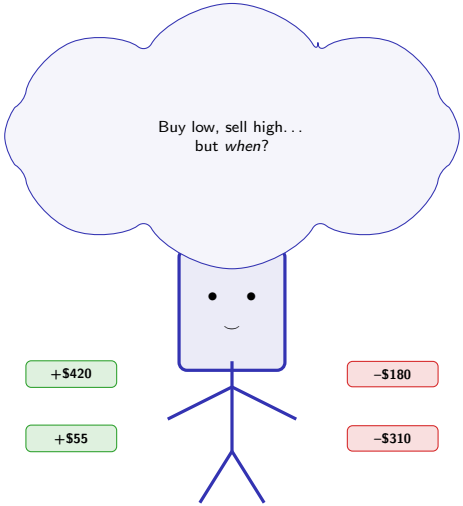
Reinforcement Learning

Learning to Act from Rewards

Methods & Algorithms

MSc Data Science – Spring 2026

Can a Machine Learn to Trade Stocks by Trial and Error?



RL agents learn from outcomes, not from labeled examples.

Learning Objectives

After this lecture, you will be able to:

1. **Formalize** sequential decisions as Markov Decision Processes
2. **Derive** the Bellman equation and implement Q-learning
3. **Analyze** exploration-exploitation trade-offs using epsilon-greedy
4. **Apply** RL to algorithmic trading and portfolio management

---

LOs at Bloom's levels 4–5: formalize, derive, analyze, apply.

What Is Reinforcement Learning?

An **agent** learns to make decisions by interacting with an **environment** and receiving **rewards**.

- Not supervised: no labeled "correct actions"
- Not unsupervised: there is a goal (maximize reward)
- Learning by **trial and error**

Key Terms

- **Agent**: the learner / decision-maker
- **Environment**: the world it acts in
- **State**: current situation
- **Action**: what the agent does
- **Reward**: feedback signal
- **Policy**: action selection strategy

RL sits between supervised and unsupervised learning: it has a goal but no labels.

Why Not Just Use Supervised Learning?

Supervised Learning

- Needs labeled data (x, y)
- "What is the correct action?" requires a label
- Assumes i.i.d. samples

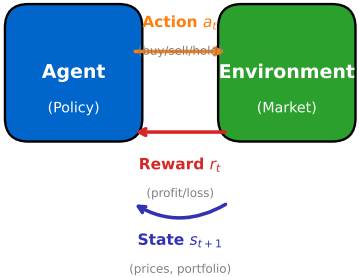
Reinforcement Learning

- Learns from outcomes (reward signals)
- No one tells you the "right" trade
- Delayed rewards: today's trade affects tomorrow

In trading, you only know if a decision was good after the market moves.

The Agent-Environment Loop

Reinforcement Learning: Agent-Environment Interaction



At each time step t:

Agent observes state, takes action, receives reward

https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L06_Embeddings_RL/03_rl_loop

State → Action → Reward → Next State → repeat.

Markov Decision Process (MDP)

Definition

An MDP is a tuple (S, A, P, R, γ):

- S: set of states A: set of actions
- $P(s' \mid s, a)$: transition probability
- $R(s, a)$: reward function $\gamma \in [0, 1]$: discount factor

Markov property: the future depends only on the current state, not the history.

$$P(S_{t+1} \mid S_t, A_t) = P(S_{t+1} \mid S_0, A_0, \ldots, S_t, A_t)$$

The MDP formalism is the mathematical foundation of all RL algorithms.

The Return and Value Functions

Return (discounted cumulative reward):

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

State-value function (how good is state s?):

$$V^\pi(s) = \mathbb{E}_\pi\left[G_t \mid S_t = s\right]$$

Action-value function (how good is action a in state s?):

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

γ close to 1: far-sighted agent. γ close to 0: myopic agent.

The Bellman Equation

The optimal action-value satisfies a **recursive** relationship:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q^*(s',a')$$

This expands the $\mathbb{E}[\cdot]$ form from the overview by substituting transition probabilities $P(s' \mid s, a)$.

Worked example ($\gamma = 0.9$, deterministic transitions):

- State A $\xrightarrow{\text{right}}$ State B, reward $= +1$
- State B $\xrightarrow{\text{right}}$ State C (terminal, $Q = 0$ by definition), reward $= +10$
- $Q(B, \text{right}) = 10 + 0.9 \times 0 = 10$
- $Q(A, \text{right}) = 1 + 0.9 \times 10 = 10$

The Bellman equation decomposes value into immediate reward + discounted future.

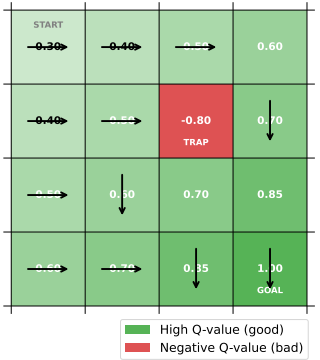
Q-Learning: Model-Free Temporal Difference

Update rule:

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

- α: learning rate
- No transition model P needed!
- Converges to Q^* with sufficient exploration

Q-Learning: Grid World with Learned Q-Values



https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L06_Embeddings_RL/04_q_learning_grid

Q-learning is off-policy: it learns the optimal Q regardless of the behavior policy.

Exploration vs Exploitation

The Dilemma

- **Exploit**: choose the best known action
- **Explore**: try a random action to discover more
- Too much exploitation $\rightarrow$ stuck at local optimum
- Too much exploration $\rightarrow$ never converges

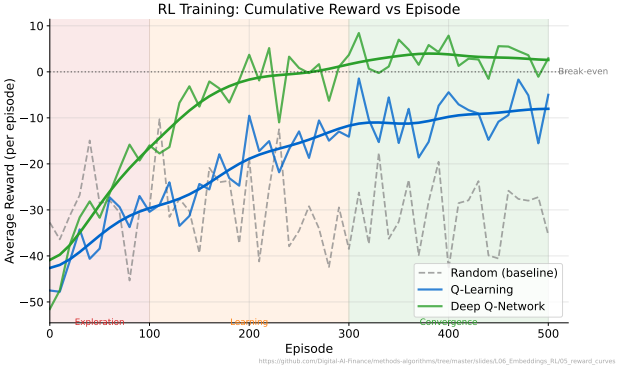
Epsilon-Greedy Policy

- With probability ε: random action
- With probability $1 - \varepsilon$: greedy action
- **Decaying** ε: explore early, exploit later

Balancing exploration and exploitation is a fundamental challenge in RL.

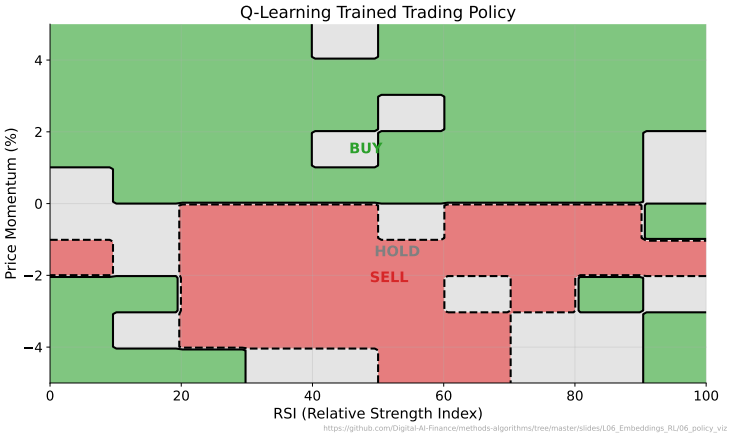
Watching the Agent Learn

- Early episodes: near-random, low reward
- Mid training: discovers good paths
- Late training: converges to optimal policy



The learning curve shows cumulative reward increasing as the agent improves.

Visualizing the Learned Policy

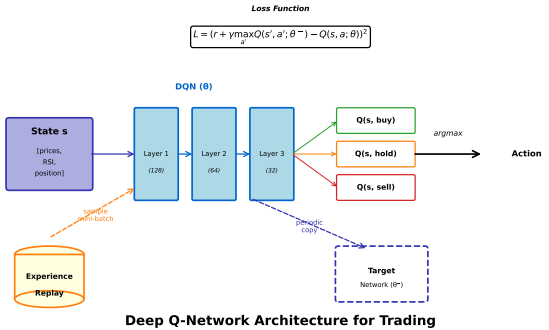


Arrows show the optimal action in each state – the agent has learned to navigate.

Deep Q-Networks (DQN)

- Replace Q-table with a **neural network**
- Input: state → Output: $Q(s, a)$ for all a
- **Experience replay**: store and resample transitions
- **Target network**: stabilize training

Scales to continuous or very large state spaces.



Deep Q-Network Architecture for Trading

DQN (Mnih et al., 2015) enabled RL to play Atari games from raw pixels.

Algorithmic Trading as an MDP

MDP Formulation

- **State**: (price, position, time)
- **Actions**: buy, sell, hold
- **Reward**: realized P&L per step

Why RL Fits

- Sequential decisions over time
- Delayed reward (hold for days)
- No "correct label" for each action

The trading agent learns when to enter and exit positions to maximize returns.

Portfolio Optimization with RL

Setup

- **State**: portfolio weights + market features
- **Action**: rebalance weights
- **Reward**: risk-adjusted return (Sharpe ratio)

Advantages

- Handles transaction costs naturally
- Adapts to changing market regimes
- Optimizes multi-period objectives

RL-based portfolio management optimizes the Sharpe ratio while accounting for costs.

Optimal Execution: Minimizing Market Impact

Problem: Execute a large order without moving the market.

1. Split the order into smaller slices over time
2. **State**: remaining shares, time left, market volatility
3. **Action**: how many shares to trade now
4. **Reward**: $-$slippage cost (minimize market impact)

The RL agent learns to trade more aggressively in calm markets and slow down during volatile periods.

Optimal execution is one of the most commercially successful RL applications in finance.

Reward Engineering: The Hardest Part of RL

Bad Reward Design

- Maximize raw return $\rightarrow$ extreme leverage
- Maximize win rate $\rightarrow$ tiny gains, huge losses
- No drawdown penalty $\rightarrow$ reckless risk

Better Reward Design

- Sharpe ratio: return / volatility
- Drawdown penalty: punish large losses
- Risk-adjusted P&L with position limits

"The reward function IS your specification of the problem."

Misaligned rewards produce agents that technically optimize but behave badly.

RL Challenges in Finance

Technical Challenges

- **Non-stationarity**: markets change over time
- **Partial observability**: you cannot see everything
- **Sample efficiency**: live trading data is expensive

Practical Challenges

- **Sim-to-real gap**: backtests $\neq$ live markets
- **Regime changes**: bull $\rightarrow$ bear invalidates policy
- **Regulatory**: explain decisions to compliance

These challenges are why RL in finance remains an active research frontier.

When to Use RL – and When Not To

Pros

- Handles sequential decision problems
- No labeled data required
- Adapts to changing environments

Cons

- Slow training, high sample cost
- Reward design is hard to get right
- Sim-to-real gap in finance
- Difficult to interpret and debug

Use RL when decisions are sequential and delayed – not for one-shot predictions.

RL vs Supervised Learning vs Optimization

	Supervised	**RL**	**Optimization**
Data	Labeled (x, y)	Reward signal	Objective function
Sequential	No	Yes	Sometimes
Adapts	Retrain needed	Online learning	Re-solve
Interpretable	Model-dependent	Low	High
Training time	Fast	Slow	Medium

Each paradigm has its sweet spot – RL shines for sequential, adaptive problems.

Hands-on: Train a Q-Learning Agent

```
import numpy as np
# 5x5 gridworld: start (0,0), goal (4,4)
Q = np.zeros((25, 4))  # 25 states, 4 actions
alpha, gamma, epsilon = 0.1, 0.99, 0.1

for episode in range(500):
    s = 0  # start state
    while s != 24:  # goal state
        if np.random.rand() < epsilon:
            a = np.random.randint(4)
        else:
            a = np.argmax(Q[s])
        s_next, r = step(s, a)  # environment step
        Q[s,a] += alpha * (r + gamma*np.max(Q[s_next]) - Q[s,a])
        s = s_next
```

Full notebook: `L06_rl.ipynb`

Try the notebook to train your own gridworld agent and visualize the policy.

Key Takeaways

Concepts

- MDP formalizes sequential decisions
- Bellman equation decomposes value recursively
- Q-learning is model-free and off-policy
- DQN scales Q-learning to large state spaces

Practice

- Start with tabular Q-learning
- Reward design matters more than algorithm
- Always compare to a simple baseline
- Beware overfitting to backtests

RL is powerful for sequential decisions but demands careful reward engineering.

Course Wrap-Up: From Regression to RL

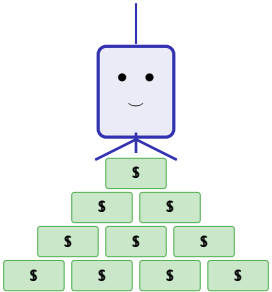
- **L01**: Predict (Linear Regression)
- **L02**: Classify (Logistic Regression)
- **L03**: Cluster (KNN & K-Means)
- **L04**: Ensemble (Random Forests)
- **L05**: Reduce (PCA & t-SNE)
- **L06**: Represent & Act (Embeddings & RL)

Your ML toolbox is complete.

Six sessions, six paradigms – from prediction to decision-making under uncertainty.

Closing Thought

"I learned this all by trial and error...
mostly error."



Thank you for a great semester! Good luck with your group assignments.