NEW YORK UNIVERSITY

Lab Review

Reinforcement Learning Review

Solim LeGris
New York University

Lab review session covering key RL concepts that you will see in the class.

Today's Goals

What we'll cover

1. **The RL Framework** - MDPs, states, actions, rewards
2. **Value Functions & Bellman Equation** - The heart of RL
3. **Key Algorithms** - DP, Monte Carlo, TD Learning
4. **Q-Learning vs SARSA** - On-policy vs off-policy
5. **Exploration vs Exploitation** - The tradeoff
6. **Quick Overview** - Deep RL and policy gradients

Types of Learning

Supervised Learning

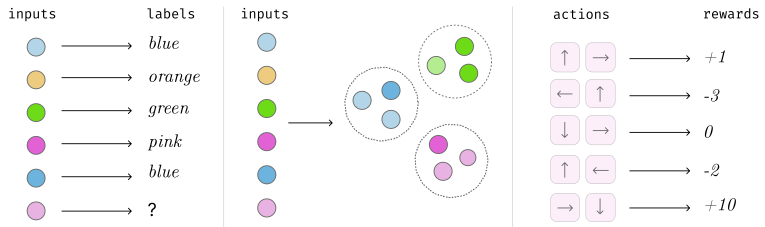
- Corrective feedback: "Your answer should have been X"
- Learn from labeled examples

Unsupervised Learning

- No feedback
- Find patterns and structure in data

Reinforcement Learning

- Evaluative feedback: "That was good/bad"
- NOT corrective: doesn’t tell you the right answer
- Agent must discover good actions through trial and error



Supervised Learning
Inputs are paired with “ground truth” labels which provide corrective feedback to the learning algorithm.

Unsupervised Learning
Inputs are presented with no explicit feedback. Learning occurs by detecting the latent structure of the input, e.g. clustering inputs based on their similarity to one another.

Reinforcement Learning
The algorithm takes actions (e.g., button presses in a game) and is given feedback about if behavior was good or bad only in a relative sense.

The RL Problem: Key Components

Agent - The learner/decision maker

Environment - Everything the agent interacts with

State (s) - Current situation/configuration

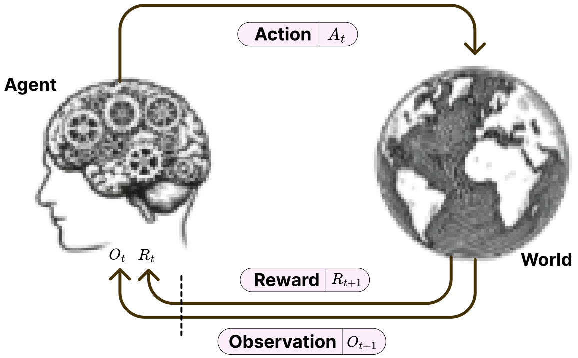
Action (a) - Choices available to the agent

Reward (r) - Immediate feedback signal

- Positive = good, Negative = bad

Policy (π) - Strategy for choosing actions

- $\pi(s, a)$ = probability of action a in state s



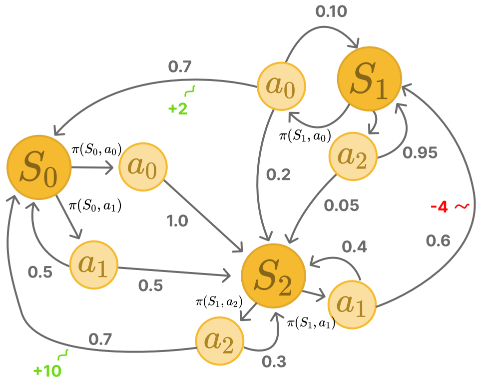
Goal: Learn a policy that maximizes cumulative reward over time

Markov Decision Process (MDP)

Formal framework for RL problems

An MDP is defined by tuple $\langle S, A, T, R, \gamma \rangle$:

- S: Set of states
- A: Set of actions
- T: Transition function $\mathcal{P}_{ss'}^{a} = P(s'|s,a)$
- R: Reward function $\mathcal{R}_{ss'}^{a}$
- γ: Discount factor (0 to 1)



Markov Property: Future depends only on current state, not history

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_1, a_1, \ldots, s_t, a_t)$$

Returns and Discounting

The agent's goal is to maximize cumulative discounted reward (the **return**):

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

If $\gamma = 0$

Only immediate reward matters

"Myopic" agent

If $\gamma = 1$

All future rewards weighted equally

Works for finite tasks

If $0 < \gamma < 1$ (typical)

Balance immediate vs. delayed rewards

More distant rewards matter less

Value Functions

State-Value Function $V^\pi(s)$

Expected return starting from state s and following policy π:

$$V^\pi(s) = E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid s_t = s\right\}$$

"How good is it to be in this state?"

Action-Value Function $Q^\pi(s,a)$

Expected return starting from state s, taking action a, then following π:

$$Q^\pi(s,a) = E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

"How good is taking this action in this state?"

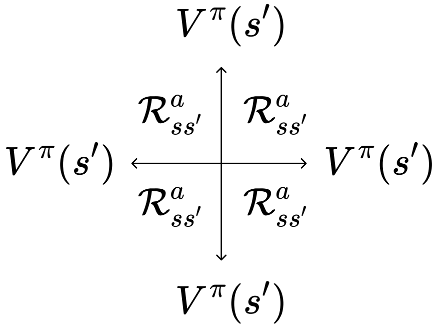
Key Relationship: $V^\pi(s) = \sum_a \pi(s,a)Q^\pi(s,a)$

State value = average over action values weighted by policy

The Bellman Equation

The recursive relationship that defines value functions:

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s')\right]$$



Key Insight: Value of a state depends on:

1. Immediate reward you get
2. Discounted value of where you end up

This is the foundation of bootstrapping!

Two Critical Problems in RL

Credit Assignment

Rewards are often delayed from the actions that caused them.

- You score a goal, but which pass led to it?
- You fail an exam, but which study decisions were wrong?

How RL helps: Propagates value backward through states using bootstrapping

Exploration vs Exploitation

- **Exploit**: Choose best known action
- **Explore**: Try new actions to learn more

The best long-term strategy may require short-term sacrifices to gather information!

How RL helps: Various strategies (epsilon-greedy, softmax, UCB)

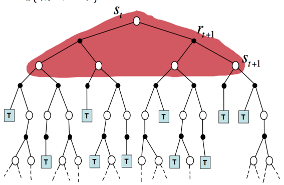
Algorithm Overview: Three Approaches

Dynamic Programming

- Requires **model** of environment
- Sweeps through all states
- Exact solution (given model)
- Bootstrapping: uses estimates to update e

Dynamic Programming

$V(s_t) \leftarrow E_\pi\{r_{t+1} + \gamma V(s_t)\}$



The DP target is an estimate not because of the expected values, which are assumed to be completely provided by a model of the environment, but because V^π is not known and the current estimate is used instead.

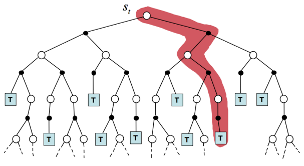
Monte Carlo

- **Model-free**: learns from experience
- Waits until episode ends
- Averages observed returns
- No bootstrapping: uses actual returns

Simple Monte Carlo

$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)]$

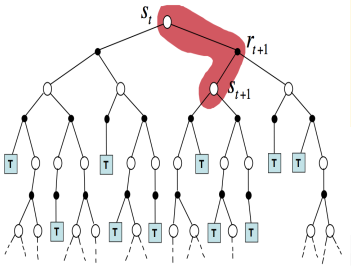
where R_t is the actual return following state s_t.



Monte Carlo uses an estimate of the actual return.

Temporal Difference

- **Model-free**: learns from experience
- Updates **within** episode
- Bootstrapping: uses estimates
- Best of both worlds!



Dynamic Programming: Policy Iteration

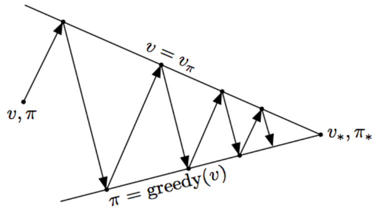
Two alternating steps:

1. **Policy Evaluation**: Given policy π, compute $V^\pi(s)$ for all states

2. **Policy Improvement**: Make policy greedy w.r.t. current values

$$\pi'(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right]$$

Repeat until policy stops changing



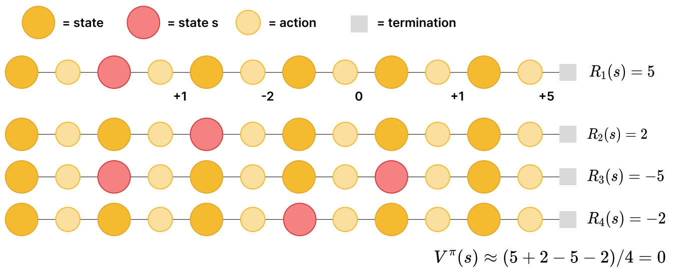
Policy Improvement Theorem:

Greedy improvement always makes policy as good or better!

Limitation: Requires knowing $\mathcal{P}^a_{ss'}$ and $\mathcal{R}^a_{ss'}$ (full model)

Monte Carlo Methods

Idea: Estimate values by averaging observed returns



First-Visit MC Algorithm:

1. Generate episode following policy π
2. For first visit to each state s:
   - Record return R from that point
   - Append to `Returns[s]`
3. $V(s) = \text{average}(\text{Returns}[s])$
4. Repeat for many episodes

$$V(s) = \frac{1}{N(s)} \sum_{i=1}^{N(s)} R_i(s)$$

Advantage: No model needed!

Limitation: Must wait for episode to end

Temporal Difference Learning: TD(0)

Key insight: Update values immediately using bootstrapping

$$V(s_t) = V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$

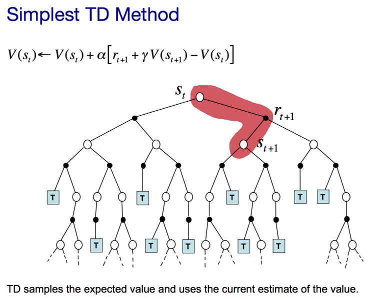
TD Error (prediction error):

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

- Measures "surprise": expected vs actual
- Critical in neuroscience: relates to dopamine!

Why TD is cognitively plausible:

- Trial-by-trial learning
- Incremental updates
- No model required
- Solves credit assignment
- Matches human/animal learning patterns



TD Learning: Interactive Demo

▶ ⏭ ⏩ Ep ×10 ×50 ↻ α 0.90 γ 0.90 ☐ Vπ(DP) Ep1 S0 T0

d 0.0	e 0.0	f★ 0.0
a 0.0	b 0.0	c 0.0

TD(0) Update

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$$

Press **Step** or **Play** to begin

Learning Q-Values: SARSA vs Q-Learning

SARSA (On-Policy)

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$$

- Uses the actual next action a' taken
- Learns value of policy being followed
- More conservative (accounts for exploration)

State-**A**ction-**R**eward-**S**tate-**A**ction

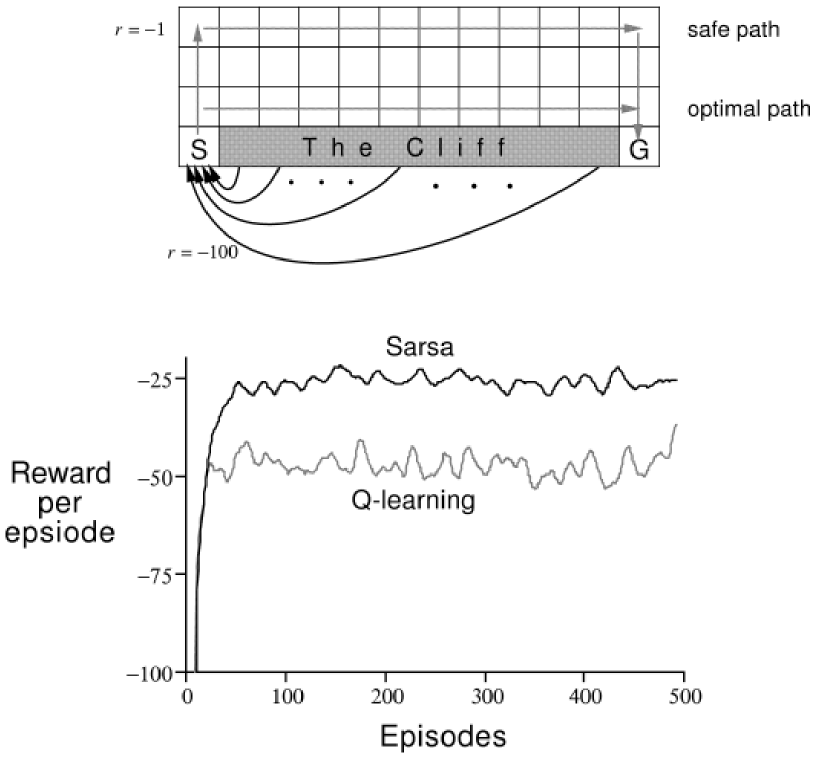
Q-Learning (Off-Policy)

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

- Uses best possible next action (max)
- Learns optimal policy regardless of behavior
- Can learn from any data source

The $\max$ makes it **off-policy**!

SARSA vs Q-Learning: Cliff Walking Example



Q-Learning: Learns optimal path along cliff edge

- But falls off during exploration!

SARSA: Learns safer path along top

- Accounts for its own exploration

Key difference:

- Q-learning assumes optimal future play
- SARSA assumes actual future play (with exploration)

When exploration reduces, both converge to same policy.

Exploration Strategies

Epsilon-Greedy

With probability ϵ: random action With probability $1 - \epsilon$: greedy action

- Simple, guarantees exploration
- All non-greedy actions equally likely

Softmax (Boltzmann)

$$P(a) = \frac{e^{Q(a)/\tau}}{\sum_b e^{Q(b)/\tau}}$$

- $\tau \to 0$: greedy
- $\tau \to \infty$: uniform random
- Popular in psychology/neuroscience

Upper Confidence Bounds (UCB)

$$a_t = \arg\max_a \hat{Q}(a) + \sqrt{\frac{2 \log t}{N(a)}}$$

- Adds "uncertainty bonus"
- Less explored = more uncertainty = explore more
- **Optimism under uncertainty**

Key insight: Information has value!

- Longer horizon = more exploration justified
- Humans adjust exploration with horizon (Wilson et al., 2014)

Function Approximation & Deep RL

Problem: Tabular methods don’t scale

- Too many states to enumerate
- What about states never seen?

Solution: Represent states as features

$$\phi(s) = [x, y, \text{heading}, \text{battery}, \ldots]^T$$

Use neural networks as function approximators:

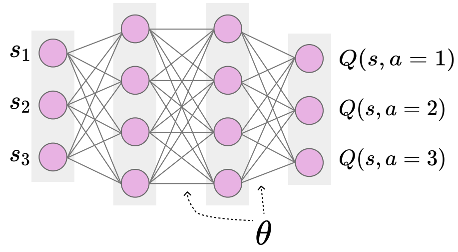
- Input: state features
- Output: Q-values for each action

Key assumption: Similar states have similar values

This is an inductive bias - but reasonable for real environments!

state \ action →				
	0.1	0.3	1.3	0.2
	0.9	0.3	0.7	0.2
	0.3	1.3	0.5	1.2
	0.8	0.3	0.1	5.2
	0.1	0.7	4.7	0.9

$Q(s, a)$



Deep Q-Networks (DQN)

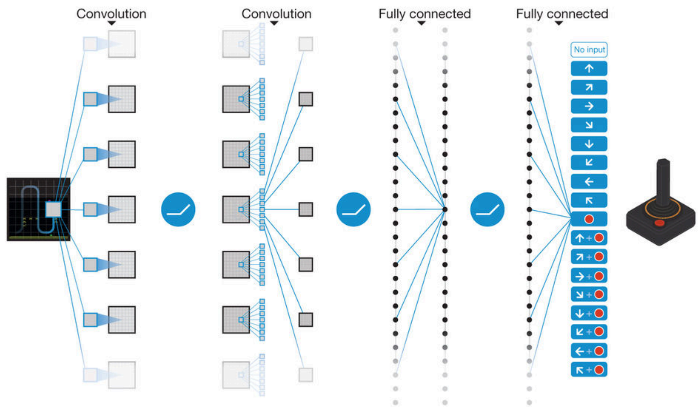
Combine Q-learning with neural networks:

$$L = \frac{1}{2}\left[r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta)\right]^2$$

Key innovations:

1. **Target Network**: Separate copy for computing targets, updated periodically (stabilizes learning)

2. **Experience Replay**: Store experiences, sample randomly (decorrelates data)



Mnih et al. (2015): Mastered Atari from raw pixels!

Policy Gradient Methods

Alternative approach: Learn policy directly!

Parameterize policy as $\pi(a|s, \theta)$

Objective: maximize expected return

$$J(\pi_\theta) = E_{\tau \sim \pi}[R(\tau)]$$

REINFORCE algorithm:

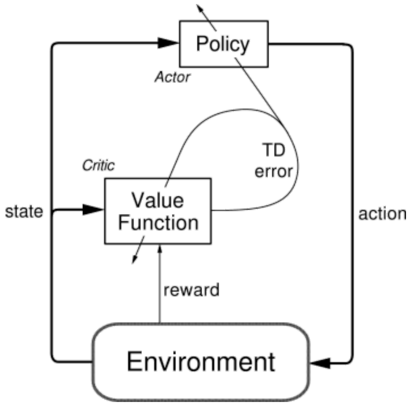
$$\nabla_\theta J = E\left[\sum_{t=0}^{T} R_t \nabla_\theta \log \pi(a_t|s_t, \theta)\right]$$

Intuition:

- $R_t > 0$: increase probability of action
- $R_t < 0$: decrease probability of action

Actor-Critic: Combine both approaches

- **Actor**: learns policy
- **Critic**: learns value function



Summary: Algorithm Comparison

Method	Model?	Bootstrapping?	When to Update?	On/Off Policy
Dynamic Prog	Yes	Yes	All states	N/A
Monte Carlo	No	No	Episode end	On-policy
TD(0)	No	Yes	Each step	On-policy
SARSA	No	Yes	Each step	On-policy
Q-Learning	No	Yes	Each step	Off-policy

Key Takeaways:

- TD methods combine best of DP (bootstrapping) and MC (model-free)
- Q-learning's $\max$ makes it off-policy - can learn from any data
- Exploration is crucial - novel information has value!

Key Equations Summary

Bellman Equation:

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')]$$

TD(0) Update:

$$V(s_t) \leftarrow V(s_t) + \alpha[\underbrace{r_{t+1} + \gamma V(s_{t+1})}_{\text{target}} - V(s_t)]$$

Q-Learning:

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

SARSA:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$$

Softmax:

$$P(a) = \frac{e^{Q(a)/\tau}}{\sum_b e^{Q(b)/\tau}}$$

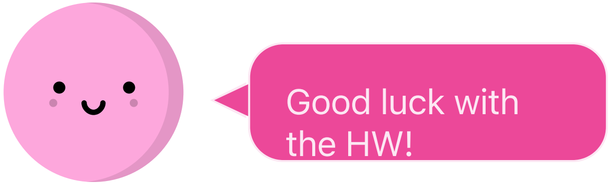
Return:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

For the Homework

The HW review session next week will cover:

- Detailed implementation hints
- Code walkthrough
- Probability concepts you'll need
- Common pitfalls to avoid



Questions?

Key Resources:

- Sutton & Barto (2018): incompleteideas.net/book/the-book-2nd.html
- David Silver's UCL RL Course: YouTube playlist
- Lecture slides 04 & 05