PAPER WALKTHROUGH

Introduction to Reinforcement Learning

Ghasemi & Ebrahimi, 2024 — foundations, algorithm families, and one value-based and one policy-based algorithm, worked end to end on one grid.

arXiv:2408.07712 · Wilfrid Laurier University

Contents

01 Foundations — states, returns, values, Bellman

02 Method families — model-free, on- and off-policy

03 Q-learning — the grid, the rule, three steps

04 Proximal policy optimization — the same grid, a policy not a table

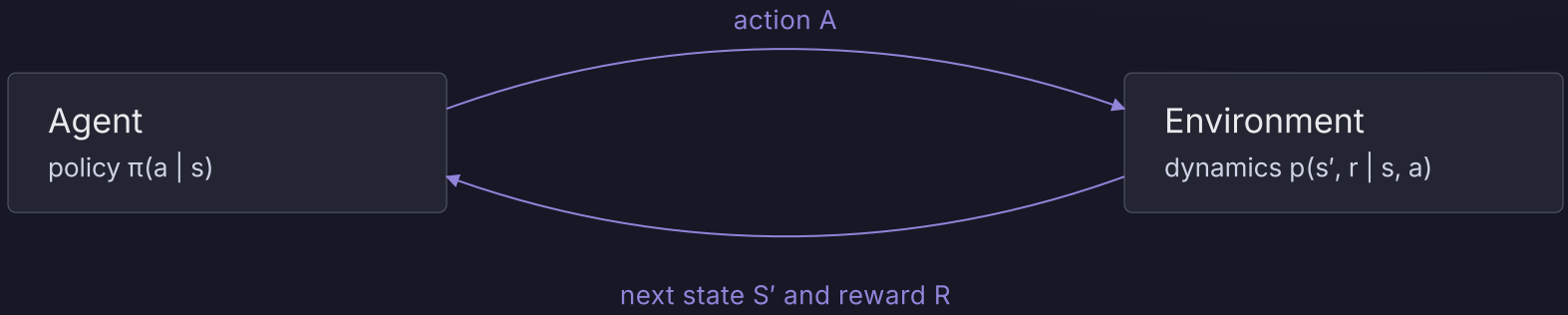
05 Resources — where the paper points next

01

Foundations

Sections 1 and 2 of the paper: the agent–environment loop, MDPs, returns, value functions and the Bellman equations.

The agent–environment loop



The agent observes a state, selects an action under its policy, and receives a reward and a successor state. Learning happens entirely inside that feedback, by trial and error, without labelled data.

The Markov decision process

THE FIVE-TUPLE

$$\mathcal{M} = (\mathcal{S},\ \mathcal{A},\ \mathcal{P},\ \mathcal{R},\ \gamma)$$

State space, action space, transition dynamics, reward function, and the discount factor γ ∈ [0, 1].

DYNAMICS

$$p(s', r \mid s, a) \doteq \Pr\{ S_t = s',\ R_t = r \mid S_{t-1} = s,\ A_{t-1} = a \}$$

Markov: the outcome depends on the current state and action alone, not on the history that led there — so state-transition probabilities and expected rewards all follow from this one function.

Core vocabulary

States and actions

A state is the configuration of the environment the agent perceives; an action is a move available in it. Either can be discrete (a chess position) or continuous (a robot's coordinates).

Reward

The scalar signal that defines the goal at every step, separating good outcomes from bad. Deterministic or stochastic, and always a function of state and action.

Policy

The map from states to actions — a table, a function, or a neural network. Deterministic, $\pi(s) = a$, or stochastic, $\pi(a \mid s)$, a distribution the agent samples from.

Dynamics and model

Transition dynamics $p(s' \mid s, a)$ say where actions lead. An agent that learns or is given them is model-based; one that learns only from experience is model-free.

Return and discounting

DISCOUNTED RETURN

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k \geq 0} \gamma^k R_{t+k+1}$$

RECURSIVELY

$$G_t = R_{t+1} + \gamma\, G_{t+1}$$

One form covers both task types: episodic tasks end at a terminal state, continuing tasks never do and rely on γ < 1 to keep the sum finite. At γ = 0 the agent maximizes only the next reward; as γ approaches 1 distant rewards count almost as much as immediate ones.

“NewEstimate ← OldEstimate + StepSize [Target – OldEstimate]”

— the shape shared by almost every update rule that follows

Exploration strategies

ε-greedy

$$P(\text{random}) = \varepsilon$$

Take the greedy action almost always, a uniformly random one with probability ε. Simple, and it samples every action eventually.

Upper confidence bound

$$Q_t(a) + c\sqrt{\ln t / N_t(a)}$$

Add an uncertainty bonus that grows for actions tried rarely, so exploration targets what is least known rather than firing at random.

Optimistic initialization

$$Q_1(a) = +5$$

Start every estimate high, so untried actions look attractive and a purely greedy agent still explores early on.

State and action values

State value

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

Expected return from a state under π. Compact — one number per state — and the natural object for policy evaluation and iteration.

Action value

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

Expected return from taking a in s, then following π. More to store, but it ranks the actions in a state directly — which is what control needs.

Everything in sections 3 and 4 of the paper is a way of estimating one of these two functions, a policy directly, or both at once.

The same two functions, on a grid

s_1 0.59	s_2 0.66	s_3 0.73	s_4 0.81
s_5 0.66	s_6 0.73	wall	s_8 0.90
s_9 0.73	s_{10} 0.81	s_{11} 0.90	s_{12} 1.00
s_{13} 0.81	s_{14} 0.90	s_{15} 1.00	s_{16} goal

ACTION VALUES · $q(s_{13}, a)$

ACTION	WHERE IT LEADS	$q(s_{13}, a)$
up	→ s_9	0.66
down	blocked, stays	0.73
left	blocked, stays	0.73
right	→ s_{14}	0.81

THE RELATIONSHIP

$$v(s_{13}) = \max q(s_{13}, a) = 0.81$$

A state is worth exactly what its best action is worth.

But v alone never says which action that is — so control needs q.

The Bellman equations

EXPECTATION FORM

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

OPTIMALITY FORM

$$q_*(s, a) = \sum_{s',r} p(s', r \mid s, a)\big[\, r + \gamma\, \max_{a'} q_*(s', a') \,\big]$$

A value expressed through its successors' values — bootstrapping. Read as assignments these become the dynamic-programming updates; the max is what will reappear, sampled, in the Q-learning target.

02

Method families

Section 3: model-free against model-based, on-policy against off-policy — the two axes the paper classifies every algorithm along.

Model-free and model-based

Model-free

No transition or reward model — learn a value function or a policy straight from observed experience. Simpler to implement, and the family every algorithm in this deck belongs to. Splits into value-based, policy-based and hybrid.

Model-based

Learn or assume $p(s' \mid s, a)$, then plan against it — simulating action sequences without paying for them in the real environment. Faster to adapt and more sample-efficient, at the cost of model accuracy and compute.

The paper's split of model-free methods: value-based (Q-learning, SARSA), policy-based (REINFORCE, PPO), and hybrid actor–critic methods that carry both.

On-policy and off-policy

On-policy

Behavior policy and target policy are the same: the agent improves the policy it is acting with. Learning is stable and coherent, but data from an older policy is no longer usable and exploration is limited by the policy itself.

Off-policy

A behavior policy b explores while a separate target policy π is improved. Experience from any policy can be reused — sample-efficient and strongly exploratory, but unstable if b drifts far from π.

Q-learning and DQN are off-policy; REINFORCE, A2C and A3C are on-policy. Actor–critic methods split the roles between the actor that acts and the critic that evaluates.

The algorithm map

ALGORITHM	POLICY	WHAT IT LEARNS
Q-learning	Off-policy	q of the greedy policy
SARSA	On-policy	q of the acting policy
DQN	Off-policy	q, as a network
PPO	On-policy	π, with a clipped step

Four of Table 1's thirteen rows — and all thirteen are model-free: none of them learns $\mathscr{P}$ or $\mathscr{R}$.

03

Q-learning

Watkins, 1989 — the off-policy temporal-difference rule. One number per state–action pair, improved by a single line of arithmetic every step.

One grid, used throughout

STATES

16 cells, s_1 … s_{16}

ACTIONS

up, down, left, right · deterministic; walls block

REWARD

+1 on entering s_{16} · 0 on every other step

SETTINGS

$\gamma = 0.9$ · $\alpha = 0.5$ · episodes start at s_{13}

every Q(s, a) starts at 0, so nothing is learned until the goal is reached once

The update rule

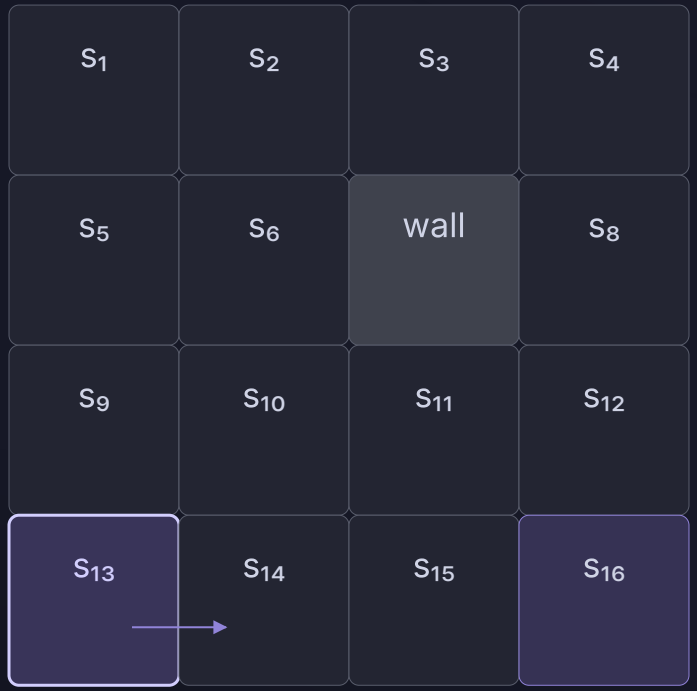
OFF-POLICY TD CONTROL · WATKINS, 1989

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

The bandit update with a Bellman target substituted in: one sampled reward plus the discounted best value the next state is currently believed to have. α sets how much of that disagreement is absorbed — at α = 1 the old estimate is thrown away.

- The max is over the *greedy* policy, not the one that acted — so an ε-greedy walk can teach the optimal policy.
- SARSA is one symbol apart — γ $Q(S_{t+1}, A_{t+1})$ — and learns the policy actually followed.
- Nothing here plans ahead: the only lookahead is one step, into estimates that are themselves guesses.

Step 1 · nothing to learn yet



Q TABLE · ALL ZEROS AT THE START

STATE	UP	DOWN	LEFT	RIGHT
▶ s_{13}	0	0	0	0
s_{14}	0	0	0	0
s_{15}	0	0	0	0

CURRENT STATE · s_{13}

STEP 1 · s_{13} → right → s_{14}

r = 0 · best next value max $Q(s_{14}, \cdot) = 0$

target $y = 0 + 0.9 \times 0 = 0$

$Q(s_{13}, \text{right}): 0 \rightarrow 0$

no disagreement, so nothing is written

Step 2 · still nothing

Q TABLE · STILL UNTOUCHED

STATE	UP	DOWN	LEFT	RIGHT
s_{13}	0	0	0	0
▶ s_{14}	0	0	0	0
s_{15}	0	0	0	0

CURRENT STATE · s_{14}

STEP 2 · s_{14} → right → s_{15}

r = 0 · best next value max $Q(s_{15}, \cdot) = 0$

target $y = 0 + 0.9 \times 0 = 0$

$Q(s_{14}, \text{right}): 0 \rightarrow 0$

still nothing — every estimate ahead is zero

Step 3 · the first number

Q TABLE · ONE ENTRY, AFTER THREE STEPS

STATE	UP	DOWN	LEFT	RIGHT
s_{13}	0	0	0	0
s_{14}	0	0	0	0
▶ s_{15}	0	0	0	0.50

CURRENT STATE · s_{15}

STEP 3 · s_{15} → right → s_{16} · TERMINAL

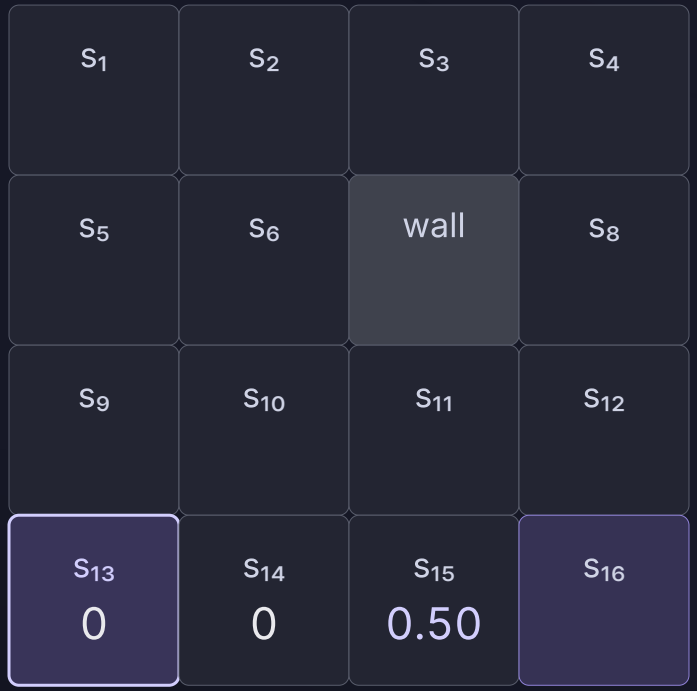
r = +1 · terminal, so there is no bootstrap term

target y = 1.00

$Q(s_{15}, \text{right})$: $0 + 0.5\ (1.00 - 0) = 0.50$

the first number the table has ever held

After episode 1 · one cell



Q TABLE AFTER EPISODE 1

STATE	UP	DOWN	LEFT	RIGHT
▶ s_{13}	0	0	0	0
s_{14}	0	0	0	0
s_{15}	0	0	0	0.50

CURRENT STATE · s_{13} · reset for episode 2

AFTER EPISODE 1

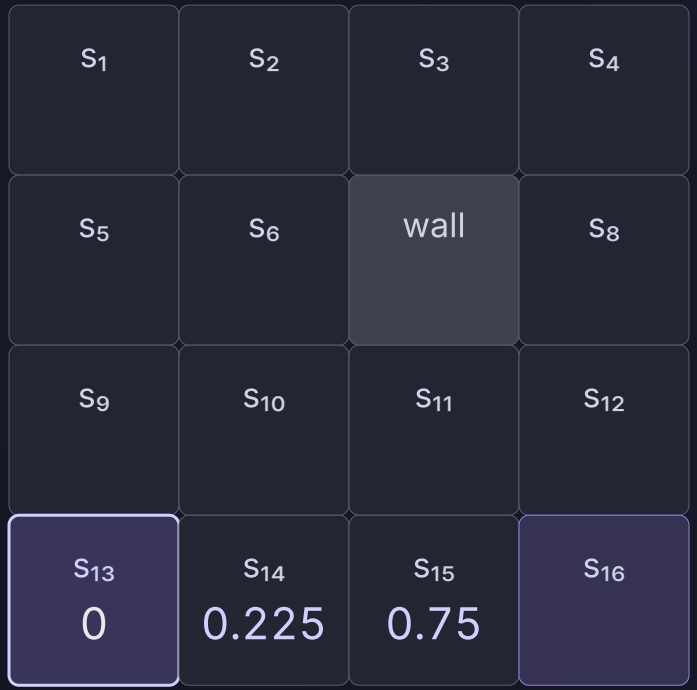
the walk $s_{13} \rightarrow s_{14} \rightarrow s_{15} \rightarrow s_{16}$, once

one cell holds a value

$$Q(s_{15}, \text{right}) = 0 + 0.5\ (1.00 - 0) = 0.50$$

s_{13} and s_{14} still see only zeros ahead

After episode 2 · two cells



Q TABLE AFTER EPISODE 2

STATE	UP	DOWN	LEFT	RIGHT
▶ s_{13}	0	0	0	0
s_{14}	0	0	0	0.225
s_{15}	0	0	0	0.75

CURRENT STATE · s_{13} · reset for episode 3

AFTER EPISODE 2

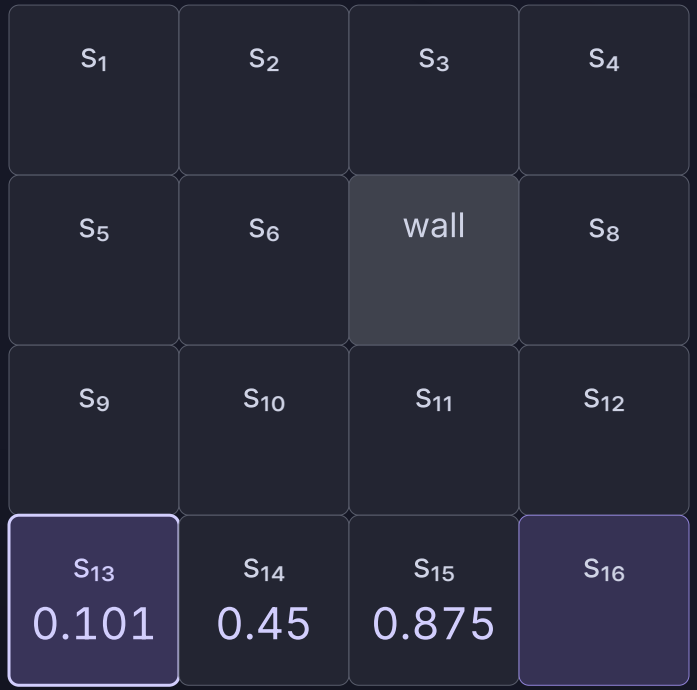
s_{14} now has something worth bootstrapping from

two cells hold a value

$$Q(s_{14}, \text{right}) = 0 + 0.5\ (0.9 \times 0.50) = 0.225$$

and s_{15} climbs to 0.75, halfway again

After episode 3 · the whole row



Q TABLE AFTER EPISODE 3

STATE	UP	DOWN	LEFT	RIGHT
▶ s_{13}	0	0	0	0.101
s_{14}	0	0	0	0.45
s_{15}	0	0	0	0.875

CURRENT STATE · s_{13} · reset for episode 4

AFTER EPISODE 3

the learned region has reached the start state

the whole row points at the goal

$$Q(s_{13}, \text{right}) = 0 + 0.5\ (0.9 \times 0.225) = 0.101$$

one cell per episode — none learns before its successor

Where the table runs out

One entry per pair

$16 \times 4 = 64$

Fine here. A single Atari frame is 210 × 160 pixels, and the same table would need more rows than could ever be visited, let alone stored.

The max must be tractable

$\max_{a \in \mathcal{A}} Q(s, a)$

Every update and every action taken needs that maximum. Over four grid moves it is trivial; over a continuous torque there is no maximum to take at all.

Fixes that stay tabular

$\text{SARSA} \cdot \text{double } Q$

Swapping the max for the acted action, or splitting it across two tables, cures other problems. Both keep the table, and both keep its ceiling.

Two doors out. Keep the rule and replace the table with a network — that is DQN. Or stop learning values and optimize the policy itself — that is PPO.

04

Proximal policy optimization

Schulman et al., 2017 — optimize the policy directly, but refuse to move it far from the one that collected the data.

Learn the policy, not the values

- Q-learning has no policy of its own — it stores values and reads one off with an argmax. PPO carries $\pi_\theta(a \mid s)$ as the object being learned.
- On our grid that changes what is stored: not 64 action values, but four probabilities per state, produced by θ.
- No max to take, so the action space may be continuous; and the output is already a distribution, so exploration needs no ε bolted on.
- The catch: policy gradients are on-policy and high-variance, and one step slightly too large collapses a policy that was working.

That last line is the whole reason PPO exists — everything on the next two slides is a guard rail on step size.

From return to ratio

REINFORCE · WILLIAMS, 1992

$$\theta \leftarrow \theta + \alpha \, \nabla_\theta \log \pi_\theta(a \mid s) \, G_t$$

Push up the log-probability of each action in proportion to the return that followed it. Unbiased, very noisy, and good for exactly one update per episode.

THE SURROGATE, WRITTEN AS A RATIO

$$L(\theta) = \mathbb{E}[\, r_t(\theta) \, \hat{A}_t \,], \qquad r_t(\theta) = \pi_\theta(a_t \mid s_t) \,/\, \pi_{\text{old}}(a_t \mid s_t)$$

The same gradient, expressed relative to the policy that collected the batch, and weighted by the advantage Â rather than the raw return. Because the ratio is measurable, the step can be policed — which is the next slide.

The loop, in seven steps

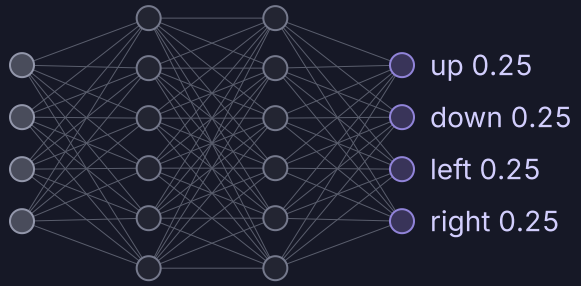
01	Initialize the *actor* network π_θ	once · state in, probabilities out
02	Initialize the *critic* network V_φ	once · state in, one number out
03	Collect a batch of trajectories	act with the current actor
04	Compute the future reward from each state	backwards, one pass
05	Subtract the baseline to get advantages	the critic supplies it
06	Optimize the critic	plain regression, unbounded
07	Optimize the actor	the clipped step, several epochs

Steps 3 to 7 are one iteration. The batch is then thrown away and the loop starts again with the policy that step 7 produced — which is why the ratio in step 7 always compares against the actor as it was at step 3.

Steps 1 & 2 · initialize two networks

1 · Actor network · π_θ

the thing that acts

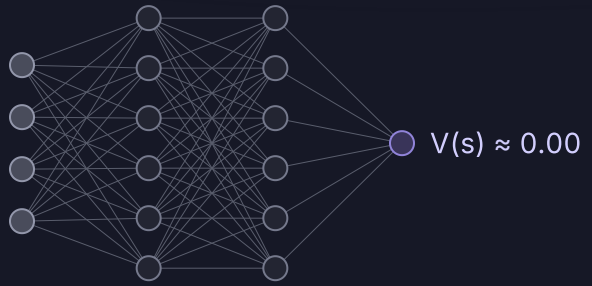


state in

random weights → near-uniform

2 · Critic network · V_φ

never acts — it only judges



the same state in

one number, currently meaningless

Same input, different jobs and different losses. The actor is nudged by the clipped objective in step 7; the critic is fitted by regression in step 6. Neither is useful yet — at initialization the policy is a coin flip and the value estimate is noise.

Step 3 · collect a batch of trajectories

EPISODE	T	STATE	ACTION TAKEN	REWARD	$\Pi_{OLD}(A \mid S)$ · STORED
ep 2	4	s_{13}	right	0	0.30
ep 5	11	s_{14}	right	0	0.40
ep 9	17	s_{15}	up	0	0.50

A real batch is a few thousand timesteps spanning many episodes; three are shown, deliberately from three different ones — t indexes the batch, not a walk. Reward is 0 everywhere but the goal, which pays +1 and ends the episode. $\gamma = 0.9$. The actor is now frozen as π_{old} for the rest of the iteration.

Step 4 · future reward from each state

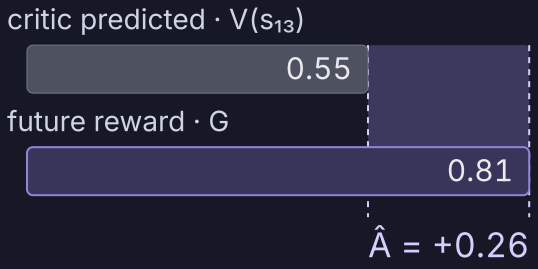
EPISODE	T	STATE	REWARD R	G = R + Γ · G(NEXT)
ep 2	6	s_{15}	+1 · terminal	1.00
ep 2	5	s_{14}	0	0 + 0.9 × 1.00 = 0.90
ep 2	4	s_{13}	0	0 + 0.9 × 0.90 = 0.81

Computed backwards, from the end of the episode to its start — one pass, no network involved. Episode 2 is shown; the same pass over episodes 5 and 9 gives G = 0.90 for s_{14} and G = 0.59 for s_{15}, the latter low because going up put the goal six steps away instead of one.

Step 5 · future reward minus the baseline

Better than expected

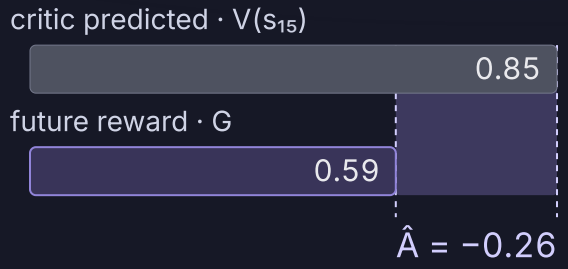
ep 2, t = 4 · from s_{13} the agent went right



beat the state's reputation → make this action more likely

Worse than expected

ep 9, t = 17 · from s_{15} the agent went up



fell short of it → make this action less likely

Advantage is $\hat{A} = G - V_{\varphi}(s)$. The third sample, s_{14}, works out at +0.20 the same way. Near the goal every action returns a high number, so G alone would reward a bad action taken in a good place — subtracting the baseline leaves only the credit that belongs to the action.

Step 6 · optimize the critic

$$\mathcal{L}(\phi) = \frac{1}{N} \sum \left(V_\phi(s) - G\right)^2$$

STATE	$V_\Phi(S)$ BEFORE	TARGET G · STEP 4	ERROR	$V_\Phi(S)$ AFTER
s_{13}	0.55	0.81	+0.26	0.60
s_{14}	0.70	0.90	+0.20	0.74
s_{15}	0.85	0.59	−0.26	0.80

Plain gradient descent on squared error — no ratio, no clip, no trust region. The critic may move as far as the data justifies, because a wrong value estimate costs accuracy, not a collapsed policy.

The clipped objective

CLIPPED SURROGATE · E ≈ 0.2

$$L^{\mathrm{CLIP}}(\theta) = \mathbb{E}\big[\, \min\big(r_t(\theta)\,\hat{A}_t,\ \mathrm{clip}\big(r_t(\theta),\,1-\epsilon,\,1+\epsilon\big)\,\hat{A}_t\big)\,\big]$$

Two copies of the same quantity, one with the ratio flattened outside [1−ε, 1+ε]. The min takes whichever is worse, so once the ratio leaves the range the objective stops rewarding further movement.

THE ADVANTAGE IT IS WEIGHTED BY

$$\hat{A}_t = \delta_t + (\gamma\lambda)\,\delta_{t+1} + \cdots, \qquad \delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$$

Generalized advantage estimation: a learned value head V_φ supplies the baseline, and λ trades its bias against the variance of the raw return. So PPO does learn a value function — just not to act with.

Step 7 · optimize the actor

SAMPLE	$\hat{A}$ · STEP 5	$r_T(\Theta) = \Pi_\Theta / \Pi_{OLD}$	$r \cdot \hat{A}$	CLIP(r, 0.8, 1.2) · $\hat{A}$	MIN → USED
ep 2 · s_{13} right	+0.26	0.45 / 0.30 = 1.50	0.390	1.20 × 0.26 = 0.312	0.312 · clipped
ep 5 · s_{14} right	+0.20	0.44 / 0.40 = 1.10	0.220	1.10 × 0.20 = 0.220	0.220 · gradient live
ep 9 · s_{15} up	−0.26	0.35 / 0.50 = 0.70	−0.182	0.80 × −0.26 = −0.208	−0.208 · clipped

Averaging the last column gives the loss this minibatch reports, and its gradient is the step taken. Two of the three rows are pinned at their clipped value, so further movement earns nothing — which is what lets the same batch be reused for several epochs before the loop returns to step 3.

2

places to keep what you have learned. In the values, and read the policy off them with an argmax — or in the policy itself, and bound how far one batch may move it. Every row of the paper's Table 1 sits on one side, the other, or astride both.

Where the paper points next

01	Reinforcement Learning: An Introduction	Sutton & Barto — the reference text
02	Algorithms for Reinforcement Learning	Szepesvári — short and formal
03	Deep Reinforcement Learning Hands-On	Lapan — implementation first
04	RL Specialization, University of Alberta	Coursera — course form
05	David Silver's RL course	DeepMind × UCL lecture series
06	OpenAI Spinning Up in Deep RL	tutorials with working code

Walkthroughs and a playground

01	Q-learning, explained visually youtube.com/watch?v=TiAXhVAZQI8	the tabular update, step by step
02	PPO, explained visually youtube.com/watch?v=5VHLd9eCZ-w	the source for this deck's seven steps
03	Web RL playground awjuliani.github.io/web-rl-playground	train an agent in the browser, no install

The two videos take the same route through Q-learning and PPO that sections 03 and 04 do. The playground is the fastest way to watch value propagate backwards on a grid for yourself.

Two places to keep it

Q-learning writes the Bellman target into a cell and reads its policy off the table; PPO carries the policy itself and bounds how far one batch may move it.

arXiv:2408.07712 · Ghasemi & Ebrahimi, 2024