

Temporal-Difference Learning

Prediction, Control, and Eligibility Traces

Farnaz Adib Yaghmaie

Reinforcement Learning Lecture

13 August 2026

Core questions

- ▶ Why learn online, one step at a time?
- ▶ What is the TD error and why is it the central learning signal?
- ▶ What is the difference between prediction and control?
- ▶ How do on-policy and off-policy methods differ?

Today's arc

1. Motivation and Bellman recursion
2. TD prediction: $TD(0)$, MC vs TD, n -step, $TD(\lambda)$
3. TD control: SARSA, Expected SARSA, Q-learning
4. Maximization bias and Double Q-learning
5. Convergence and practical remarks



Temporal-Difference Learning

Why TD?

Three routes to value learning

Method	Needs model?	Waits for episode end?	Bootstraps?
Dynamic programming	Yes	No	Yes
Monte Carlo	No	Yes	No
Temporal difference	No	No	Yes

Key idea

TD learns directly from experience without a model, but updates *before* an episode ends by using the current value estimate as a target.

Bellman expectation equation

For a fixed policy π the value function satisfies

$$v^\pi(s) = \sum_a \pi(a | s) \sum_{s', r} p(s', r | s, a) [r + \gamma v^\pi(s')].$$

- ▶ The value equals an immediate reward plus discounted next-state value.
- ▶ TD exploits this recursion without knowing $p(s', r | s, a)$.
- ▶ This is why TD is called a **bootstrapping** method: it uses one estimate to update another.

From Bellman recursion to a one-step update

After observing transition $(S_t, A_t, R_{t+1}, S_{t+1})$ under policy π :

$$Y_t^{\text{TD}} = R_{t+1} + \gamma \hat{v}(S_{t+1}).$$

The update rule moves the current estimate toward that target:

$$\hat{v}(S_t) \leftarrow \hat{v}(S_t) + \alpha(Y_t^{\text{TD}} - \hat{v}(S_t)).$$

Comparison with Monte Carlo

MC uses the actual return G_t as the target and waits until the episode ends. TD uses a *one-step estimate* and updates immediately.



Temporal-Difference Learning

Prediction

TD(0) prediction

For state-value prediction under a fixed policy π :

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

- ▶ δ_t is the **temporal-difference error**.
- ▶ Positive δ_t : the successor was better than expected — raise $V(S_t)$.
- ▶ Negative δ_t : the current estimate was too optimistic — lower $V(S_t)$.
- ▶ The step-size $\alpha \in (0, 1]$ controls how fast estimates change.

What the TD error means

Signal for learning

$$\delta_t = \underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{target}} - \underbrace{V(S_t)}_{\text{prediction}}$$

Positive error: raise the prediction. Negative error: lower it.

- ▶ The update is local and immediate — no waiting.
- ▶ Each visit propagates information one step backward in the trajectory.

Analogy

Think of a weather forecast corrected after observing the next hour of data, without waiting for tomorrow.

Note

δ_t is also the foundation of actor-critic methods and many other RL algorithms.

Monte Carlo vs TD(0)

Property	Monte Carlo	TD(0)
Target	G_t (full return)	$R_{t+1} + \gamma V(S_{t+1})$
Update timing	End of episode	Every step
Variance	Higher	Lower
Bias	None	Due to bootstrapping
Online use	No	Yes
Continuous tasks	No	Yes

Tradeoff

MC is unbiased but high variance and slow. TD is biased but low variance and naturally online. Neither dominates in all settings.

n -step TD: interpolating MC and TD(0)

Delay the bootstrap by n steps before using an estimate:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}).$$

- ▶ $n = 1$: standard TD(0).
- ▶ $n \rightarrow \infty$: converges to Monte Carlo.
- ▶ Intermediate n : practical sweet spot in many problems.

Design choice

n trades off bias and variance. Small n gives low variance but higher bias; large n reduces bias but increases variance.

The λ -return G_t^λ

Instead of choosing a single n , take a weighted average of *all* n -step returns:

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}, \quad \lambda \in [0, 1].$$

- ▶ The factor $(1 - \lambda)$ normalises the weights so they sum to 1.
- ▶ $\lambda = 0$: all weight on the 1-step return $\Rightarrow$ TD(0).
- ▶ $\lambda = 1$: all weight on the full return $\Rightarrow$ Monte Carlo.

Problem

Computing G_t^λ at time t requires all *future* rewards. It is a **forward view** — elegant theoretically, but awkward online.

Eligibility traces: the backward view

The eligibility trace $e_t(s)$ records how recently and frequently state s was visited:

$$e_0(s) = 0, \quad e_t(s) = \gamma\lambda e_{t-1}(s) + 1[S_t = s].$$

- ▶ Each visit to s spikes the trace by 1; otherwise it decays by $\gamma\lambda$ per step.
- ▶ Recent, frequent visits $\Rightarrow$ large trace; old visits $\Rightarrow$ nearly zero.

At every step, apply the one-step TD error to *all* states weighted by their trace:

$$V(s) \leftarrow V(s) + \alpha \delta_t e_t(s) \quad \forall s.$$

Key theorem (Sutton, 1988)

The online trace update above produces *exactly* the same parameter changes as using G_t^λ as the offline target, on the same trajectory.

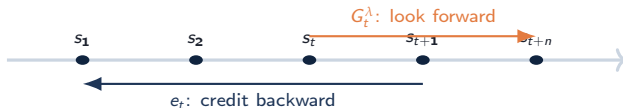
Forward view vs backward view

Forward view: G_t^λ

- ▶ Look *forward* in time from t .
- ▶ Weight future n -step returns by λ^{n-1} .
- ▶ Requires full trajectory before updating.
- ▶ Theoretical motivation.

Backward view: $e_t(s)$

- ▶ Look *backward* at which states led here.
- ▶ Positive δ_t credits recent states proportionally.
- ▶ Updates every step, online.
- ▶ Practical implementation.



Bottom line

G_t^λ and $e_t(s)$ are two views of the same algorithm. Use G_t^λ to understand; use $e_t(s)$ to implement.



Temporal-Difference Learning

Control

From prediction to control

To improve a policy without a model, learn *action* values:

$$q^{\pi}(s, a) = \mathbb{E}_{\pi}[G_t \mid S_t = s, A_t = a].$$

Then act greedily with respect to Q :

$$\pi(s) = \arg \max_a Q(s, a).$$

Exploration is mandatory

Purely greedy control stops visiting useful actions too early. We must maintain exploration to ensure convergence.

ϵ -greedy exploration

At each state s :

$$\pi(a | s) = \begin{cases} 1 - \epsilon + \epsilon / |\mathcal{A}(s)|, & a = \arg \max_b Q(s, b), \\ \epsilon / |\mathcal{A}(s)|, & \text{otherwise.} \end{cases}$$

- ▶ Small ϵ : mostly exploit the current best action.
- ▶ Large ϵ : more exploration, slower convergence.
- ▶ Decaying $\epsilon_t \rightarrow 0$: asymptotically greedy.

Convergence condition

For convergence, every state-action pair must be visited infinitely often. ϵ -greedy with $\sum_t \epsilon_t = \infty$ and $\sum_t \epsilon_t^2 < \infty$ satisfies this.

SARSA: on-policy TD control

Update uses the action *actually taken* at the next step:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)].$$

- ▶ Named after the quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$.
- ▶ Learns the value of the *current* behaviour policy (including exploration).

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Choose A from S using policy derived from Q (e.g., ε -greedy)

 Loop for each step of episode:

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ε -greedy)

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$

$S \leftarrow S'; A \leftarrow A';$

 until S is terminal

Expected SARSA

Replace the sampled next action with its expectation under the behaviour policy:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a | S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right].$$

Advantage over SARSA

Lower variance because the expectation averages out the randomness of the next action.

Cost

Requires summing over all actions at each step — feasible only for small discrete action spaces.

Q-learning: off-policy TD control

Update uses the *greedy* next action, regardless of what was taken:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)].$$

- ▶ The behaviour policy can be exploratory (ϵ -greedy).
- ▶ The target policy is always greedy.
- ▶ Q-learning learns the optimal value function regardless of how data is collected.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Loop for each step of episode:

 Choose A from S using policy derived from Q (e.g., ε -greedy)

 Take action A , observe R, S'

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

 until S is terminal

SARSA vs Q-learning

Aspect	SARSA	Q-learning
Policy type	On-policy	Off-policy
Target next action	Sampled from π	Greedy $\max_a$
Learns value of	Current behaviour policy	Optimal policy
Typical behaviour	Conservative (risk-aware)	Aggressive
Safe-path example	Learns to avoid cliffs	May cut close to cliffs

Teaching point

In risky environments SARSA often learns safer behaviour because the exploratory actions taken during learning are reflected in the target.



Temporal-Difference Learning

Maximization Bias

Why Q-learning tends to overestimate

The update target uses:

$$\max_a Q(S_{t+1}, a).$$

The problem

If the Q -values are noisy, taking the maximum over noisy estimates tends to pick a positive error more often than a negative one. This leads to **overestimation bias**.

- ▶ In early learning, all Q -values are approximations and noise is high.
- ▶ The bias is amplified when state-action pairs are visited infrequently.
- ▶ It can cause slow convergence or suboptimal policies.

Double Q-learning

Separate *action selection* from *action evaluation* using two independent value tables Q^A and Q^B .

Q-learning target:

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

Both selection and evaluation use the same Q — hence the bias.

Double Q-learning target:

$$R_{t+1} + \gamma Q^B(S_{t+1}, \arg \max_a Q^A(S_{t+1}, a))$$

Selection uses Q^A , evaluation uses the independent Q^B .

Result

Because Q^A and Q^B are estimated from different data, their errors are less correlated, and the systematic upward bias is substantially reduced.

Practical note

Double Q-learning converges to q^* under the same conditions as Q-learning.



Temporal-Difference Learning

Convergence

When does tabular TD converge?

TD(0) prediction

Converges to v^π with probability 1 when:

- ▶ every state is visited infinitely often, and
- ▶ step sizes satisfy $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$.

Q-learning control

Converges to q^* with probability 1 under the same step-size conditions, provided every state-action pair is visited infinitely often.

SARSA control

Converges to q^* if both the policy and step sizes satisfy the Robbins-Monro conditions and the policy becomes greedy in the limit.

Key point

These proofs apply to *tabular*, *finite-state* settings. Function approximation changes the convergence picture significantly — that is the topic of Deep TD.

Why tabular TD matters as a foundation

- ▶ Tabular TD methods are provably correct and well-understood.
- ▶ They define the target behaviour that deep RL attempts to replicate at scale.
- ▶ Every deep value-based algorithm is a principled approximation of a tabular idea.

Bridge to the next lecture

When the state space is too large for a table, we replace the table with a neural network. That step introduces instability and requires additional mechanisms — covered in the Deep TD lecture.



Temporal-Difference Learning

Takeaways

What to remember

1. TD updates before the episode ends by bootstrapping from the current value estimate.
2. The TD error δ_t is the core learning signal for both prediction and control.
3. n -step returns and $TD(\lambda)$ interpolate between $TD(0)$ and Monte Carlo.
4. SARSA is on-policy; Q-learning is off-policy.
5. Q-learning converges to the optimal policy, but suffers from maximization bias.
6. Double Q-learning removes that bias by separating action selection and evaluation.

For class

- ▶ When would Monte Carlo be preferable to TD, despite slower feedback?
- ▶ Why can SARSA produce safer behaviour than Q-learning in a hazardous environment?
- ▶ What would happen if ϵ never decays in SARSA?
- ▶ How does eligibility trace parameter λ affect what gets updated after a rewarded action?
- ▶ What breaks when we replace the tabular Q with a neural network?

The TD error is the engine of
online value learning.

One step at a time, bootstrapping toward the optimum.