Reinforcement Learning

Mini-Lecture: Learning from Trial and Error

Methods & Algorithms

MSc Data Science – Spring 2026

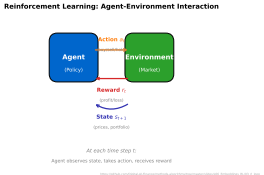
When Machines Learn by Doing

XKCD #1838 by Randall Munroe (CC BY-NC 2.5)

The Agent-Environment Loop

The RL Framework

- An **agent** observes the current **state** s_t of the environment
- It chooses an **action** a_t based on its policy π
- The environment returns a **reward** r_t and transitions to a new state s_{t+1}
- Goal: learn a policy that maximizes cumulative reward $\sum_{t=0}^{T} \gamma^t r_t$



Key Difference from Supervised Learning

No labeled data. The agent must discover which actions are good through trial and error – reward is the only signal.

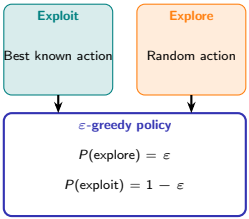
$\gamma \in [0, 1]$ **is the discount factor – it controls how much the agent cares about future vs immediate rewards**

The Exploration-Exploitation Dilemma

Explore or Exploit?

- **Exploit**: choose the action with highest known reward – greedy, but may miss better options
- **Explore**: try a random action – costly short-term, but discovers new opportunities
- **ε-greedy**: with probability ε explore randomly, otherwise exploit the best known action

Typical schedule: start with $\varepsilon = 1.0$ (all exploration) and decay to $\varepsilon = 0.01$ over training.



Insight

This is the same dilemma a trader faces: stick with a known profitable strategy or experiment with a new one?

Multi-armed bandit: the simplest RL problem. *N* **slot machines with unknown payout distributions – how do you maximize winnings?**

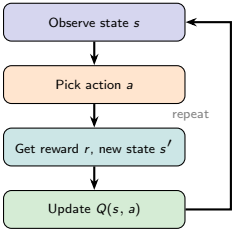
Q-Learning: Estimating Future Rewards

The Q-Value Function

- $Q(s, a)$ = expected cumulative reward from taking action a in state s, then acting optimally
- **Update rule** (after observing reward r and next state s'):

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

- α: learning rate (step size)
- γ: discount factor (patience)
- The term in brackets is the **temporal difference error**



Insight

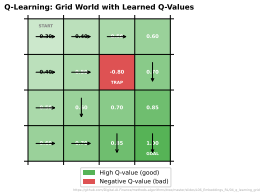
Q-learning is model-free: it learns the value of actions without knowing the environment's transition dynamics.

Q-learning converges to the optimal policy given infinite exploration and a decaying learning rate (Watkins & Dayan, 1992)

Q-Table: A Simple Grid World Example

Learning to Navigate

- Agent starts in a grid, goal is to reach the reward cell
- Q-table stores values for every (state, action) pair
- After many episodes, high Q-values form a path to the goal
- The agent's policy: in each cell, pick the action with highest Q



Insight

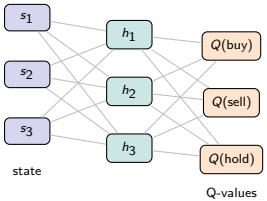
The Q-table is a lookup table – it works for small state spaces but becomes impractical when states are continuous or high-dimensional.

A 4×4 grid with 4 actions has 64 Q-values. A stock trading environment with continuous prices has infinitely many – hence deep Q-networks.

Deep Q-Networks: Scaling Up with Neural Nets

From Table to Network

- Replace the Q-table with a neural network: $Q(s, a; \theta) \approx Q^*(s, a)$
- Input: state s; output: Q-values for all actions
- **Experience replay**: store past transitions, sample random mini-batches to break correlation
- **Target network**: a frozen copy of Q updated periodically to stabilize training



Insight

DQN (Mnih et al., 2015) learned to play Atari games from raw pixels – the same architecture can learn trading policies from price histories.

Experience replay and target networks were the two key innovations that made deep RL stable enough to work in practice

Reward Design: Shaping the Objective

The Reward Is Everything

- The agent optimizes exactly what you reward – nothing more, nothing less
- **Sparse rewards**: only signal at episode end (e.g., final PnL). Hard to learn from.
- **Shaped rewards**: intermediate signals (e.g., Sharpe ratio per step). Faster learning but risk of reward hacking.
- **Credit assignment**: which past action caused today's reward? Delayed rewards make learning harder.

Trading Reward Examples:

Naive: $r_t = \text{PnL}_t$
$\rightarrow$ Agent takes maximum leverage

Better: $r_t = \frac{\text{PnL}_t}{\sigma_t}$ (Sharpe)
$\rightarrow$ Balances return and risk

With constraint: penalize drawdown
$r_t = \text{Sharpe}_t - \lambda \cdot \text{DD}_t$

Insight

Reward design is the hardest part of RL. A badly designed reward function produces an agent that "cheats" – technically optimal but practically useless.

Goodhart's Law: "When a measure becomes a target, it ceases to be a good measure" – applies directly to RL reward design

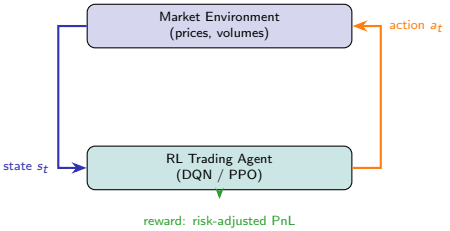
Finance Application: Algorithmic Trading with RL

RL for Portfolio Management

- **States**: price history, technical indicators, portfolio positions, market regime
- **Actions**: buy, sell, hold (or continuous: allocation weights)
- **Reward**: risk-adjusted return (Sharpe ratio, Sortino ratio)
- Agent learns when to enter/exit positions without explicit rules

Insight

RL is not widely deployed in live trading yet – simulation-to-reality gap, non-stationarity, and reward design remain open challenges.



JPMorgan's LOXM uses RL for optimal trade execution; other firms explore RL for market making and portfolio rebalancing

Summary: Reinforcement Learning in 4 Takeaways

1. **Framework**: Agent observes state, takes action, receives reward. Goal: maximize cumulative discounted reward $\sum \gamma^t r_t$.
2. **Exploration**: ε-greedy balances trying new actions (explore) with choosing the best known action (exploit). Start high, decay over time.
3. **Q-Learning**: Learn $Q(s, a)$ via temporal difference updates. Q-tables work for small spaces; DQN scales to complex environments.
4. **Finance**: RL enables data-driven trading, but reward design, non-stationarity, and the simulation-to-reality gap remain open challenges.

Next Steps

Explore the deep dive slides for policy gradient methods, actor-critic architectures, and multi-agent RL. Try the notebook to train a Q-learning agent on a simulated trading environment.

RL is the most ambitious ML paradigm: learn behavior from interaction, not from labels. Finance is a natural but challenging domain.