrapping, and it gives the so-called TD(0) update:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[\underbrace{R_{t+1} + \gamma\, V(S_{t+1})}_{\text{target}} - V(S_t)\right]$$

$$\text{error}$$

For a given state S_t being the state we met at time t, the TD target $R_{t+1} + \gamma\, V(S_{t+1})$ is where $V(S_t)$ is pulled to: one measured reward, then the estimate of what follows.

The difference between the target and the current estimate is the TD error,

$$\delta_t = R_{t+1} + \gamma\, V(S_{t+1}) - V(S_t)$$

a measure of novelty: how much better ($\delta_t > 0$) or worse ($\delta_t < 0$) the step went than the estimate predicted. The “(0)” of TD(0) means one real reward before bootstrapping; n-step TD and TD(λ) look further ahead (section 3).

TD(0) takes the best of the two previous worlds. It samples, like Monte Carlo: no model is used but it uses samples from what the agent really experienced. And it bootstraps, like dynamic programming: its uses the current estimate in the target. Like DP, it updates after every transition, so it learns during the episode, and it does not even need the episode to end.

See how TD(0) can compute the value function on all states of a 3*3 grid example below :

Algorithm TD(0) prediction

Input: a policy π to evaluate, a step size $\alpha \in (0, 1]$

1 $V(s)$ arbitrary for every state s, with $V(\text{terminal}) = 0$
2 repeat for each episode
3 initialize S
4 repeat until S is terminal
5 take the action A given by π in S; observe R and S'
6 $V(S) \leftarrow V(S) + \alpha \left[R + \gamma V(S') - V(S)\right]$
7 $S \leftarrow S'$



policy π: a random move
R = −1 per move · γ = 1 · α = 0.5

move A from π, R, S′

target R + γV(S′)

error δ = target − V(S)

update V(S) ← V(S) + αδ

V(s) = 0 everywhere, the exit included

Pause Step Restart Fast ×4

TD(0) on a 3×3 grid, with the line being run highlighted. The agent starts top-left and moves at random until the exit, bottom-right; every move costs R = −1, γ = 1, α = 0.5. After each move, V(S) moves halfway to the TD target R + γV(S′): one real reward, then the estimate of the cell reached. The first episode runs line by line, the next ones faster.

There is no strict criterion of stopping as this process could run forever. We draw a subsequent number of runs that we consider meaningful.

2.2 TD(0) on a random walk

The random walk is a chain of five states, A to E, between two exits. Every episode starts in C; the policy is a random move at each step, *left* or *right*; the reward is $+1$ on the right exit and 0 everywhere else, with $\gamma = 1$. The value of a state is then the probability of leaving on the right: $v_\pi = \frac{1}{6}, \frac{2}{6}, \ldots, \frac{5}{6}$ from A to E.

Start with $V = 0.5$ for A to E, $V(\text{end}) = 0$, and $\alpha = 0.1$. Suppose the random policy takes *right* three times: $C \to D \to E \to \text{end}$.

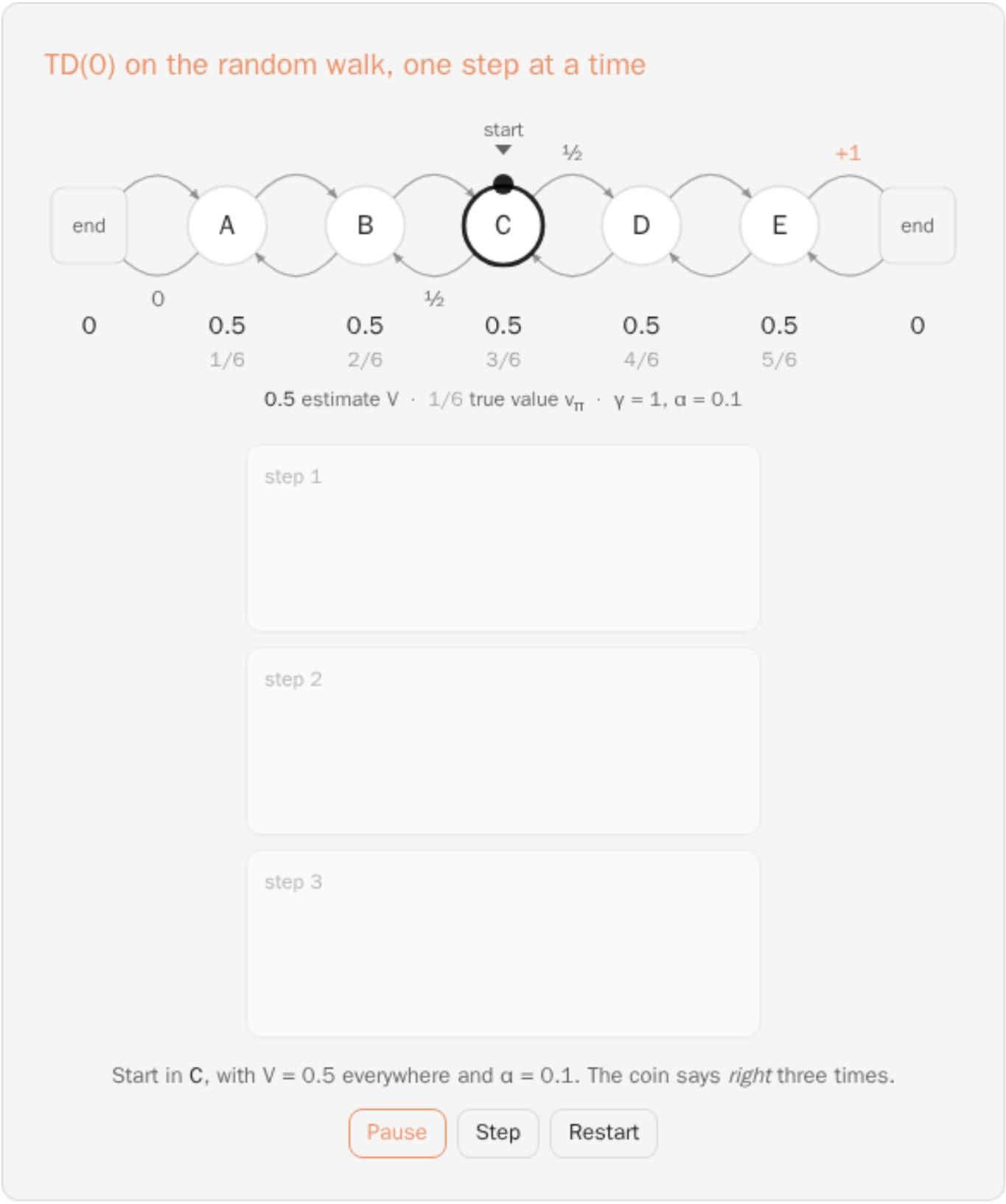


Figure 6 · The random walk: five states between two exits, start in C, left or right with probability ½, reward +1 only on the right exit, γ = 1. Under each state, its estimate V (bold) and its true value v_π (faint). The agent plays the episode of the text, C → D → E → end, with V = 0.5 everywhere and α = 0.1; each card shows the TD target, the TD error δ and the update of one step.

#does it make sense to have the 3 3 grid and this? very similar examples?

Only $V(E)$ moves. The first two steps held no surprise: the estimates said 0.5, and the next state also said 0.5. The reward is felt only at the last step, and it travels back one state per episode. Monte Carlo, on the same episode, would have pulled

C, D and E all at once towards the return, 1.

Exercise 2.2 TD(0) by hand Compute Show

3. Monte Carlo and TD comparison

Let's quickly review the practical differences these algorithms have in practice as they influence different family of RL solutions.

3.1 Estimating one the way : Driving home example

An everyday example makes the computing difference concrete. Each evening, leaving the office, you predict how long it will take to get home, and you revise the prediction on the way. The reward of each leg is the minutes it took; the value of a state is the predicted number of minutes still to go, with $\gamma = 1$. The figure shows the elapsed time, the prediction of what remains, and their sum, the predicted total for both algorithms:

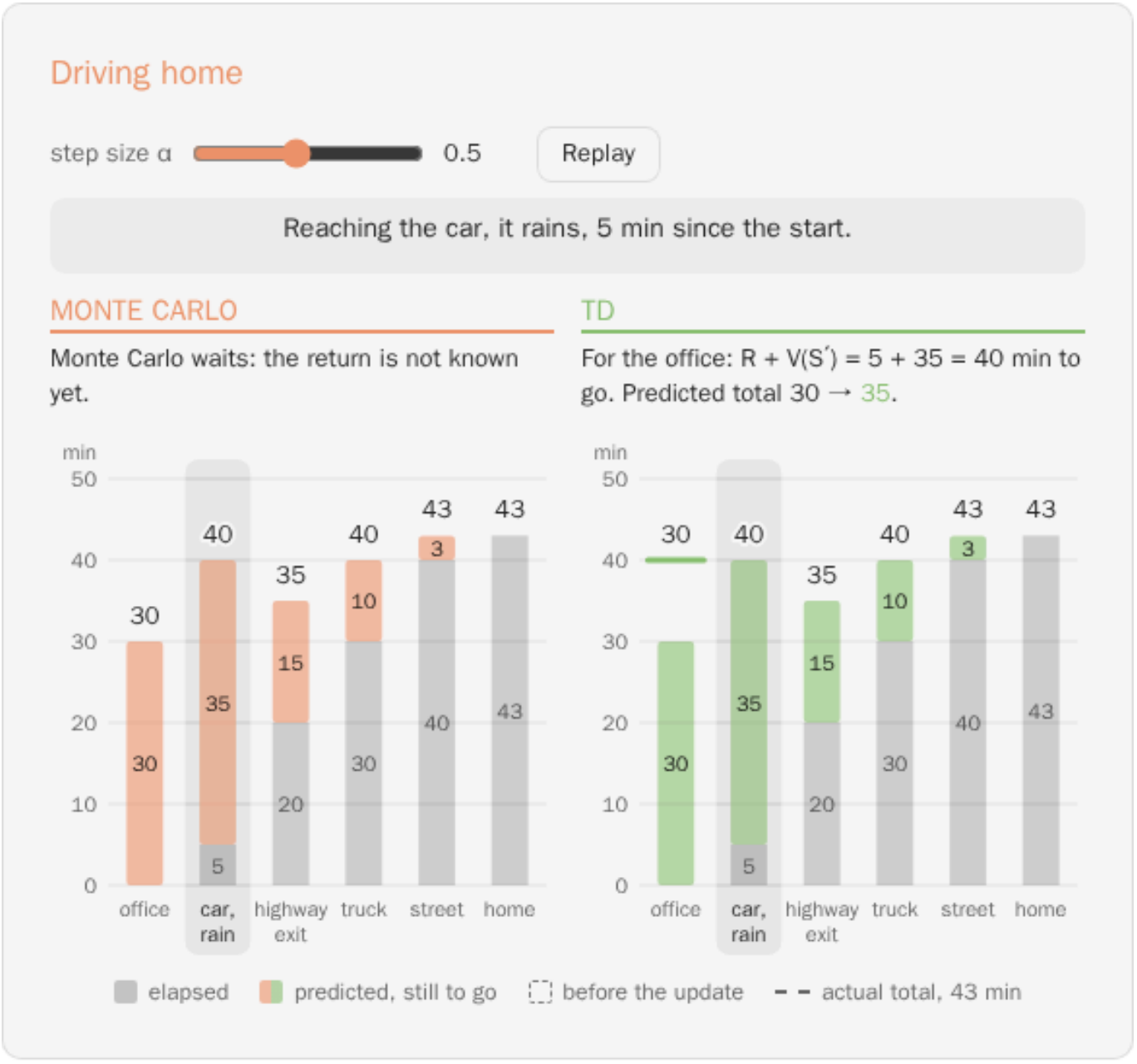


Figure 7 · Each bar is a state of the trip: time elapsed (gray) plus predicted time still to go, which sum to the predicted total. Both methods run on the same trip, side by side. Monte Carlo (left) waits until home and moves every prediction toward the actual 43 minutes; TD (right) moves each prediction toward the next one, on the way. Change the step size α (the book uses α = 1).

Monte Carlo waits until you are home. Then every prediction of the trip moves towards the actual total, 43 minutes: you needed the whole episode to learn anything.

TD learns on the way: each prediction moves towards the next one. At the car, the “30 minutes” of the office already becomes “40”, because it rains, before you know how the trip ends.

3.2 Bias and variance

The two methods both correspond to different tradeoffs on various aspects :

	Monte Carlo	TD(0)
update	✗ at the end of the episode	✓ after every step
bias	✓ none	✗ yes: it bootstraps on estimates
variance	✗ high	✓ low
continuing tasks	✗ no	✓ yes
sample efficiency	✗ lower	✓ usually higher
converges to v_π	✓ yes	✓ yes

Monte Carlo's target, G_t, is a true sample of $v_\pi(S_t)$, so it is unbiased; but it sums every random event until the end of the episode, so it is noisy. TD's target, $R_{t+1} + \gamma V(S_{t+1})$, contains a single random reward and a single random transition, so it varies much less; but $V(S_{t+1})$ is only an estimate, wrong at the beginning, so the target is biased.

In one sentence : Monte Carlo pays high variance to be unbiased; TD pays some bias for low variance, and learns online.

Watch it on random-walk example :

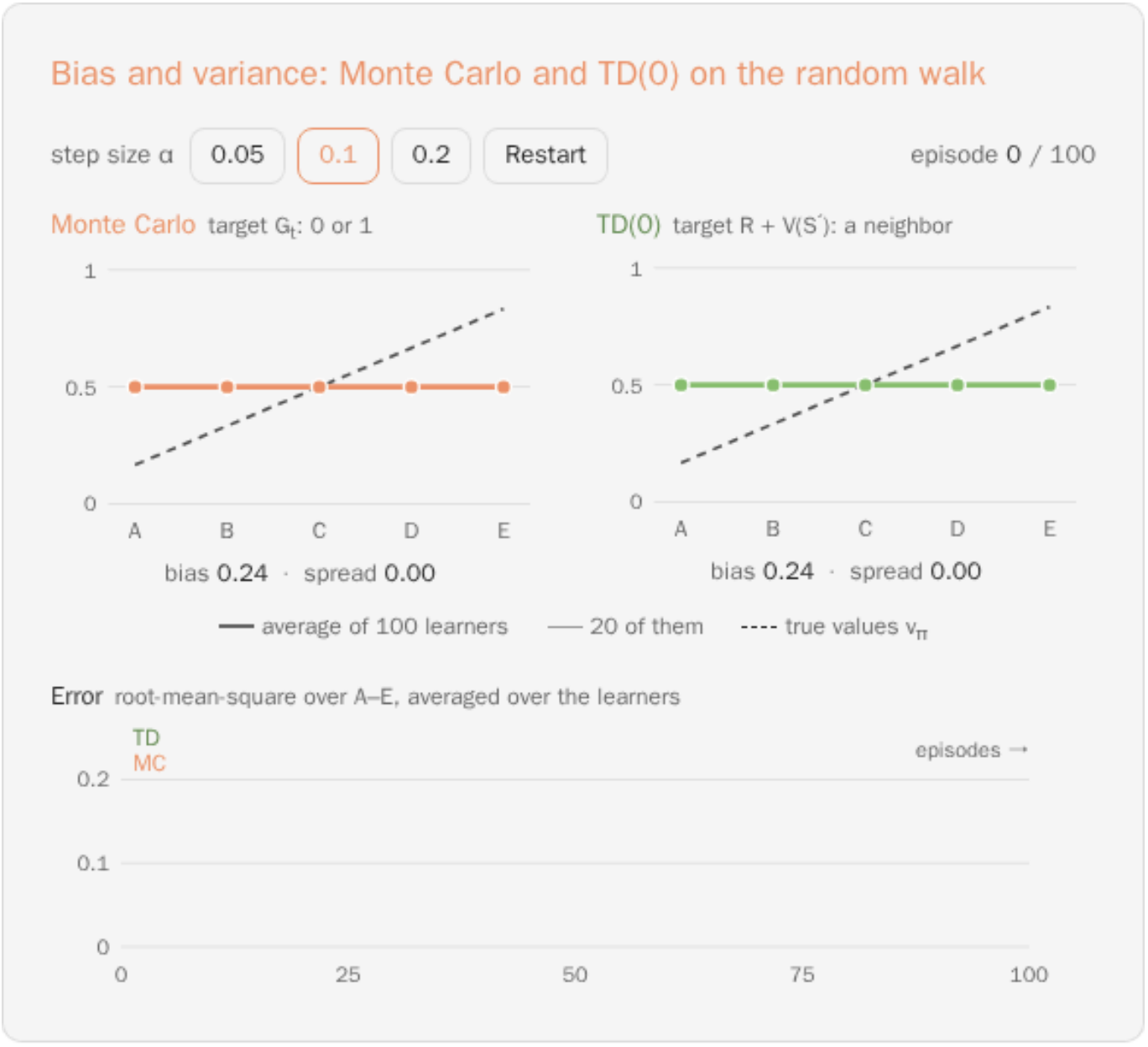


Figure 8 · 100 learners per method, each on its own random episodes of the random walk (Monte Carlo and TD(0) see the same ones), all starting from V = 0.5. Thin lines: 20 learners, whose spread is the variance; bold line: their average, whose gap to the true values (dashes) is the bias. Monte Carlo's targets are 0 or 1, so its learners scatter; TD's targets are a neighbor's estimate, so they stay together, and early on they share the same error. Below, the error, episode after episode.

Early on, Monte Carlo's average is closer to the truth, but each of its learners is far from it; TD's learners agree with each other, on a value that is wrong at first. After a few dozen episodes, the low variance wins: TD's error is lower, at every step size.

3.3 RL in one plot: width and depth

Dynamic Programming (DP), Monte Carlo and TD all update $V(s)$ towards a target. They differ in how they build it, among two aspects:

- Width: does the target average over every possible action and next state, which needs the model, or does it use the transition that really happened?

- Depth: does it look one step ahead and then bootstrap on an estimate, or does it go to the end of the episode?

Dynamic programming is full width, one step deep: $V(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\,[r + \gamma V(s')]$, no sampling noise but the model is required.

TD(0) is one sample, one step deep: online, low variance, biased.

Monte Carlo is one sample, full depth: it never bootstraps, it is unbiased, and it needs complete episodes.

The fourth corner, full width and full depth, is exhaustive search (which is of course not an appealing solution 😁).

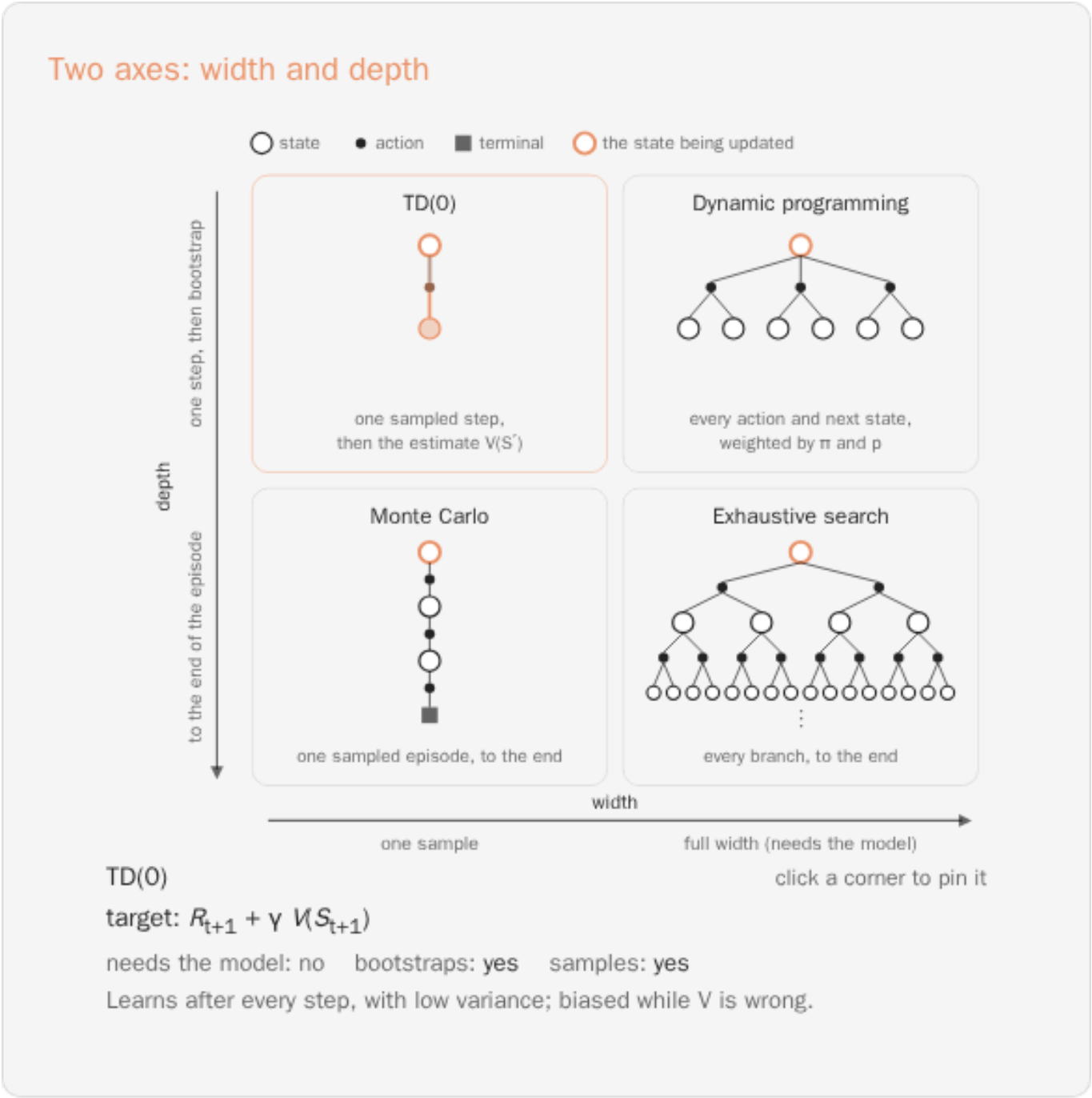


Figure 9 · The backup diagram of each method: from the state being updated, the transitions its target reads, lit from the leaves back to the root. Left column, one sampled transition: no model needed. Right column, every action and next state, weighted by π and p: the model is needed. Top row, one step, then the current estimate: these methods bootstrap. Bottom row, to the end of the episode: no bootstrapping. Click a corner for its target.

The next exercise shows that the difference is not only about speed: on the same data, Monte Carlo and TD can give different answers, and it is worth knowing which one to trust.

Exercise 3.3 You are the predictor Compute Show

OPTIONAL Between TD(0) and Monte Carlo: n-step returns and TD(λ) Reason

4. Quick wrap-up before 3.2

In a few lines:

- To predict without a model: replace the expectation over p by an average over experience.
- Monte Carlo averages the returns: unbiased, high variance, needs the end of the episode.
- TD updates after every step, from the next estimate: biased, low variance, online.
- Both are the same update formula, an estimate moved by a step toward a target; only the target changes.
- These approaches are tractable only if the environment is fast to evaluate: learning takes many episodes, so on a simulator, not on the real system.

Part 3.2: Learning to Act uses these estimates to find a good policy, still without a model: exploration, Monte Carlo control, SARSA and Q-learning.

Citation

For attribution in academic contexts, please cite this work as

```
Etienne Chassaing (2026). "Learning from Experience".
```

BibTeX citation

```
@misc{chassaing2026_learning_from_experience,
  title={Learning from Experience},
  author={Etienne Chassaing},
  year={2026},

}
```

Reuse

Text and figures are licensed under CC-BY 4.0. Built with the Research Article Template (Thibaud Frère, CC-BY 4.0) and the exercise components of NLP before LLMs (David Louapre).

Made with ❤️ with research article template