Reinforcement Learning Fundamentals

2026-05-06 · cheerful mango Haubentaucher

Playing Atari.

The Why

In the previous chapter supervision still arrived as labels, but a label on a single input cannot transmit a sequence of decisions or the consequences of one action on the next. Reinforcement learning answers a complementary question: What if the supervision is a scalar reward and depends on the sequence of actions a model takes?

Reward. A scalar number returned by the environment after an action, carrying no information about which action would have been better, no gradient, no class identity, only a magnitude and a sign. Let's extract signal yet again.

Markov decision processes. The setting is formalised [1] as a tuple of a state s, an action a, a transition probability $P(s' \mid s, a)$, a reward $r(s, a)$, and a discount factor $\gamma \in [0, 1)$. The Markov property requires that the next state depends only on the current state and action, which lets the agent reason forward without dragging the past along, while γ downweights rewards t steps in the future by γ^t, keeping the sum of returns finite over infinite horizons.

Simulation as substrate. Most reinforcement learning happens inside a simulator, since collecting millions of transitions on real hardware is prohibitive. The simulator's reward and transition function are perfect inside the sim and biased outside it, and the resulting sim-to-real gap is the central practical concern.

[1]

Bellman equations. The value of a state under policy π is the expected discounted return when starting in s and following π from then on. Bellman's insight is that this value satisfies a local recursion: The value of a state equals the expected immediate reward plus the discounted expected value of the next state. This turns long-horizon optimisation into a fixed-point problem which suits Markovian dynamics.

Pressure One: Credit assignment. The reward arrives long after the action that helped earned it. Bellman decomposition turns the long-horizon problem into a local fixed-point one.

Value-based methods learn what each action is worth. A Q-learning agent maintains an action-value table and bootstraps each estimate against the next, converging to the optimum even when behaviour is off-policy. The Deep Q-Network, DQN, scales the table to a neural network for high-dimensional inputs, which means a

Model and policy. A model θ is the bag of parameters that gradient descent updates. A policy π is the behaviour rule the agent acts under, mapping states to actions. The two coincide in policy-gradient methods, where θ directly parameterises π_θ, but separate in value-based methods, where θ parameterises a value function Q_θ and the policy is derived from it as greedy or ε-greedy.

single set of weights now produces the value for every state-action pair, and updating the weights for one state changes the values for all of them. Two instabilities follow.

Replay Buffer First, the agent's experience is a stream of correlated transitions: Each new state is the consequence of the previous one, and the agent often spends long stretches in similar regions of the environment. A network trained directly on this stream overfits to whatever region the agent is currently in and forgets what it learned about earlier ones. The experience replay buffer stores past transitions in a large pool and samples minibatches uniformly from it, so each gradient step sees a mix of transitions from many different points in time. Notice that this quickly becomes a compute bottleneck and does not scale.

Target Network Second, the bootstrapped target uses the same network whose weights we are updating, so each update changes both the predictions and the targets they are regressing against, and convergence is unstable because the network is chasing a target that moves with it. A target network is a slow-updating copy used only to compute that target, kept frozen for thousands of steps so the online network has a stable regression problem to converge to before the target shifts.

Pressure Two: Stability. Bootstrapped value estimates and policy gradients can amplify their own noise. Replay buffers, target networks, and trust regions stop the loop from running away.

Policy-based methods learn the action rule directly. A value-based method learns what each action is worth and then acts greedily, deriving the policy from the value estimates. A policy-based method skips the value function and parameterises a stochastic policy $\pi_\theta(a \mid s)$ that outputs the probability of each action directly. This also allows for continuous action spaces, where a value-based method regresses. The policy gradient theorem turns the long-run expected return into a quantity we can backpropagate against, and applies it by rolling out a trajectory, observing the return, and nudging π_θ to make the actions that led to high return more likely and those that led to low return less likely.

Trust regions and clipping. Policy-gradient methods are on-policy: The gradient is computed under the old policy, so if a single update moves the new policy too far the data no longer applies and performance collapses. Trust-region methods address this by explicitly constraining the size of the policy update, which requires a second-order optimisation step that is expensive to compute. Proximal Policy Optimisation, PPO [2], replaces the explicit constraint with a much simpler clipping trick. Each action in a trajectory has a prob-

Off-policy vs on-policy. Q-learning and DQN learn from data collected under any behaviour, including a replay buffer of past transitions. Policy-gradient methods need data collected under the current policy, which is why trust regions matter: They control how far the new policy can drift before the old data stops applying.

[2]

ability under the old policy and a probability under the new policy, and their ratio measures how much the policy has shifted on that action since the last update. PPO clips this ratio to a narrow band around one, typically $[0.8, 1.2]$, before plugging it into the policy-gradient update. The width of the band opts for stability over sample efficiency.

TRADE-OFF EXPLORATION AND EXPLOITATION. Every method above needs a behaviour rule for collecting data, and that rule has to balance two pressures. Exploit what is currently known to be good so reward is collected, explore actions that are uncertain so the value estimates can improve. ε-greedy, and entropy bonuses on the policyare the instruments that show up everywhere in practice, and the design of better exploration is an open subfield in its own right.

PRESSURE THREE: EXPLORATION. Without enough variety in the data, the agent never sees the actions whose value it has yet to estimate. Exploration is the open subfield that supplies that variety.

IN ORDER TO MOVE TO LEARNING FROM A SCALAR, MODEL DEPENDANT REWARD we have good reason to find ways to,

- formalise sequential decision problems as Markov Decision Processes and decompose long-horizon value into a local Bellman recursion.
- learn action values by bootstrapping each estimate against the next, and stabilise the neural-network case with a replay buffer for decorrelated samples and a target network for a fixed regression target.
- parameterise a stochastic policy directly and update it by rolling out trajectories and reweighting actions according to the returns they led to.
- constrain how far each policy update is allowed to move from the previous one, so that the data collected under the old policy still informs the new one.
- collect informative experience under uncertainty, balancing exploitation of known reward against exploration of unfamiliar actions.

ATARI AND GO. A DQN agent learned to play forty-nine Atari games from raw pixels and a single score signal . The score arrives at the end of a life or an episode, not after each frame, and the agent has to figure out by itself which of the thousands of intermediate joystick presses earned it.

THE FUNDAMENTAL QUESTION this chapter answers is whether a single scalar reward, delayed and sparse, carries enough signal to teach an agent how to act, without any teacher ever specifying what the right action would have been.

Self-Reflection and Recap

SELF-REFLECTION questions to guide your thinking:

- Why does the Markov property let the agent reason about the future without dragging the past along, and what would change if the next state depended on the full history?
- Why does the discount factor γ exist at all, and how does $\gamma < 1$ keep the sum of returns finite over infinite horizons?
- How does the Bellman recursion turn a long-horizon optimisation into a local fixed-point problem?
- What does it mean for Q-learning to be off-policy, and why does that property let DQN learn from a replay buffer of past transitions?
- Why does scaling the Q-table to a neural network introduce instabilities that the tabular case did not have, given that one set of weights now produces values for every state-action pair?
- What problem does the experience replay buffer solve, how does uniformly sampling past transitions from a large pool address it, and where does the approach run into a compute bottleneck?
- Why is a target network needed on top of the replay buffer, and what goes wrong when the target moves with the network being trained?
- How is a policy-based method different from a value-based method, why does parameterising $\pi_\theta(a \mid s)$ directly let you skip learning the value function, and how does this open the door to continuous action spaces?
- Why is the policy gradient theorem the key technical move that lets us treat the long-run expected return as a quantity we can backpropagate against?
- Why are trust regions and PPO clipping a policy-gradient concern only, and not relevant for Q-learning?
- Why does PPO clip the ratio between new and old policy probabilities to a narrow band such as $[0.8, 1.2]$, and how does the width of that band trade stability against sample efficiency?
- Why does every method in this chapter need a behaviour rule for exploration, and what role do ε-greedy, entropy bonuses on the policy, and stochastic policies play as the three blunt instruments?

Recap of key concepts:

- Markov decision processes formalise sequential decision making as the tuple (s, a, P, r, γ) with the Markov property: the next state depends only on the current state and action, and γ downweights rewards t steps in the future by γ^t to keep returns finite over infinite horizons.
- The Bellman equations decompose the value of a state into immediate reward plus discounted value of the next state, and almost every method in this chapter is a way to solve that local fixed-point problem under different practical constraints.
- Value-based methods learn what each action is worth and act greedily; Q-learning is off-policy and converges in the tabular case, and DQN scales it to neural networks with an experience replay buffer that mixes correlated transitions and a target network that holds the regression target still, though replay sampling becomes a compute bottleneck at scale.
- Policy-based methods skip the value function and parameterise a stochastic policy $\pi_\theta(a \mid s)$ directly, which also unlocks continuous action spaces where value-based regression struggles; the policy gradient theorem provides a backpropagatable surrogate, and PPO bounds the on-policy update by clipping the probability ratio between new and old policy to a narrow band around one.
- Exploration supplies the data variety the methods above consume; ε-greedy, entropy bonuses on the policy, and stochastic policies are the three blunt instruments, and better exploration is an open subfield in its own right.

Teaser. Bellman decomposition makes long-horizon credit assignment tractable in principle. In practice, sparse rewards mean infeasible many steps before any informative signal arrives. What signal would shortcut all that?

Reinforcement learning's hardest problem is long-horizon credit assignment. Bellman decomposition reduces it to local updates, but across sparse, long horizons it arrives too rarely to learn from on any practical budget.

The next chapter turns to interactive and imitation learning, where we compress the long-horizon credit-assignment problem into a single demonstration, correction, or preference. Reinforcement learning derived its supervision from environmental reward; interactive learning derives it from few-shot demonstrations, corrections, or preferences from an annotator.