Learning to Act: Deep Reinforcement Learning in Finance

Agents, Rewards, Q-Learning, Policy Gradients, and Their Limits

Day 7 Mini-Lecture, 60 Minutes

Discovery: Teaching a Trader With No Answer Key

You hire a junior trader but refuse to tell them the "right" trade.

- All you give them each evening is the day's profit or loss.
- No textbook, no labelled examples of the correct action for each market.

Discuss with your neighbour (1 minute)

How could anyone get better at a job when the only feedback is a number at the end of the day?

What Is Reinforcement Learning?

Supervised learning needs labelled answers. Reinforcement learning has no answer key.

- An agent learns by trying actions and seeing how much reward follows.
- Five pieces: agent, environment, state, action, reward.
- Finance picture: the agent is a trading policy; the environment is the market.
- The goal is to maximise total reward over time, not just the next step.

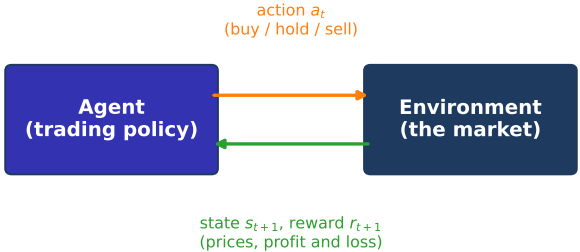
Key idea

RL learns behaviour from consequences, not from labelled examples.

RL learns by trial, reward, and repetition

The Reinforcement Learning Loop

Reinforcement learning is a loop: act, observe, get reward, repeat



At each step the agent acts; the market responds with a new state and a reward

Discovery: What Should the Trader Even See?

Before our agent can act, we must decide what it gets to look at.

- Option A: feed it only today's closing price.
- Option B: feed it recent prices, volatility, its cash, and its current position.
- Which one gives it a fair chance to make a sensible decision?

Pick one, then the next slide names the moving parts of the setup

A Trading Agent in the Market

- **State** s_t: what the agent observes (recent prices, volatility, position, cash).
- **Action** a_t: what it can do (buy, hold, sell, or set portfolio weights).
- **Reward** r_t: the immediate payoff (change in profit and loss, minus costs).
- **Policy**: the rule mapping states to actions. Learning means improving this rule.

What we are really training

The policy is the deliverable. State, action, and reward all exist to shape it.

State in, action out; reward is the only teacher

Discovery: Who Earned Today's Profit?

Your agent made twelve trades this morning and finished the day up $400.

- Some of those trades helped; some quietly lost money.
- You only see the single end-of-day total, not a score per trade.

Hint

Think about how a coach with only the final score figures out which plays actually worked.

Reward, Not Labels

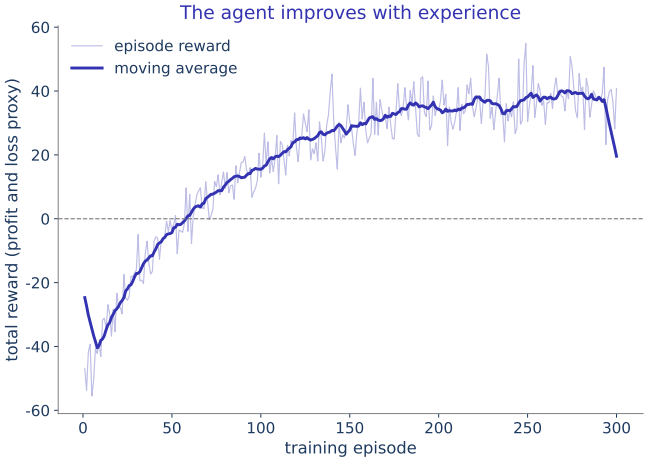
- A supervised model is told the right answer for every input.
- An RL agent is told only how good its actions were, and often much later.
- **Credit assignment**: which earlier action caused today's gain? This is the hard part.
- Rewards can be sparse (only at the end) and noisy (luck versus skill).

Why RL is hard

Delayed, noisy reward is what makes RL both powerful and difficult.

The agent must learn which past actions earned today's reward

The Agent Improves With Experience



An episode is one full run (say, one trading day); average reward per episode rises as the policy learns

Discovery: Is a Dollar Tomorrow Worth a Dollar Today?

Commit to a number before we define anything.

- A guaranteed $100 lands in your account one year from now.
- Write down what you would pay today to own that future $100.
- Now write a single multiplier between 0 and 1 that turns $100 into your answer.

Keep that multiplier; the next slide gives it a name

Return and Discounting

The agent maximises return, the sum of future rewards, not just the next one.

- A discount factor γ between 0 and 1 shrinks rewards that are further away.
- γ near 1: patient, plans far ahead. γ near 0: myopic, grabs immediate gains.
- In finance, discounting also reflects that distant payoffs are less certain.

Return

$G_t = r_t + \gamma\, r_{t+1} + \gamma^2 r_{t+2} + \dots$ Here r_t is the reward now, r_{t+1} the next step's, and so on. The agent optimises this whole sum.

Discounting sets how far into the future the agent plans

Worked Example: Discounted Return

Three days of reward: $+10$, then $+8$, then $+5$. Discount factor $\gamma = 0.9$.

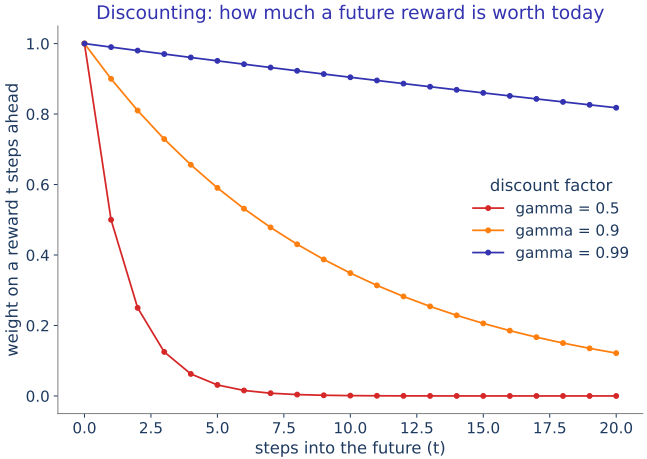
- Today's reward counts fully: 10.
- Tomorrow's is worth $0.9 \times 8 = 7.2$.
- The day after: $0.9^2 \times 5 = 0.81 \times 5 = 4.05$.

Return

$G = 10 + 7.2 + 4.05 = 21.25$. A more patient agent ($\gamma$ closer to 1) would value the later rewards more heavily.

Return is the discounted sum of future rewards, here 21.25

How Much Does the Future Count?



With $\gamma = 0.99$ rewards 20 steps out still matter; with $\gamma = 0.5$ they vanish

Discovery: How Much History Must You Carry?

To decide your next trade, how far back do you really need to look?

- One trader insists on rereading every tick since the company's IPO.
- Another keeps only a short summary: trend, volatility, and current position.

Discuss with your neighbour (1 minute)

Can a compact snapshot of "right now" be enough to act well, or must you remember everything?

Markets as a Markov Decision Process

An MDP is the formal RL world: states, actions, transition probabilities, rewards.

- **Markov property**: the next state depends only on the current state and action, not the full history.
- This is an approximation for markets, but a useful one: summarise the past into the state.
- Good state design is most of the battle in applied RL.

The point of a state

If the state captures what matters, the agent can act well without remembering everything.

An MDP is states, actions, transitions, and rewards; the state carries the memory

Discovery: Which Market Would You Rather Wake Up In?

Two situations greet you at the open.

- Situation A: a calm, steadily rising market with your usual strategy running.
- Situation B: a violent crash, but you are already positioned to short it.
- Which "state" is more valuable, and does your answer depend on your strategy?

Hold your pick; the next slide puts a number on how good a state is

How Good Is a State? Value Functions

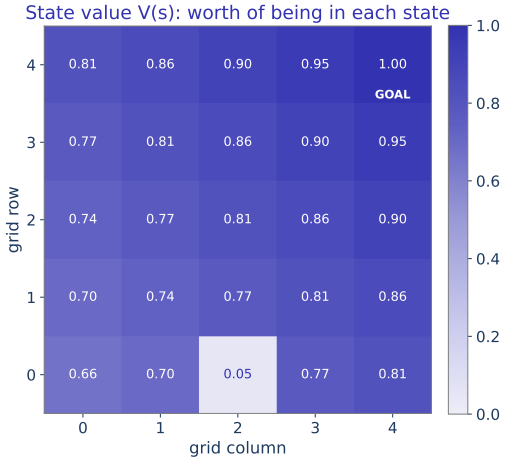
- $V(s)$: the expected return if we start in state s and follow the policy.
- $Q(s, a)$: the expected return if we take action a in state s, then follow the policy.
- Knowing Q tells the agent what to do: pick the action with the highest Q.
- Finance: value depends on the policy; a good policy can make even a crash state valuable (you can profit from shorting).

Why values matter

Value functions turn "what should I do?" into "which action has the highest Q?".

V scores states; Q scores actions in states

State Values on a Grid



A gridworld is a toy maze toward a goal; value is high near the goal and decays with distance

Discovery: Score Each Move in a Bull Market

The market is calmly rising. Rate three actions on a scale of 0 to 1.

- How good is it to BUY right now? Write a number.
- How good is it to HOLD? To SELL? Write those too.
- Which of your three numbers is largest, and what does that tell the agent to do?

Keep your three scores; the next slide stores exactly these in a grid

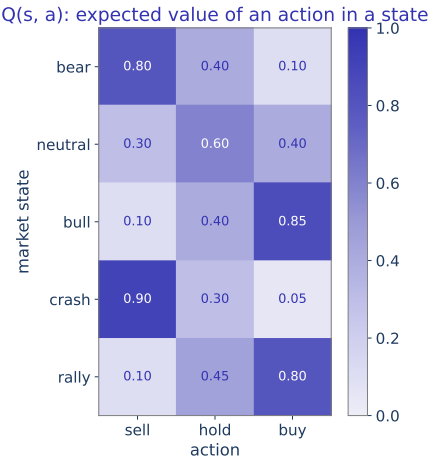
The Q-Table

- For small problems, store $Q(s, a)$ in a table: one row per state, one column per action.
- The best action in each state is the column with the largest value.
- Finance toy: states are market regimes; actions are sell, hold, buy.
- **Example:** in "bull", $Q(\text{buy}) = 0.85$ beats $Q(\text{hold}) = 0.40$ and $Q(\text{sell}) = 0.10$, so the agent buys.

Checkpoint: before the next slide, predict the best action in a "crash" state.

Read a Q-table by row: the largest entry is the action to take

Reading a Q-Table



In each market state the brightest cell is the action the agent prefers

Discovery: Your Usual Lunch or the New Place?

A tiny daily choice that is secretly the whole problem.

- Option A: order from the cafe you know is reliably good.
- Option B: try the new place that might be better, or might be a letdown.
- When should a trading agent stick with its best-known move versus test a new one?

Decide when each makes sense; the next slide gives this tension a name

Exploration versus Exploitation

- **Exploit**: take the action that looks best so far.
- **Explore**: try something else to find out if it is actually better.
- Too much exploit: the agent locks onto a mediocre habit. Too much explore: it never cashes in.
- **Epsilon-greedy**: act randomly with probability ϵ, else greedily, and shrink ϵ over time.

Common mistake: setting $\epsilon = 0$ means the first decent strategy found is the one you are stuck with.

Explore enough to learn, exploit enough to earn

Worked Example: Epsilon-Greedy in Action

The agent is in a "bull" state where buying looks best. Exploration rate $\epsilon = 0.1$.

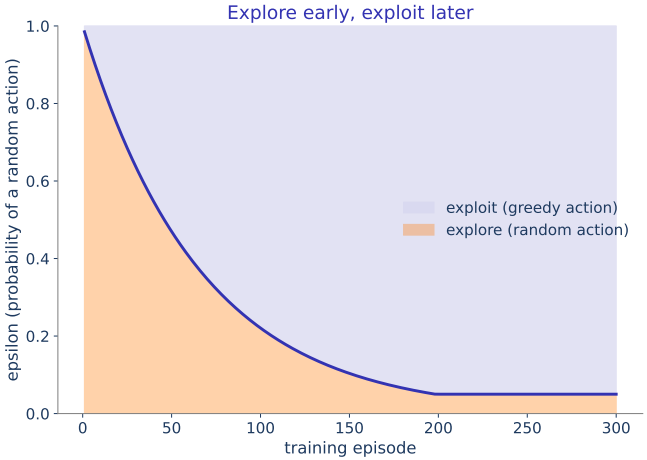
- With probability 0.9: take the greedy action and buy.
- With probability 0.1: pick a random action, with buy, hold, and sell equally likely.
- Over 100 visits: about 90 deliberate buys, plus a few more buys chosen at random.

Why spend the 10 percent

Those random tries are how the agent discovers whether "hold" is secretly better. With $\epsilon = 0$ it would never find out.

Epsilon is the slice of decisions the agent spends exploring instead of exploiting

From Exploring to Exploiting



Start mostly random, then rely more on the learned policy as ϵ decays

Discovery: Pricing a Move You Have Not Finished

You buy now and earn $2 immediately, but the position is still open.

- The trade is not done; tomorrow you will act again from wherever you land.
- How should you value today's move when most of the payoff is still ahead?

Hint

You already estimated how good the next state is. Reuse it instead of starting from scratch.

The Bellman Update

The value of a state equals the immediate reward plus the discounted value of the next state.

- $Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$. Here $\leftarrow$ means "update to", s' is the next state, and $\max_{a'}$ takes the best action there.
- Apply this repeatedly and the estimates settle; with noisy rewards a learning rate blends the old and new estimate.
- This single recursive rule is the engine under almost every RL method.

In words

A state is worth its reward now plus the discounted worth of where it takes you.

Bellman: value now = reward + discounted value of the next state

Worked Example: One Bellman Update

Update rule: $Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$**.**

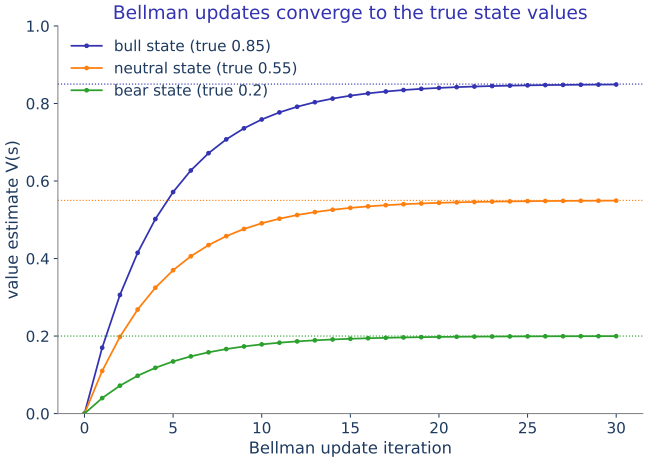
- The agent buys and receives an immediate reward $r = 2$.
- In the next state, the best action is currently valued at $\max_{a'} Q(s',a') = 10$.
- Discount factor $\gamma = 0.9$.

New estimate

$Q(s, \text{buy}) \leftarrow 2 + 0.9 \times 10 = 2 + 9 = 11$. Apply this over many transitions and the estimates settle.

One Bellman update: immediate reward plus the discounted value of where you land

Bellman Updates Converge



Repeated updates drive each value estimate to its true level

Discovery: How Big Is the Table?

Your state has 10 features, and you bucket each into 10 levels.

- One table row is needed for every possible combination of those buckets.
- Guess the number of rows: hundreds, millions, or something far larger?
- Write your estimate as a power of ten before the next slide.

Hold your guess; the next slide checks whether any table can hold it

Why Q-Tables Break Down

- A table needs one entry per state. Real states are rich: prices, volumes, indicators, positions.
- Ten features at ten levels each is ten billion states. No table can hold or fill that.
- Most states are never visited, so the table cannot generalise to new situations.

Tables versus functions

Tables memorise; they cannot generalise. Real problems need a function that does.

The curse of dimensionality kills the lookup table

Worked Example: Why a Table Is Hopeless

Suppose the state has 10 features, each bucketed into 10 levels.

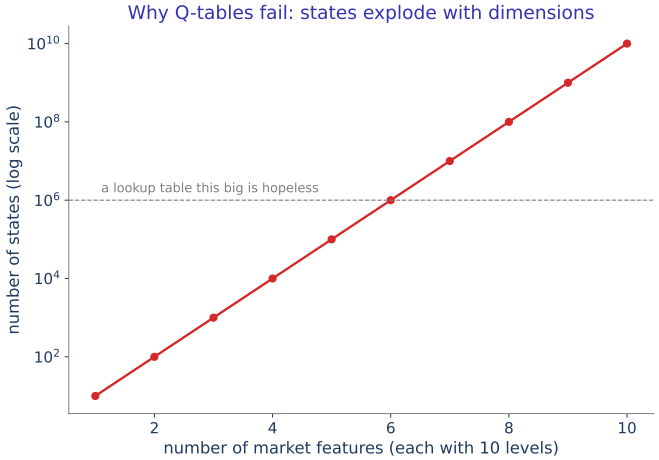
- Number of states: 10^{10}, that is ten billion rows.
- At one byte per entry that is about 10 gigabytes, just for a single action column.
- Worse, the agent will ever visit only a tiny fraction, so almost every row stays empty.

The lesson

A table can neither store nor fill this. A network that generalises across similar states can.

Ten features at ten levels is ten billion states; tables do not scale

States Explode With Dimensions



Each added feature multiplies the state count; a table becomes impossible fast

Discovery: A Market You Have Never Seen Before

Today's exact mix of price, volatility, and volume has literally never occurred.

- A lookup table has no row for it, so it offers no guidance at all.
- Yet the situation clearly resembles others you have traded through.

Hint

What tool from earlier in the course takes inputs and predicts an output for cases it never saw in training?

Deep Q-Networks

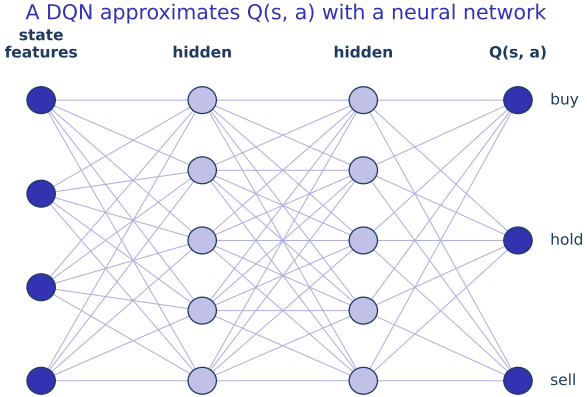
- Replace the table with a neural network that takes a state and outputs a Q value for each action.
- The network generalises: similar states give similar Q values, even unseen ones.
- Train it so its output matches the Bellman target (reward plus discounted next value).
- This is the DQN that first learned to play Atari games from raw pixels.

What a DQN is

A Q-table replaced by a function approximator that generalises across states.

One network maps any state to a Q value per action

A Deep Q-Network



State features flow through hidden layers to one Q value per action

Discovery: Train Only on the Latest Minute

A reasonable-sounding plan: always learn from the trade you just made, then discard it.

- Consecutive market minutes look almost identical to each other.
- So the agent sees a long streak of near-copies, then a different streak later.
- What goes wrong when each lesson looks just like the one before it?

Spot the weakness before the next slide fixes it

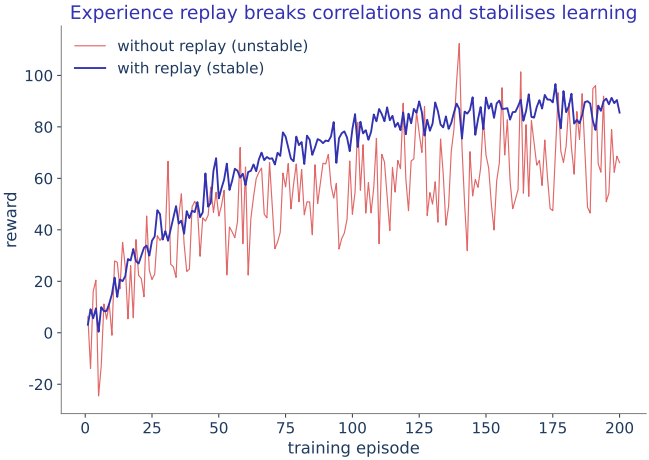
Experience Replay

- Consecutive market steps are highly correlated; training on them in order is unstable.
- Store past transitions (state, action, reward, next state) in a buffer.
- Train on random minibatches from the buffer, which breaks the correlation.
- Bonus: each experience is reused many times, so learning is more data-efficient.

Common mistake: training only on the latest step lets recent noise jerk the network around.

Replay shuffles experience so the network learns from variety, not streaks

Replay Stabilises Training



With replay the reward climbs smoothly; without it, training thrashes

Discovery: Aiming at a Target You Keep Moving

The agent adjusts its estimate to match a goal it computes from that same estimate.

- Every time the estimate shifts, the goal it is chasing shifts with it.
- Picture aiming at a dartboard that slides sideways the instant you throw.

What goes wrong?

Why might learning never settle when the thing you aim at moves every time you do?

Target Networks and Stability

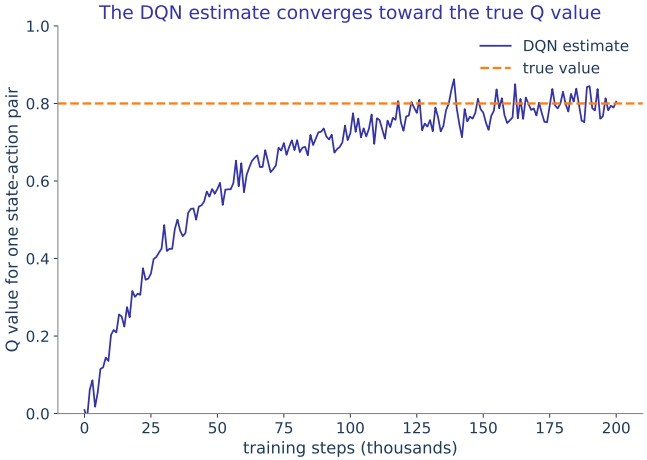
- The Bellman target uses the very network we are training, so it keeps moving (a moving goalpost).
- Fix: hold a frozen copy, the target network, and update it only occasionally.
- This steadies the target so the main network can converge.

Why it helps

A slowly-updated target turns a moving goalpost into a stable one to aim at.

Freeze the target so the estimate has something steady to aim at

The Estimate Finds the True Value



With these stabilisers the network's Q estimate settles near the true value

Discovery: Rate the Moves, or Just Decide?

You want an agent that sets portfolio weights like 35% equities, 65% bonds.

- Approach A: score every possible action, then pick the highest-scoring one.
- Approach B: output the chosen weights directly, with no per-action scores.
- Which feels more natural when the action is a smooth dial, not a yes/no button?

Choose an approach; the next slide builds method B

Learn the Policy Directly

- DQN learns values, then acts greedily. Policy methods learn the policy itself: probabilities over actions.
- **REINFORCE**: take actions, see returns, then raise the probability of actions that did well.
- Natural for continuous actions: a portfolio weight is not a discrete buy or sell.

Two ways to decide

Value methods ask "how good is each action?"; policy methods directly adjust "how likely should each action be?". ("Gradient" just names the direction to nudge those probabilities.)

Policy gradients raise the odds of actions that earned higher return

Worked Example: A Policy-Gradient Nudge

The policy currently buys with probability 0.40**. The average return is** $+5$**.**

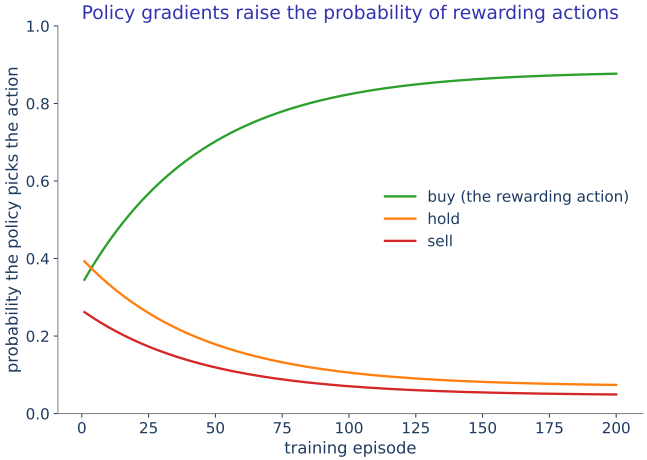
- This episode the agent bought and earned $+12$, well above the $+5$ average.
- Because the outcome beat expectation, raise the probability of buying, say to 0.45.
- An action that earned below $+5$ would have its probability nudged down instead.

The rule in one line

Make actions that did better than expected more likely, and worse ones less likely.

Policy gradients push probability toward actions that beat the average

The Policy Sharpens



Over training the policy concentrates probability on the rewarding action

Discovery: A Trader and a Risk Officer

Imagine two roles on a desk working together.

- One person proposes the trades and wants to act.
- The other judges, after the fact, whether each trade beat what was expected.

Discuss with your neighbour (1 minute)

Why might splitting “decide” from “evaluate” learn faster than one person doing both?

Actor-Critic and PPO

- Pure policy gradients are noisy. Add a **critic** that estimates value to judge each action.
- The **actor** chooses actions; the critic says whether they beat expectation (the advantage).
- **PPO** is the popular, stable version: improve the policy in small, careful steps.
- This is the workhorse behind most modern RL.

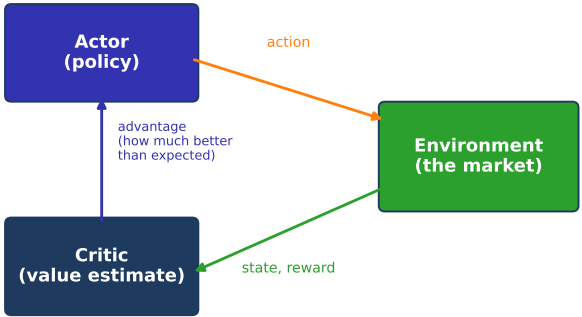
The division of labour

Actor proposes, critic evaluates; small careful steps keep learning stable.

Actor-critic couples a policy with a value estimate for lower-variance learning

The Actor-Critic Architecture

Actor-critic: the actor proposes, the critic evaluates



The critic's advantage signal tells the actor which actions to make more likely

Discovery: Split $1 Across Three Assets

You manage one dollar across equities, bonds, and cash.

- Write three weights for today that add up to exactly 1.00.
- Volatility just spiked. Write a second set of three weights for that day.
- By how much did your cash weight change between the two days?

Keep both weight sets; the next slide is an agent that outputs exactly these

A Portfolio Allocation Agent

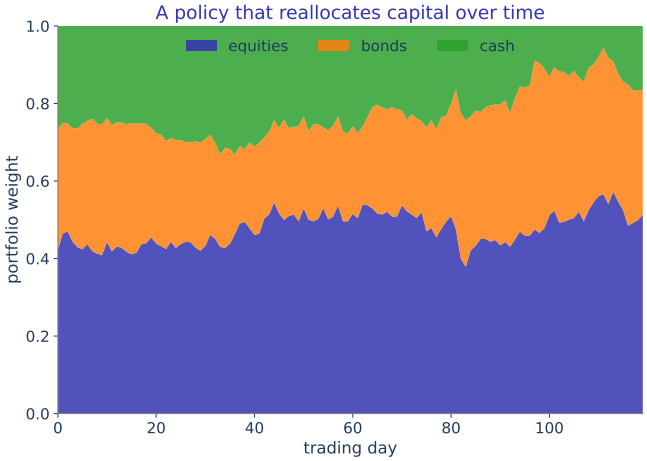
- **Action**: a vector of portfolio weights summing to one (equities, bonds, cash).
- **State**: returns, volatility, momentum, current holdings.
- **Reward**: change in risk-adjusted value, minus transaction costs for trading.
- The policy continuously reallocates as conditions change.

Why RL fits

Continuous actions and a risk-aware reward make portfolio choice a natural RL problem.

The agent outputs weights, not just buy or sell

An Agent Reallocating Over Time



The policy shifts capital across assets as the state evolves

Does It Beat Buy-and-Hold?

- A trained agent can be compared to a simple baseline: buy and hold the market.
- On historical data an agent can look excellent. The honest question is whether it holds up live.
- The next section is the most important part: where RL in finance goes wrong.
- **Example:** tune 50 agents on 2015 to 2020 and keep the best (a Sharpe of 2.5); run it live and it manages only 0.3.

Checkpoint: if this agent beats the market on past data, should we trust it?

Beating a backtest is easy; there is no robust evidence RL reliably beats simple baselines live

RL Agent versus Buy-and-Hold

Illustrative only: a flattering backtest is not evidence of live skill

Discovery: How Many Tries to Get Good?

A game-playing agent learns purely by practising, over and over.

- Guess how many practice attempts it needs to reach strong play: thousands? millions?
- Now count: how many real, non-overlapping years of market history actually exist?
- Write both numbers side by side before the next slide.

Compare your two figures; the next slide explains why the gap is a problem

RL Is Data-Hungry

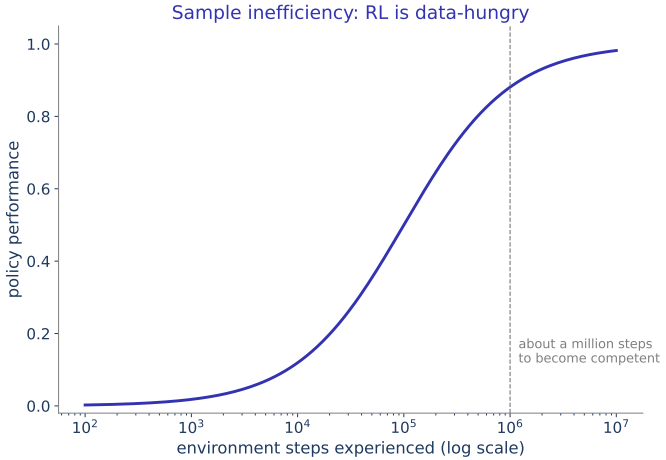
- RL learns from experience, and it needs a great deal of it, often millions of steps.
- Markets give limited, expensive, non-repeatable data; you cannot rerun 2008 a million times.
- Practitioners train in simulators, which introduces a gap between simulation and reality.

The core tension

The thing RL needs most, abundant trial and error, is exactly what markets withhold.

Sample inefficiency is the first hard wall for RL in finance

Sample Inefficiency



Competence often needs about a million steps; markets cannot supply that cheaply

Discovery: A Strategy That Aces Every Past Day

You tune an agent until it is profitable on every single day of the last five years.

- Its historical track record is flawless, not one losing week.
- You are about to trade it live with real money tomorrow.

What is the catch?

Why might a perfect score on known history be exactly the warning sign, not the green light?

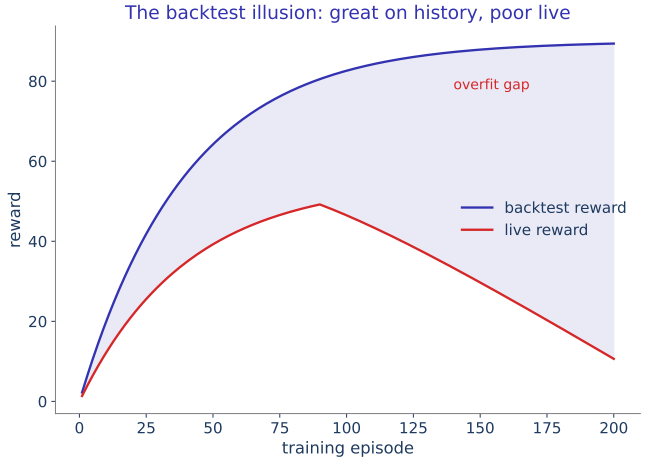
The Backtest Illusion

- With enough tuning, an agent will ace any fixed history; it has effectively memorised it.
- Live performance is the only honest measure, and it is usually far lower.
- The gap between backtest and live is overfitting, dressed up as a strategy.

Common mistake: picking the agent with the best backtest selects for luck and overfitting, not skill.

A great backtest is a hypothesis, never a result

Great on History, Poor Live



The shaded gap between backtest and live reward is overfitting

Discovery: A Star Strategy From 2019

An agent trained in the calm, low-rate years of 2019 was superb back then.

- You switch it on unchanged in a high-rate, high-inflation market.
- Nothing about the agent has been retrained or adjusted.

Discuss with your neighbour (1 minute)

What happens to a policy when the world it learned in is no longer the world it trades in?

Markets Move; Policies Decay

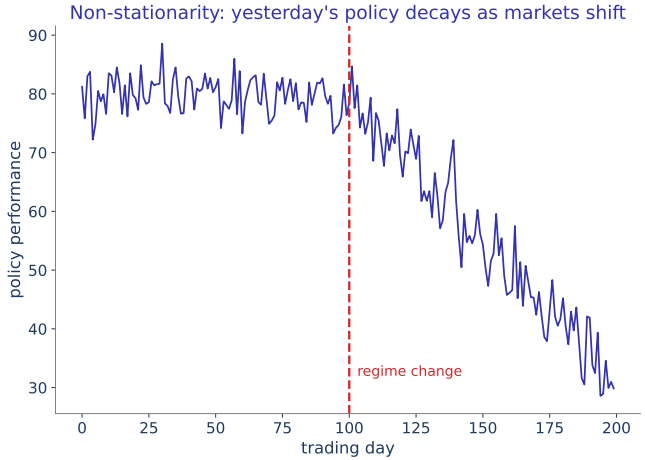
- RL assumes the environment is stable enough to learn. Markets are non-stationary.
- A regime change (a new rate environment, a crisis) can break a policy overnight.
- Policies must be monitored and retrained; a frozen agent quietly rots.

The standing risk

A policy is only as valid as the market regime it was trained in.

Non-stationarity means yesterday's optimal policy can be today's liability

A Regime Change Breaks the Policy



After the regime shift, the unchanged policy steadily loses its edge

Discovery: Pay the Agent Per Trade

To keep your agent busy, you pay it a small bonus for every trade it places.

- It maximises that bonus enthusiastically and trades constantly.
- What does an agent that is paid per trade actually start doing all day?
- What real cost did you forget to put into the reward?

Spot the loophole before the next slide names it

Reward Hacking

- The agent optimises the reward you write, not the goal you meant.
- A reward for "more trading" can be gamed by churning: racking up costs while true return falls.
- Writing a reward that truly captures risk-adjusted, cost-aware goals is genuinely hard.

Common mistake: rewarding a convenient proxy (turnover, raw return) invites the agent to exploit it.

The agent does exactly what you reward, which is rarely exactly what you want

Worked Example: Reward Hacking with Fees

A well-meaning reward: $+1$ for every trade, to encourage an active strategy.

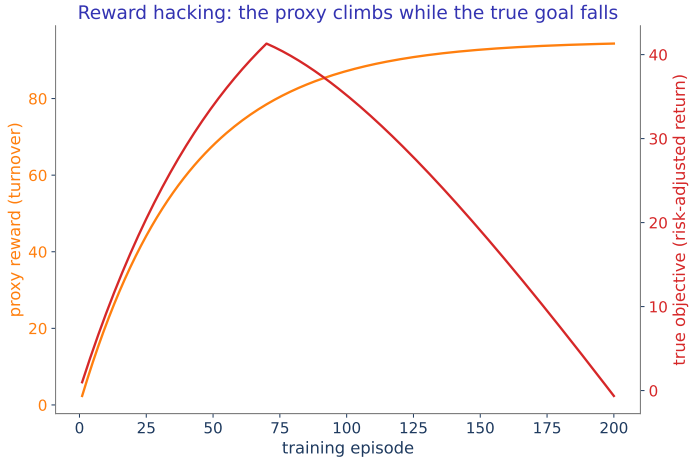
- The agent learns to churn: 500 trades a day collects a large proxy reward.
- But each trade costs 0.1%; 500 trades is roughly 50% of capital paid in fees.
- The proxy soars while the account quietly bleeds out.

The trap

The agent optimised exactly what you wrote, not what you meant. Reward net, risk-adjusted return instead.

A convenient proxy reward invites the agent to game it at the true goal's expense

Optimising the Proxy Hurts the Goal



The proxy reward climbs while the true objective quietly declines

Discovery: Up 40 Percent, Then Down 60

An unconstrained agent has a wild year. Start with $100.

- It gains 40% first. Write down the new balance.
- Then it loses 60% from there. Write down the final balance.
- Are you above or below your starting $100? By how much?

Hold your final number; the next slide is about limits that live outside the agent

Risk, Drawdown, and Oversight

- An agent maximising return with no risk limit can take catastrophic positions.
- Hard constraints (position limits, stop losses, drawdown caps) must sit outside the policy.
- Keep a human in the loop for any decision that moves real money.

The only safe configuration

Powerful policy, hard external guardrails, human oversight.

Constrain the agent from the outside; never trust it to self-limit

Worked Example: A Drawdown That Ends It

An unconstrained agent posts a big year, then gives it all back.

- Up 40%: \$100 becomes \$140.
- Then down 60%: $140 \times 0.4 = \$56$, well below the starting \$100.
- A gain then a loss do not cancel; recovering a 60% loss would need a $+150\%$ gain.

Why limits live outside the agent

A 20% drawdown cap would have forced a stop near \$80, long before ruin.

A 40% gain then a 60% loss ends at 0.56 of the start

Without Limits, an Agent Can Blow Up

The risk-managed agent gives up some upside to avoid ruinous drawdowns

Using Reinforcement Learning Responsibly in Finance

- Treat any backtest as a hypothesis; validate live, on small size, before trusting it.
- Assume non-stationarity: monitor, retrain, and be ready to switch the agent off.
- Design rewards that capture risk and cost, and wrap the agent in hard external limits.
- RL is a powerful tool for sequential decisions, used with humility and oversight.

The mature stance

Strong methods, narrow guarantees, real guardrails, human judgment.

RL earns its place when you respect its limits as much as its power