CS 760: Machine Learning
Reinforcement Learning

Misha Khodak

University of Wisconsin-Madison

1 December 2025

Announcements

- **Class roadmap:**
  - Last class on RL
  - Two classes on data-efficient learning
  - Exam review

Outline

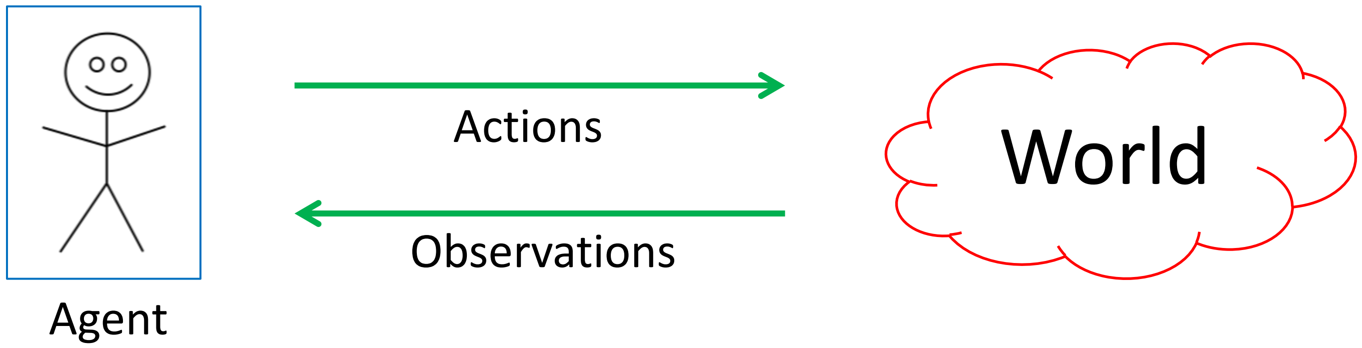
- **Review: RL**
  - MDPs, policies, value function, Q-function, etc.
- **Function Approximation**
  - Value & Q-function approximations, linear, nonlinear
- **Policy-based RL**
  - Policy gradient, policy gradient theorem, REINFORCE algorithm

Outline

- **Review: RL**
  - MDPs, policies, value function, Q-function, etc.
- **Function Approximation**
  - Value & Q-function approximations, linear, nonlinear
- **Policy-based RL**
  - Policy gradient, policy gradient theorem, REINFORCE algorithm

Review: General Model

We have an **agent interacting** with the **world**



- Agent receives a reward based on state of the world
  - **Goal**: maximize reward / utility **($$$)**
  - Note: **data** consists of actions & observations
    - Compare to unsupervised learning and supervised learning

Markov Decision Process (MDP)

The formal mathematical model:

- **State set** S. Initial state s_0. **Action set** A
- **State transition model**: $P(s_{t+1} | s_t, a_t)$
  - Markov assumption: transition probability only depends on s_t and a_t, and not previous actions or states.
- **Reward function:** $r(s_t)$
- **Policy**: $\pi(s) : S \rightarrow A$ action to take at a particular state.

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \dots$$

Defining the Optimal Policy

For policy π, **expected utility** over all possible state sequences from s_0 produced by following that policy:

$$V^{\pi}(s_0) = \sum_{\substack{\text{sequences} \\ \text{starting from } s_0}} P(\text{sequence}) U(\text{sequence})$$

Called the **value function** (for π, s_0)

Discounting Rewards

One issue: these are infinite series. **Convergence**?

- Solution

$$U(s_0, s_1 \ldots) = r(s_0) + \gamma r(s_1) + \gamma^2 r(s_2) + \ldots = \sum_{t \geq 0} \gamma^t r(s_t)$$

- Discount factor γ between 0 and 1
  - Set according to how important **present** is vs. **future**
  - Note: has to be less than 1 for convergence

Bellman Equation

Let's walk over one step for the value function:

$$V^*(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V^*(s')$$

current state reward

discounted expected future **rewards**

Value Iteration

Q: how do we find $V^*(s)$?

- Why do we want it? Can use it to get the best policy
- Know: reward $r(s)$, transition probability $P(s'|s,a)$
- Also know $V^*(s)$ satisfies Bellman equation (recursion above)

A: Use the property. Start with $V_0(s)=0$. Then, update

$$V_{i+1}(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V_i(s')$$

Policy Iteration: Algorithm

Policy iteration. Algorithm

- Start with random policy π
- Use it to compute value function V^π : a set of linear equations

$$V^\pi(s) = r(s) + \gamma \sum_{s'} P(s'|s,a) V^\pi(s')$$

- Improve the policy: obtain π'

$$\pi'(s) = \arg\max_a r(s) + \gamma \sum_{s'} P(s'|s,a) V^\pi(s')$$

- Repeat

Q-Learning (**model-free** RL)

What if we don't know transition probability P(*s*'|*s*,*a*)?

- Need a way to learn to act without it.
- **Q-learning**: get an action-utility function Q(*s*,*a*) that tells us the value of doing *a* in state *s*
- Note: $V^*(s) = \max_a Q(s,a)$
- Now, we can just do $\pi^*(s) = \arg\max_a Q(s, a)$
  - But need to estimate *Q*!

Q-Learning Iteration

How do we get Q(*s*,*a*)?

- Similar iterative procedure

learning rate

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r(s_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Idea: combine old value and new estimate of future value.

Note: We are using a policy π to take actions $a_t = \pi(s_t)$; this policy is based on Q!

Exploration Vs. Exploitation

General question!

- **Exploration:** take an action with unknown consequences
  - **Pros**:
    - Get a more accurate model of the environment
    - Discover higher-reward states than the ones found so far
  - **Cons**:
    - When exploring, not maximizing your utility
    - Something bad might happen
- **Exploitation:** go with the best strategy found so far
  - **Pros**:
    - Maximize reward as reflected in the current utility estimates
    - Avoid bad stuff
  - **Cons**:
    - Might also prevent you from discovering the true optimal strategy

Q-Learning: Epsilon-Greedy Policy

How to **explore**?

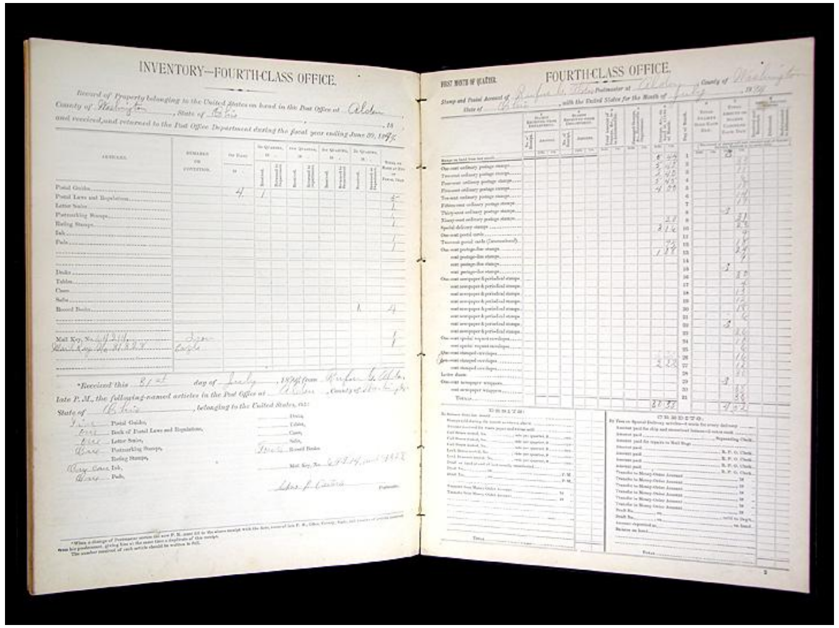
- With some 0<ε<1 probability, take a random action at each state, or else the action with highest Q(*s*,*a*) value.

$$a = \begin{cases} \text{argmax}_{a \in A} Q(s, a) & \text{uniform}(0,1) > \epsilon \\ \text{random } a \in A & \text{otherwise} \end{cases}$$

Outline

- **Review: RL**
  - MDPs, policies, value function, Q-function, etc.
- **Function Approximation**
  - Value & Q-function approximations, linear, nonlinear
- **Policy-based RL**
  - Policy gradient, policy gradient theorem, REINFORCE algorithm

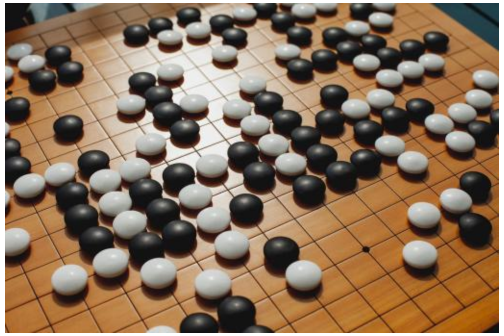
Beyond Tables



So far:

- Represent everything with a table
  - Value function **V**: table size $|S| \times 1$
  - **Q** function: table size $|S| \times |A|$

- Too big to store in memory for many tasks
  - Backgammon: 10^{20} states.
  - Go: 3^{361} states



- Need some other approach

Beyond Tables: Function Approximation

Both V and Q are functions…

- Can approximate them with models, i.e. neural networks
- So we write

$$V^\pi(s) \approx \hat{V}_\theta(s)$$

- New goal: find the weights θ
- Loss function:

$$J(\theta) = \mathbb{E}_\pi[(V^\pi(s) - \hat{V}_\theta(s))^2]$$

State **Representations** & **Models**

How do we represent a state?

- As usual, feature vectors, i.e.

$$x(s) = \begin{bmatrix} x_1(s) \\ x_2(s) \\ \vdots \\ x_d(s) \end{bmatrix}$$

- What kind of models could we use?
  - First, let's start with linear:

$$\hat{V}_\theta(s) = x(s)^T \theta$$

Linear VFA With an Oracle

- SGD update is

$$\alpha[(V^{\pi}(s) - \hat{V}_{\theta}(s))\nabla_{\theta}\hat{V}_{\theta}(s)]$$

- And for our linear model, we get

$$\alpha(V^{\pi}(s) - \hat{V}_{\theta}(s))x(s)$$

Step Size Prediction Error Feature Value

What if We **Don't Have** an Oracle?

Similar to what we've seen so far, use Monte-Carlo.

- We won't know $V^\pi(s_t)$
- Estimate returns $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

- Can just run episodes and estimate, ie, get some noisy estimates as training data:

$$(s_1, G_1), (s_2, G_2), \ldots, (s_T, G_T)$$

Q-Function Approximation

Similar idea for Q-function

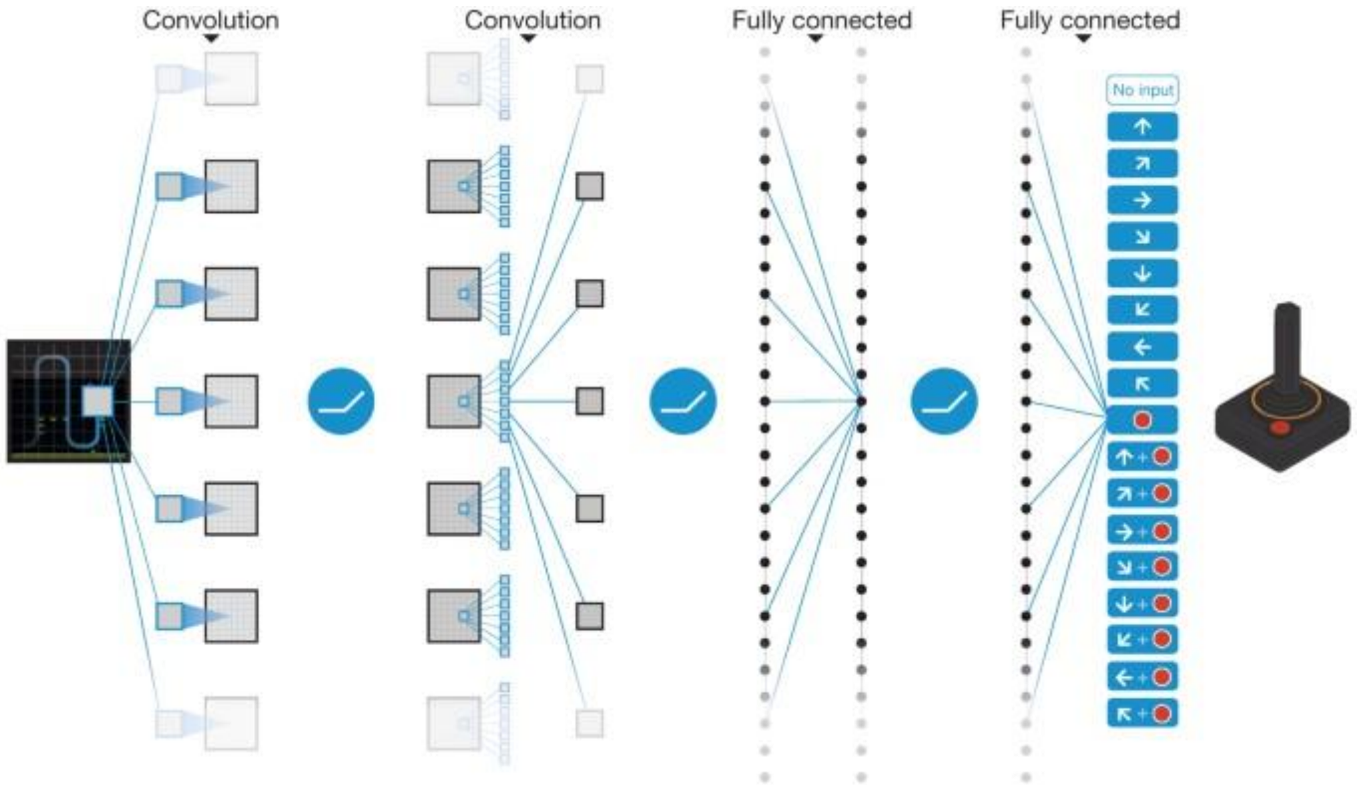
$$Q^{\pi}(s, a) \approx \hat{Q}_{\theta}(s, a)$$

Representation: use both states and values

- Can still use linear models
- Note: quite popular to use **deep models**

Q-Function Approximation: Deep Models

- Note: quite popular to use **deep models**
  - e.g. CNNs if the states are images (like in video games)



Mnih et al, "Human-level control through deep reinforcement learning"

Outline

- **Review: RL**
  - MDPs, policies, value function, Q-function, etc.
- **Function Approximation**
  - Value & Q-function approximations, linear, nonlinear
- **Policy-based RL**
  - Policy gradient, policy gradient theorem, REINFORCE algorithm

Policy-Based RL

So far, we either approximated V or Q

- Then use these to extract the optimal policy

Could do the same trick but with the policy

- Note: so far our policies were deterministic, now we'll allow a distribution over actions, i.e. $\pi(s) = P(a|s)$

- Want: $\pi_\theta(s, a) = P_\theta(a|s)$

Policy Gradient

Use the same idea. We'll define an objective $J(\theta)$

- And then can get gradients:

$$\nabla_\theta \pi_\theta(s,a) = \pi_\theta(s,a) \underbrace{\nabla_\theta \log \pi_\theta(s,a)}_{\text{Score Function}}$$

- Example: continuous action space.
  - Gaussian policy $a \sim \mathcal{N}(x(s)^T\theta), \sigma^2)$
  - has score: $(a - x(s)^T\theta)x(s)/\sigma^2$

Policy Gradient

Set our objective to be

$$J(\theta) = \underbrace{\sum_s P(s|\pi_\theta)}_{\text{stationary distribution}} \sum_a \pi_\theta(s,a) Q^\pi(s,a)$$

- Compute the gradient via the **policy gradient theorem**

$$\nabla_\theta J(\theta) = \sum_s P(s|\pi_\theta) \sum_a \nabla_\theta \pi_\theta(s,a) Q^\pi(s,a)$$

REINFORCE Algorithm

So, to learn a policy, we can run SGD (actually ascent)

- Compute gradients via policy gradient theorem

$$\nabla_\theta J(\theta) = \sum_s P(s|\pi_\theta) \sum_a \nabla_\theta \pi_\theta(s,a) Q^\pi(s,a)$$

- Just need $Q^\pi(s,a)$ estimates.

- How? Monte-Carlo again: Use G_t for our estimates.

Thanks Everyone!

Some of the slides in these lectures have been adapted/borrowed from materials developed by Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, Pedro Domingos, Jerry Zhu, Yingyu Liang, Volodymyr Kuleshov, David Silver, Emma Brunskill, Fred Sala