Deep Learning (1470)

Randall Balestriero

Class 19: Reinforcement Learning

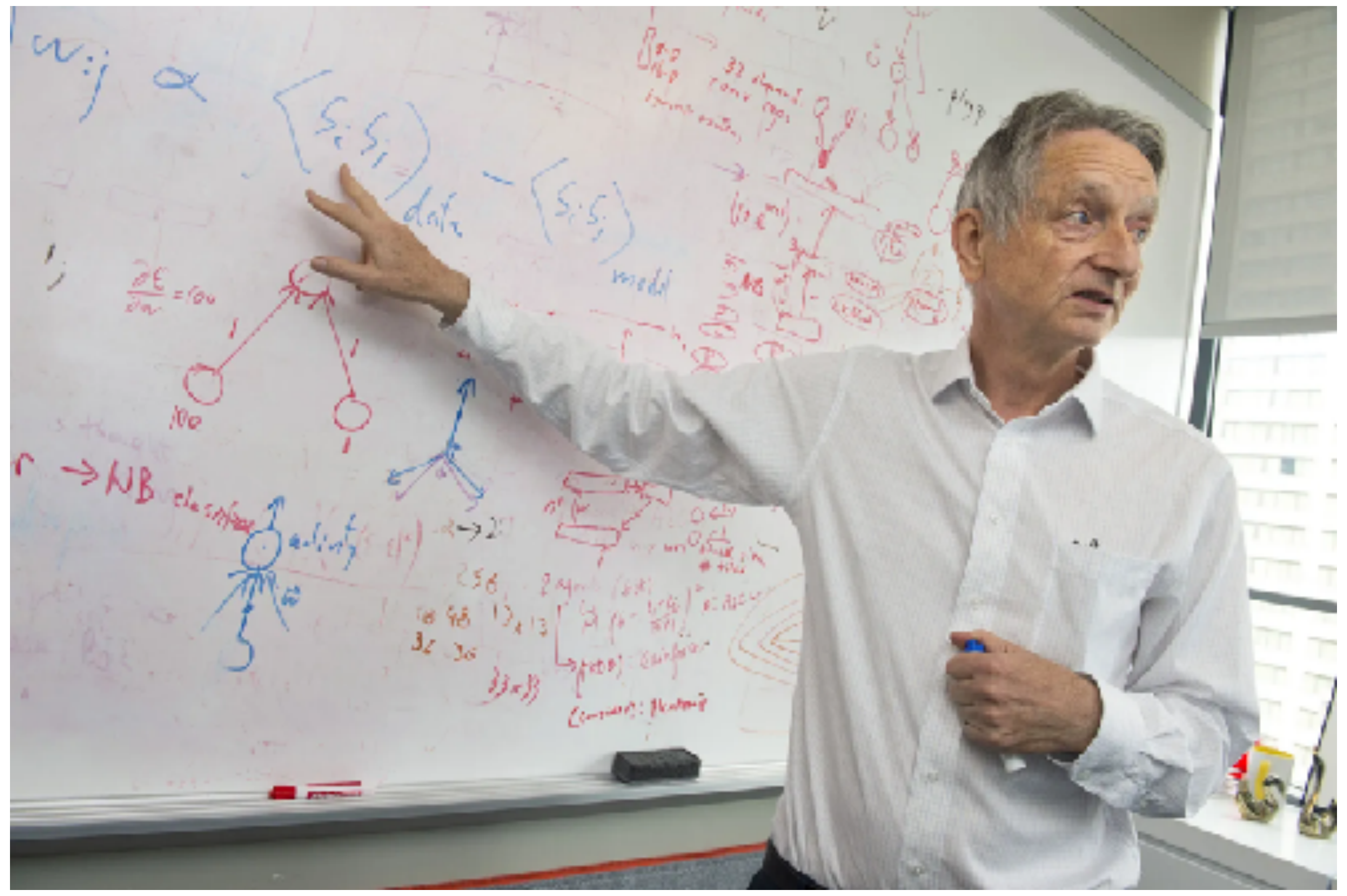
Recap!

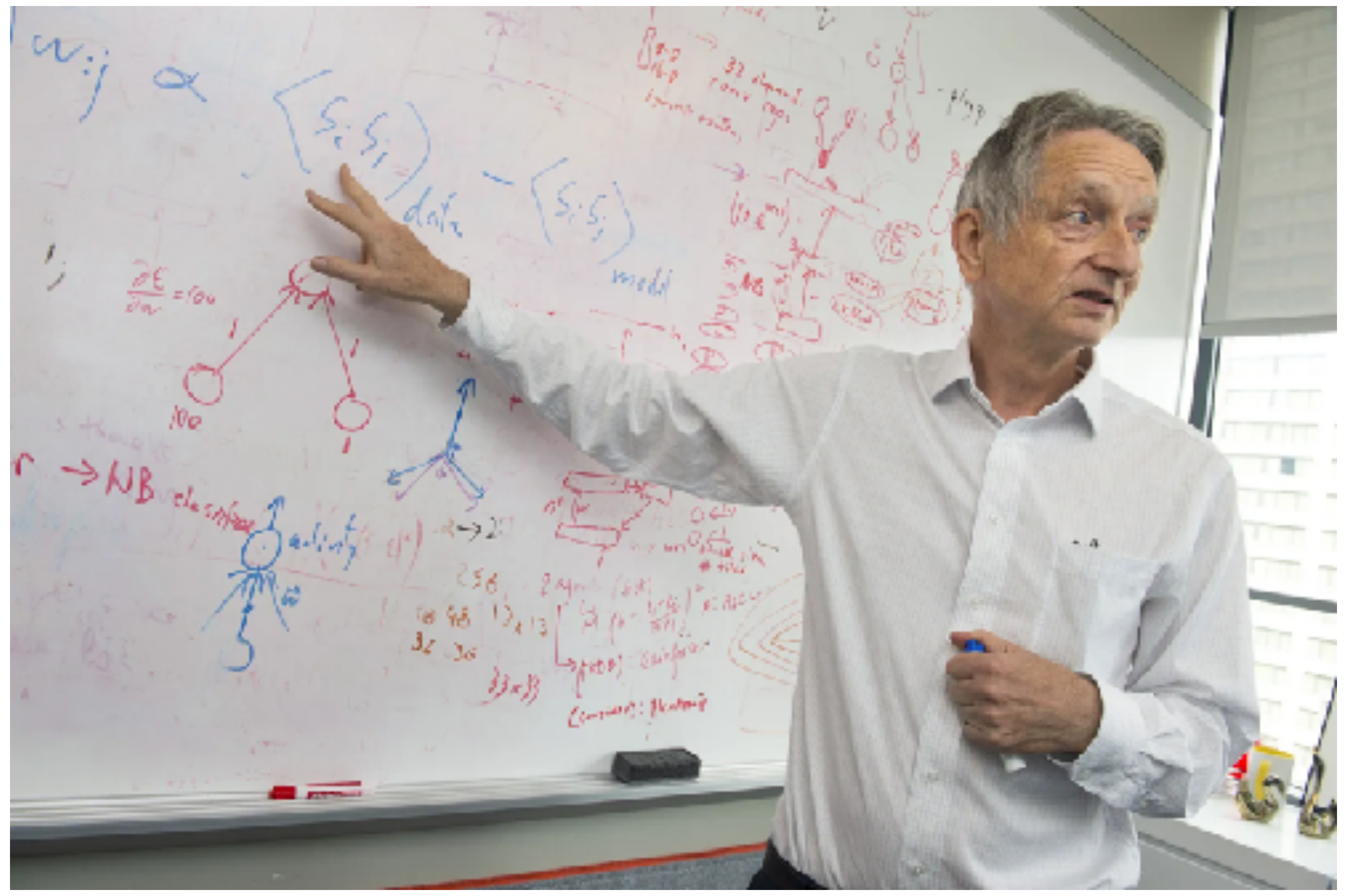
85
65
45
25
5
140
mph
206'
10
4
3
2
1
0
44
rpm

TOP
TIME
LAP 66"98
SCORE 9460
52
SPEED 180mph

85
65
45
25
5
140
mph
44
rpm

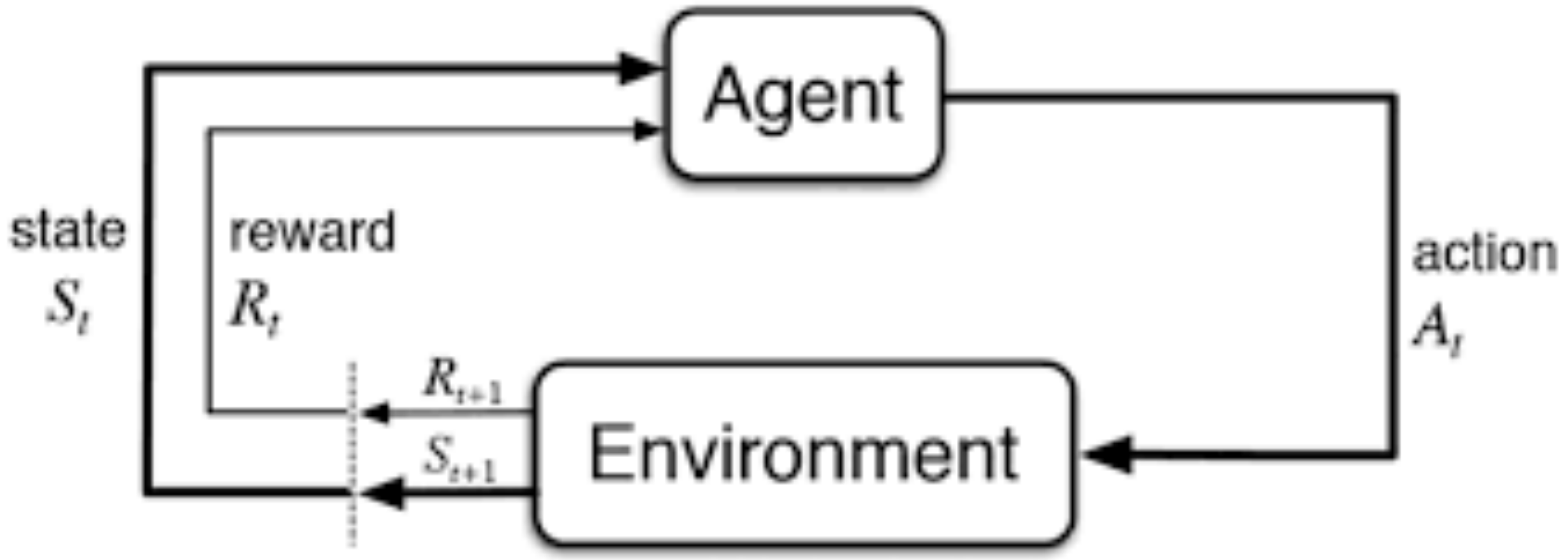
TOP 25850
TIME
LAP 66"96
SCORE 9460
52
SPEED 180mph





Action a_t
State s_{t+1}
Reward r_{t+1}
AGENT
ENVIRONMENT

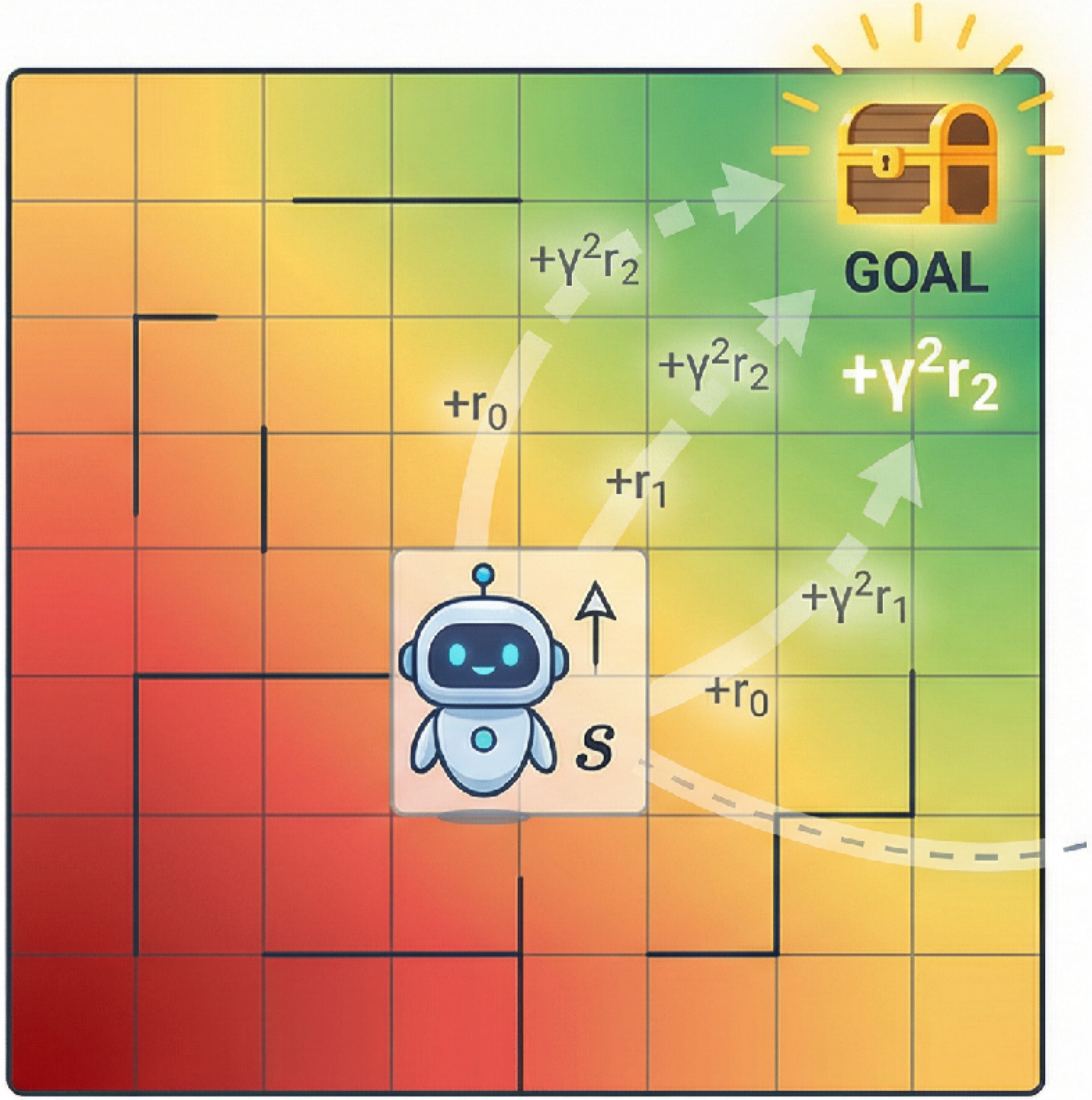
Markov Decision Processes (MDPs)



- Set of States: S
  - All possible configurations the world can be in
- Set of Actions: A
  - All possible actions the agent is able to take
- Reward Function: R: $S \to \mathbb{R}$
  - Reward function takes in a state and returns a number
- Transition Function: T: $S \times A \times S \to \mathbb{R}$
  - If you take an action in a specific state, what's the probability you transition to any other state?

How to train a Deep Network to play?


Behavior Cloning: Learning from Expert Demonstrations
Key Idea
• Learn a policy $\pi(a|s)$ by imitating expert demonstrations
• Supervised learning: treat actions as labels
$\pi_\theta(a|s) \approx \pi_{\text{expert}}(a|s)$
Step 1: Collect Expert Data
Expert π^*
$\tau = \{(s_1,a_1), (s_2,a_2), (s_3,a_3), \dots\}$
Dataset: $D = \{(s_i, a_i)\}_{i=1}^N$
Architecture
Input: State s
Policy π_θ
Output: Action a
Direct mapping: $s \rightarrow a$
Step 2: Supervised Learning
Policy Network π_θ
state s
predicted action $\hat{a}$
expert action a
gradient update
Loss Function
$\mathcal{L}(\theta) = \mathbb{E}[(a - \pi_\theta(s))^2]$
(for continuous)
Or:
$\mathcal{L}(\theta) = -\mathbb{E}[\log \pi_\theta(a|s)]$
(for discrete (cross-entropy))
Pros and Cons
Pros
✓ Simple supervised learning
✓ No reward function needed
✓ Fast to train
Cons
✗ Distribution shift / compounding errors
✗ Needs lots of expert data
✗ Can't exceed expert performance

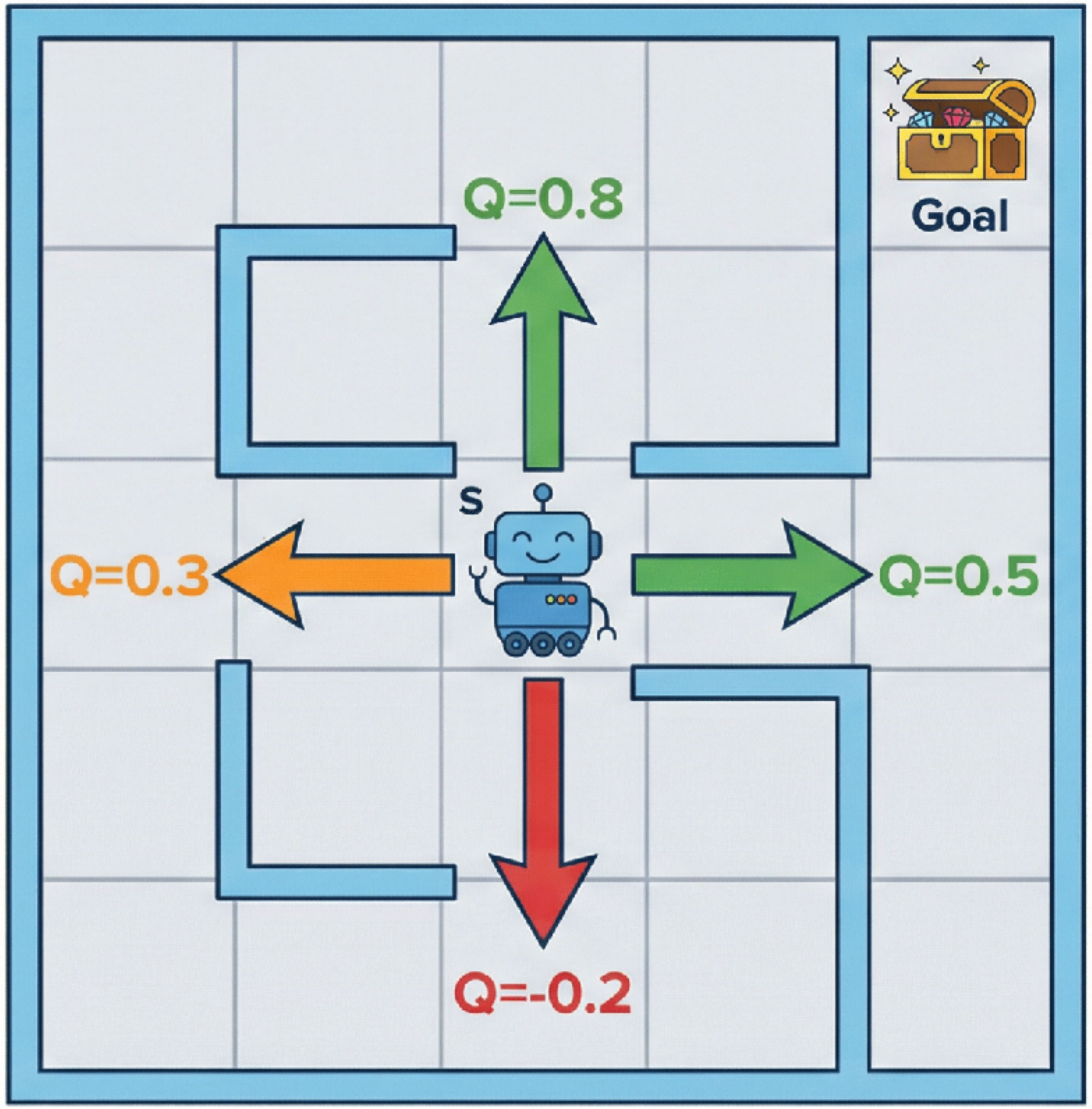


$$V(s) = \mathbb{E}\left[\sum_{t}^{t} \gamma^t r_t \mid s_0 = s\right]$$

$$V(s) = \mathbb{E}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \mid s_0 = s\right]$$

Expected cumulative discounted reward from state s

How good is it to BE in state s? 💡



$$Q(s,a) = E[\sum \gamma^t r_t \mid s_0=s,\ a_0=a]$$

$$Q(s,a) = r + \gamma V(s')$$

Expected reward from state s, taking action a

How good is it to TAKE action a in state s?


MODEL-FREE
Q1
Q2
Q2
V
Robot
Takes Action
Gets Reward
Q(s,a)
Policy weights
Updates Brain Directly
Learn V(s), Q(s,a), or π(a|s) directly
Learn WHAT to do
Q(s,a)
Policy weights
Simple but sample inefficient
Learning to ride a bike by just trying, falling, adjusting
MODEL-BASED
Robot
Has Internal "Mini World"/ Simulator
Builds Mental Model & Plans
Internal Planning (Simulation)
Learn P(s'|s,a) and R(s,a), then plan
Learn HOW the world works
Complex but sample efficient
Learning physics first, then mentally simulating before acting
Model-Free:
Trial and error
What should I do?
Model-Based:
Build mental model
What would happen if...? → What should I do?

Bellman Equations

State Value V(s)

Definition:

$$V^\pi(s) = \mathbb{E}_\pi[\sum_t \gamma^t r_t \mid s_0 = s]$$

Bellman Expectation Equation:

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

Bellman Optimality Equation:

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

Action Value Q(s,a)

Definition:

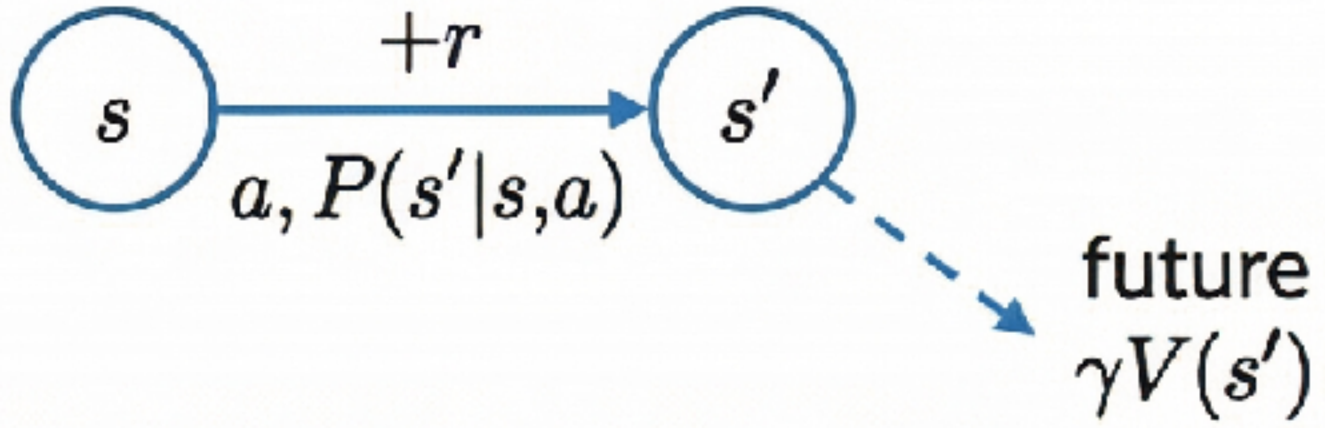
$$Q^\pi(s,a) = \mathbb{E}_\pi[\sum_t \gamma^t r_t \mid s_0 = s, a_0 = a]$$

Bellman Expectation Equation:

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

Bellman Optimality Equation:

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

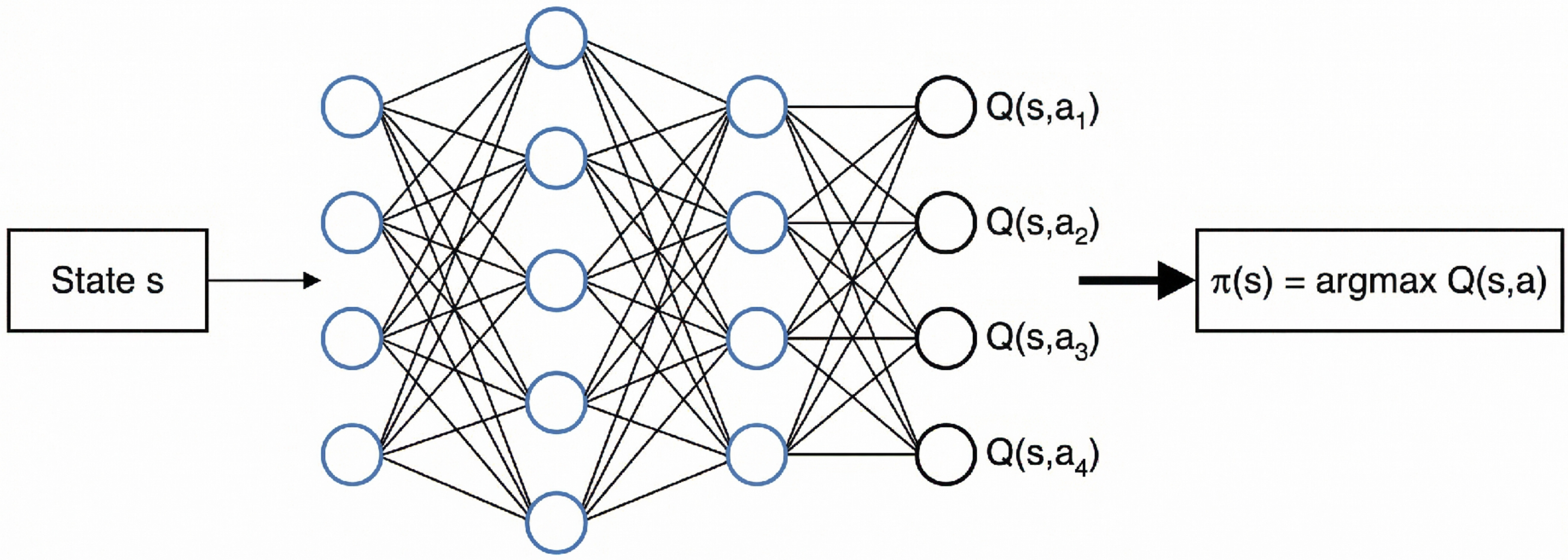


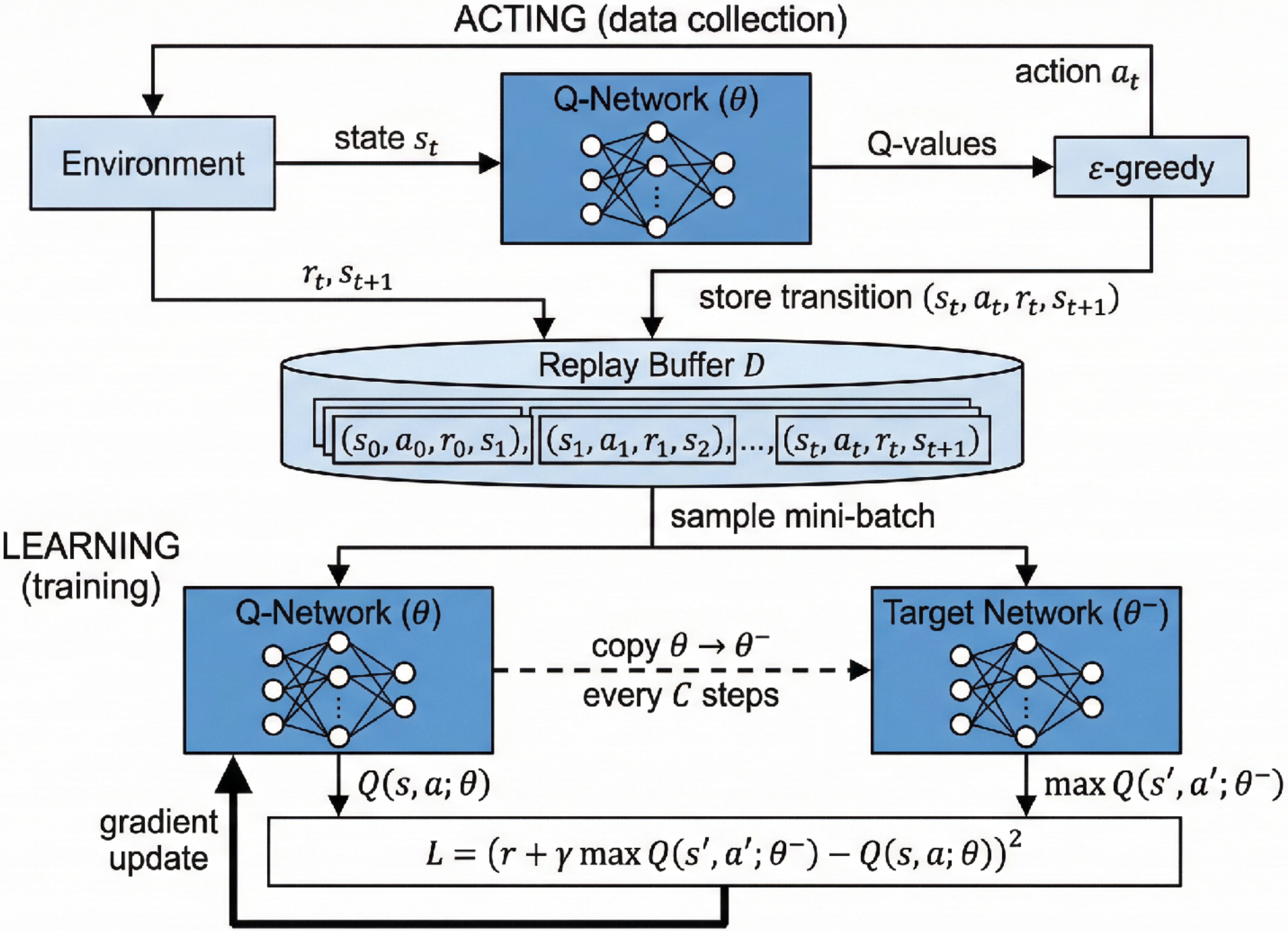
Value = Immediate Reward + Discounted Future

KEY NOTATION

- $\pi(a|s)$: policy (prob of action a in state s)
- $P(s'|s,a)$: transition probability
- $\gamma \in [0,1]$: discount factor
- r: immediate reward $R(s,a,s')$

Deep Q-Network





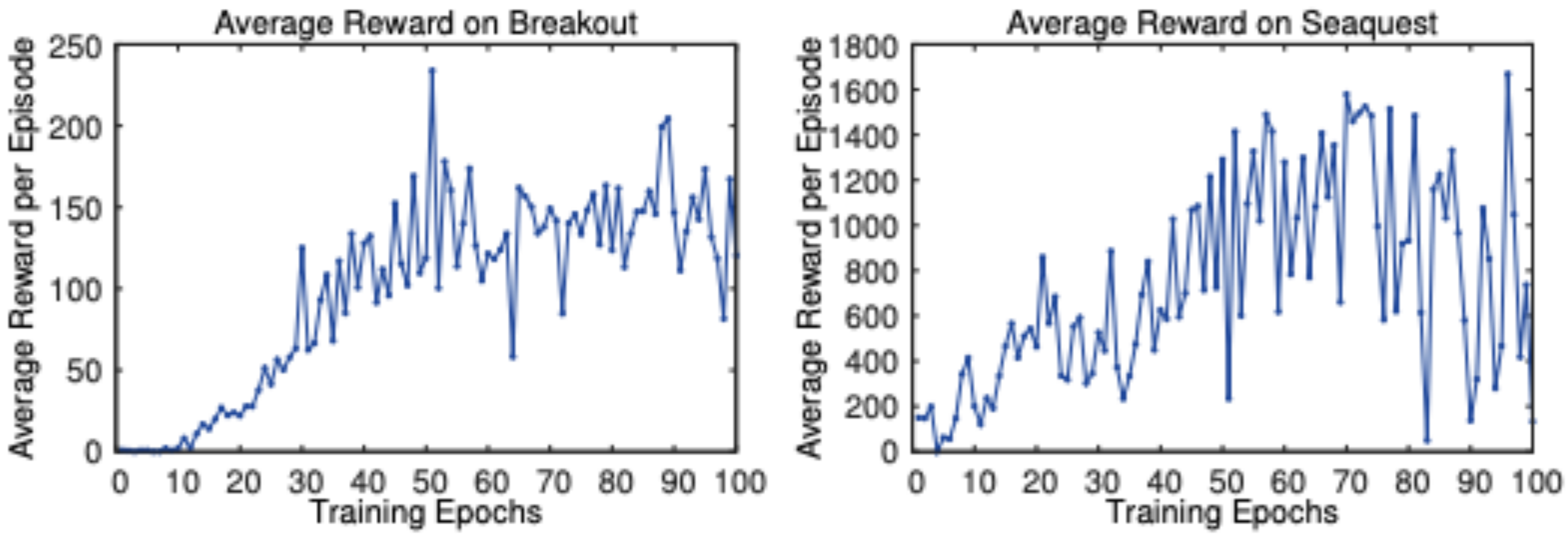
ACTING (data collection)
action a_t
Q-Network (θ)
Environment
state s_t
Q-values
ε-greedy
r_t, s_{t+1}
store transition (s_t, a_t, r_t, s_{t+1})
Replay Buffer D
(s_0, a_0, r_0, s_1), (s_1, a_1, r_1, s_2), ..., (s_t, a_t, r_t, s_{t+1})
sample mini-batch
LEARNING (training)
Q-Network (θ)
copy $\theta \rightarrow \theta^-$ every C steps
Target Network (θ^-)
$Q(s, a; \theta)$
$\max Q(s', a'; \theta^-)$
gradient update
$L = (r + \gamma \max Q(s', a'; \theta^-) - Q(s, a; \theta))^2$

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou

Daan Wierstra Martin Riedmiller

DeepMind Technologies




Value-Based Methods
Q-Network/Value Function
State (s)
Q(s,right)=0.8
Q(s,up)=0.5
Q(s,left)=0.3
argmax
pick max Q
Action
DQN
Q-Learning
Learn VALUE → Derive Policy
Implicit Policy (deterministic)
Actor-Critic Methods
Actor Network
State (s)
Action (continuous)
Actor π
gradient to improve actor
Critic Network
State (s)
Action
Q(s,a) value
Critic Q
DDPG
TD3
SAC
Learn BOTH Value and Policy
Policy Gradient Methods
Policy Network
State (s)
π(a|s)
π(left|s)=0.1
π(up|s)=0.2
π(right|s)=0.7
Sample
sample from distribution
Action
PPO
REINFORCE
A2C
Learn POLICY Directly
Explicit Policy (can be stochastic)

See you on Monday!