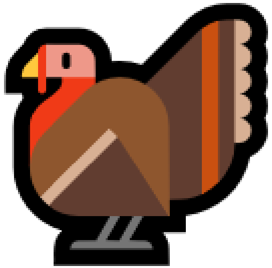
CS 540 Introduction to Artificial Intelligence
Reinforcement Learning II

University of Wisconsin–Madison
November 24, 2025
Fall 2025

Announcements



- Wednesday, November 26
  - **Class is cancelled**
  - Professor Brown’s office hours will be 11:30-12:30

- Homework:
  - HW9 due **Tuesday** Dec 2 at 11:59 pm
  - HW10 released Dec 2, due Tuesday Dec 9 at 11:59 pm

A High-Level View

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.

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

What makes a good policy?

- Want a policy π that gives us lots of reward!
- Value of initial state:

$$V^{\pi}(s_0) = \sum_{\substack{\text{sequences} \\ \text{of states}}} P_{\pi}(s_0, s_1, s_2, \ldots) \cdot U(s_0, s_1, s_2, \ldots)$$

Utility of sequence

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

- Find policy with highest $V^{\pi}(s_0)$
- Write $V^{\pi^*}(s) = V^*(s)$

Finding Good Policies: Version 1

- If we know $P(s' \mid s, a)$ and $V^*(s)$, can compute optimal policy

$$\pi^*(s) = \text{argmax}_a \sum_{s'} P(s'|s, a)V^*(s')$$

All the states we could go to

Transition probability

Expected rewards

$V^*(s)$ is not very useful without transition probabilities!

Finding Good Policies: Version 1

- How can we find $V^*(s)$?
- If we know $P(s' \mid s, a)$ and $r(s)$, can use **value iteration:**

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

Finding Good Policies: Version 2

- What if we don’t know $r(s)$ and $P(s' \mid s, a)$?
- Could learn them from experience!
- Visit all states, take all actions
  - Must do each multiple times

- What are the drawbacks to this approach?

Finding Good Policies, Version 3

- Want to be smarter: spend our time exploring states/actions we believe to be good
- Estimate a **quality** function $Q(s, a)$
- As the agent goes along, simultaneously:
  - Make decisions according to $Q(s, a)$
  - Update $Q(s, a)$ from experience

Examples:
- Q-Learning
- SARSA
- Deep Q-Learning

Finding Good Policies, Version 3

- Estimate a **quality** function $Q(s, a)$

$$Q^*(s,a) = r(s) + \gamma \sum_{s'} P(\, s' \mid s, a\,) \cdot V^*(s')$$

- Make decisions

$$a = \operatorname{argmax}_{a'} Q(s, a')$$

- Q: Why not estimate value function $V(s)$?

Finding Good Policies, Version 4

- Another approach is **policy search**
- Parameterized mapping from states to actions
  - Written π_θ
  - E.g., a neural network

- Update θ as we obtain rewards

We will not
cover these
methods in 540

(Deep) Q-Learning and SARSA

Q-Learning

- Our **main** reinforcement learning algorithm.
- Does not require knowing *r* or *P*. Learn from data of the form:$\{(s_t, a_t, r_t, s_{t+1})\}$.
- Learns an action-value function *Q**(*s*,*a*) that tells us the expected value of taking *a* in state *s*.
  - Note: $V^*(s) = \max_a Q^*(s, a)$.
- Optimal policy is formed as $\pi^*(s) = \text{arg}\max_a Q^*(s, a)$

The Q*(s,a) function

- Starting from state s, perform (perhaps suboptimal) action *a*. THEN follow the optimal policy

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

- Equivalent to

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

Q-Learning Iteration

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

- Iterative procedure

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

Learning rate

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

Note: We are using a policy to take actions; based on the estimated Q!

Q-Learning

Estimate Q*(s,a) from data $\{(s_t, a_t, r_t, s_{t+1})\}$:

1. Initialize Q(.,.) arbitrarily (eg all zeros)
    1. Except terminal states $Q(s_{terminal},.)=0$
2. Iterate over data until Q(.,.) converges:

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_b Q(s_{t+1}, b))$$

Learning rate

Q-learning Algorithm

Input: step size α, exploration probability ϵ

1. set Q(*s*,*a*) = 0 for all *s*, *a*.
2. For each episode:
3. Get initial state *s*.
4. While (*s* not a terminal state):
5. Perform a = ϵ-greedy(Q, *s*), receive *r*, *s*'
6. $Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$
7. $s \leftarrow s'$
8. End While
9. End For

Explore: take action to see what happens.

Update action-value based on result.

Q-Learning: SARSA

An alternative update rule:

- Just use the next action, no max over actions:

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

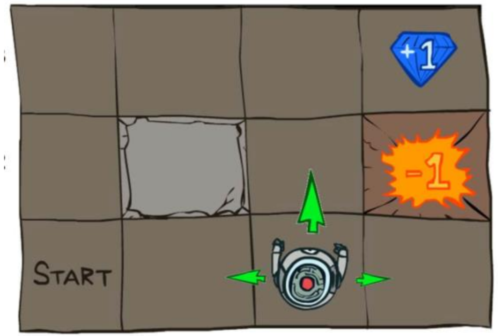
Learning rate

- Called state–action–reward–state–action (**SARSA**)
- Can use with epsilon-greedy policy

Q-Learning Details

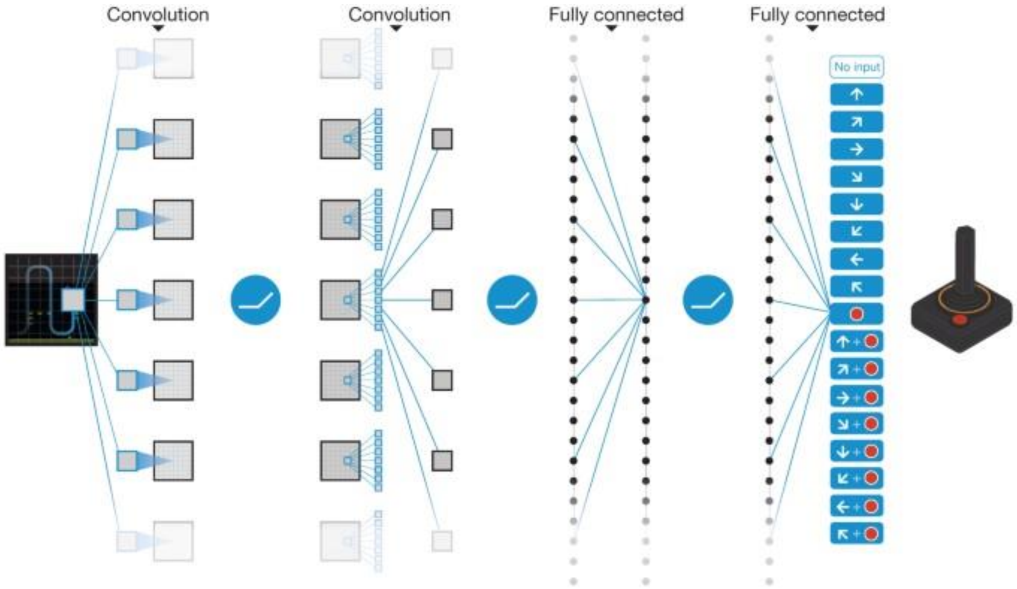
Note: if we have a **terminal** state, the process ends

- An **episode**: a sequence of states ending at a terminal state
- Want to run on many episodes
- Slightly different Q-update for terminal states



Deep Q-Learning

How do we get Q(*s*,*a*) with a large number of states?



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

Deep Q-Learning

How do we get Q(*s*,*a*) with a large number of states?

- Deep Q-learning uses a neural network to approximate Q(s,a)
  - Let $Q_\theta : S \times A \to \mathbb{R}$ be the neural network with weights and biases denoted θ.
- Training is similar to supervised regression:
  - (s, a) as input and $y = r(s) + \gamma \max_{a'} Q_\theta(s', a')$ as label.
  - Note that output of the neural network is used in the label.
  - Loss function: $\mathcal{L}(\theta) = (y - Q_\theta(s, a))^2$

Summary of RL

- Reinforcement learning setup
- Mathematical formulation: MDP
- Value functions & the Bellman equation
- Value iteration
- Q-learning