

DSC291: Machine Learning with Few Labels

Reinforcement Learning

Zhiting Hu

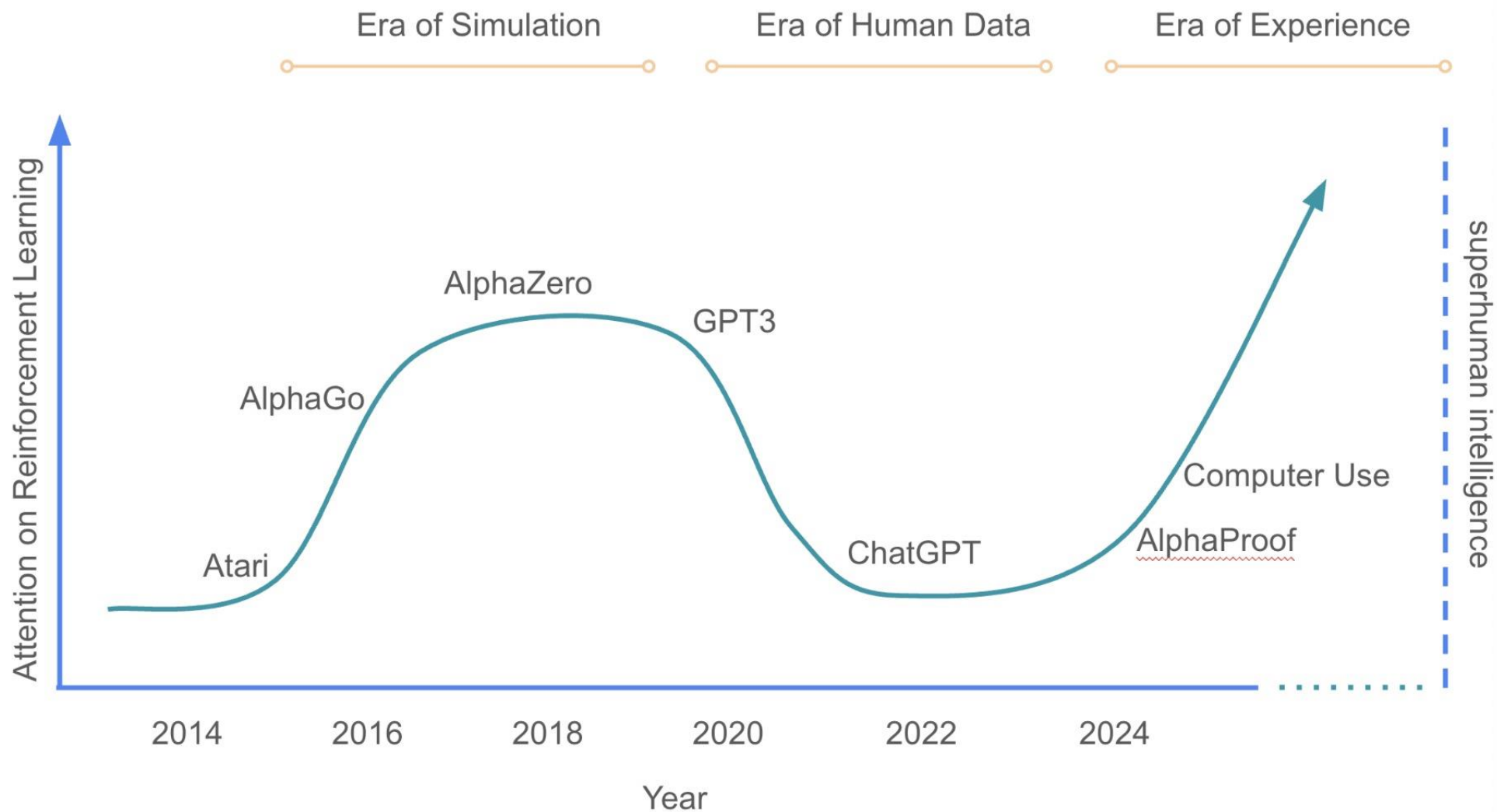
Lecture 13, May 13, 2025

Outline

- Reinforcement Learning
- Paper presentation:
 - David Lurie, Ben TenWolde: “Mixture of Agents with LLMs”
 - Qing Cheng, Zhaomei Geng: “ChatGPT in medicine: an overview of its applications, advantages, limitations, future prospects, and ethical considerations”

Welcome to the Era of Experience

David Silver, Richard S. Sutton*



So far... Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map $x \rightarrow y$

Examples: Classification,
regression, object detection,
semantic segmentation, image
captioning, etc.



→ Cat

Classification

So far... Unsupervised Learning

Data: x
no labels!

Goal: Learn some underlying
hidden *structure* of the data

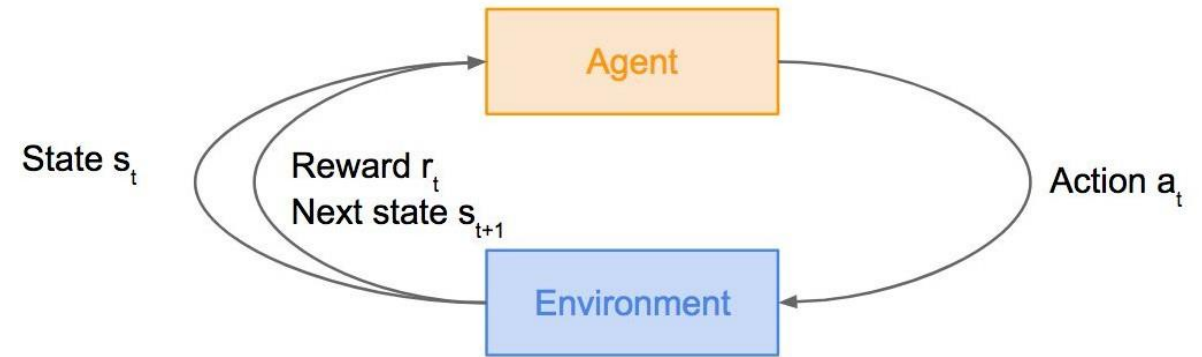
Examples: Clustering,
dimensionality reduction, feature
learning, density estimation, etc.



Today: Reinforcement Learning

Problems involving an **agent** interacting with an **environment**, which provides numeric **reward** signals

Goal: Learn how to take actions in order to maximize reward



Overview

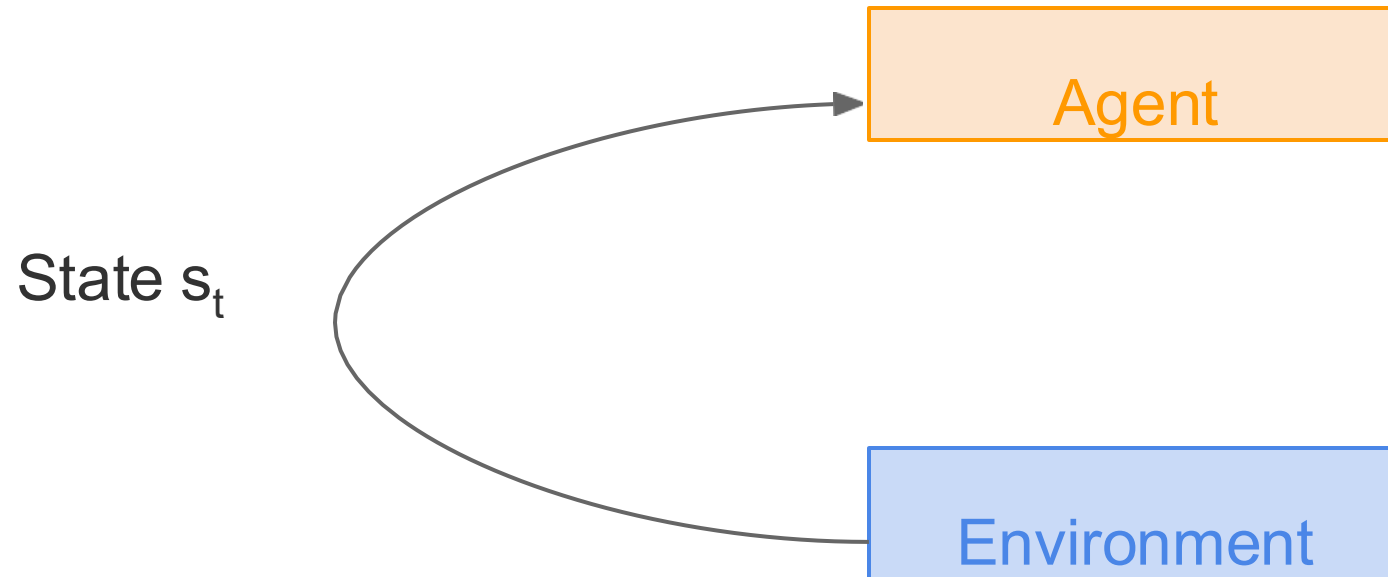
- What is Reinforcement Learning?
- Markov Decision Processes
- Q-Learning
- Policy Gradients

Reinforcement Learning

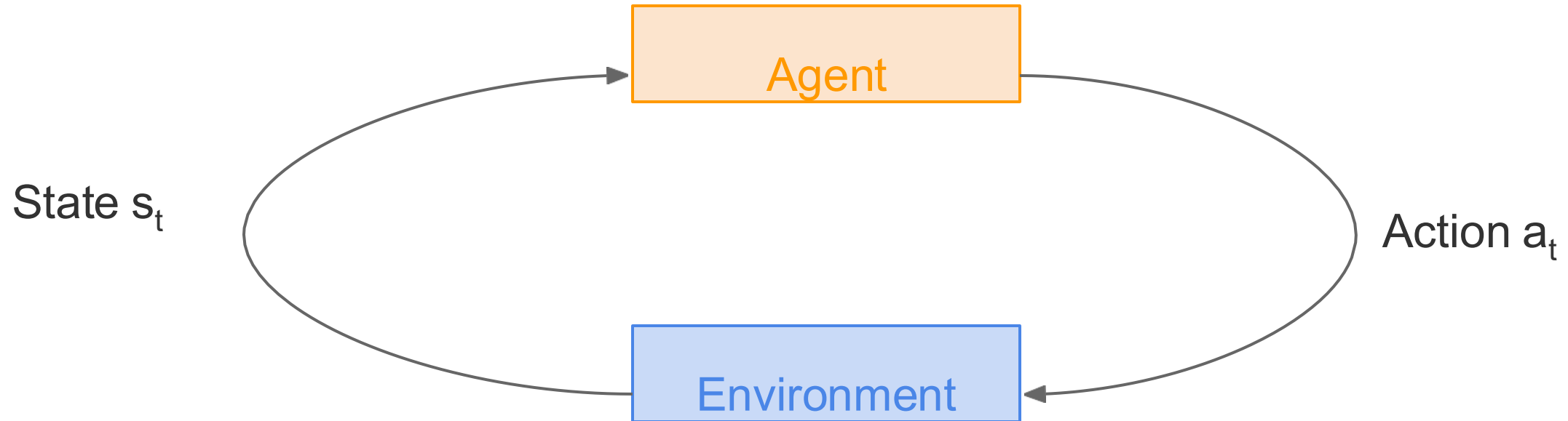
Agent

Environment

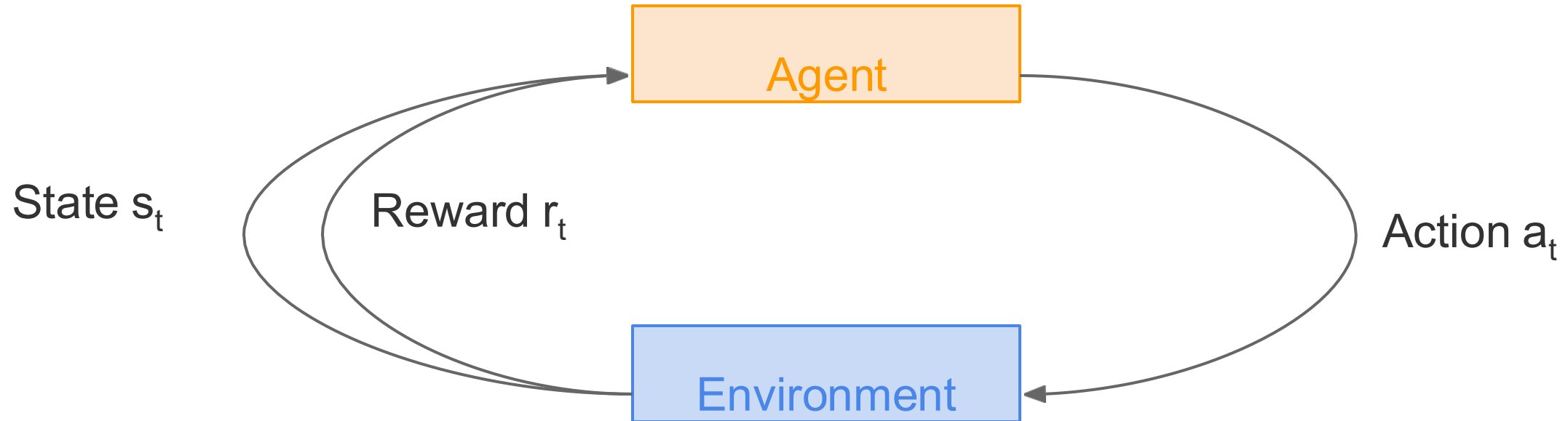
Reinforcement Learning



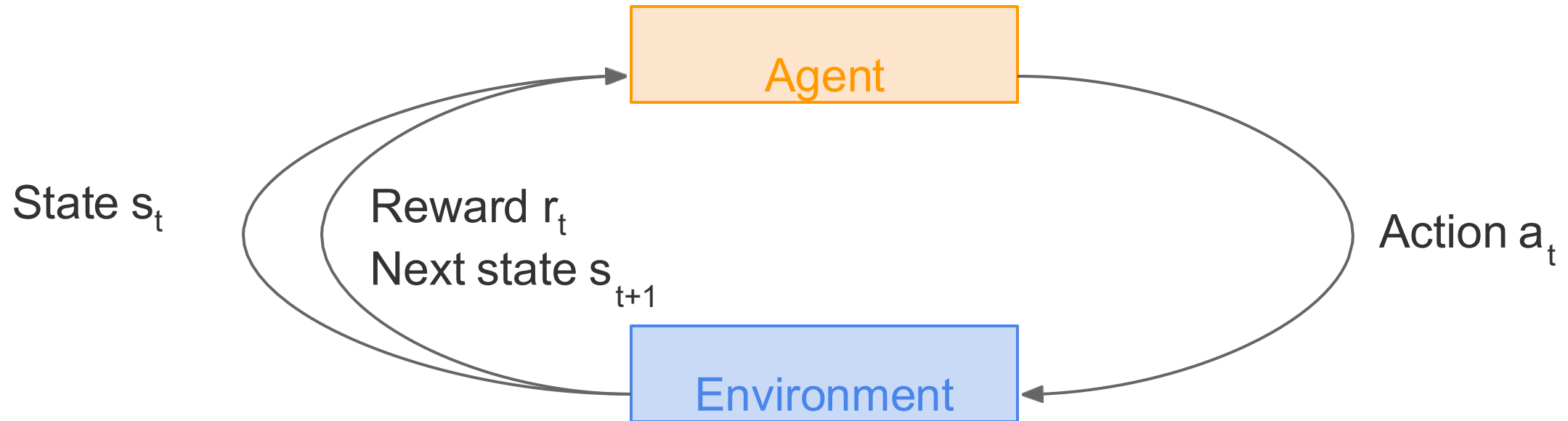
Reinforcement Learning



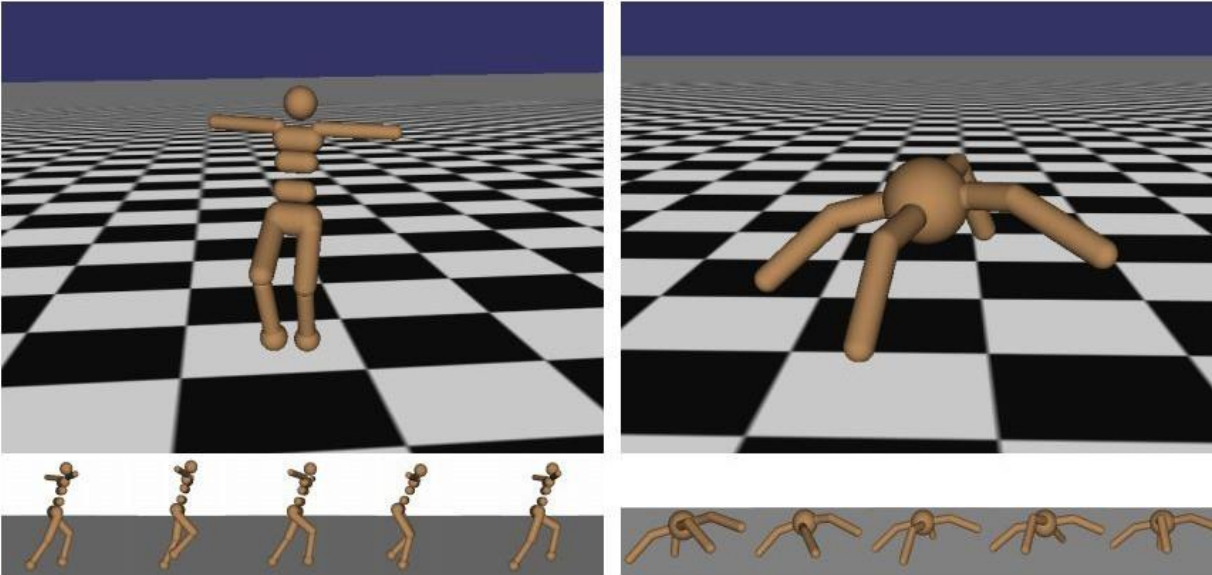
Reinforcement Learning



Reinforcement Learning



Robot Locomotion



Objective: Make the robot move forward

State: Angle and position of the joints

Action: Torque applied on joints

Reward: 1 at each time step upright + forward movement

Atari Games



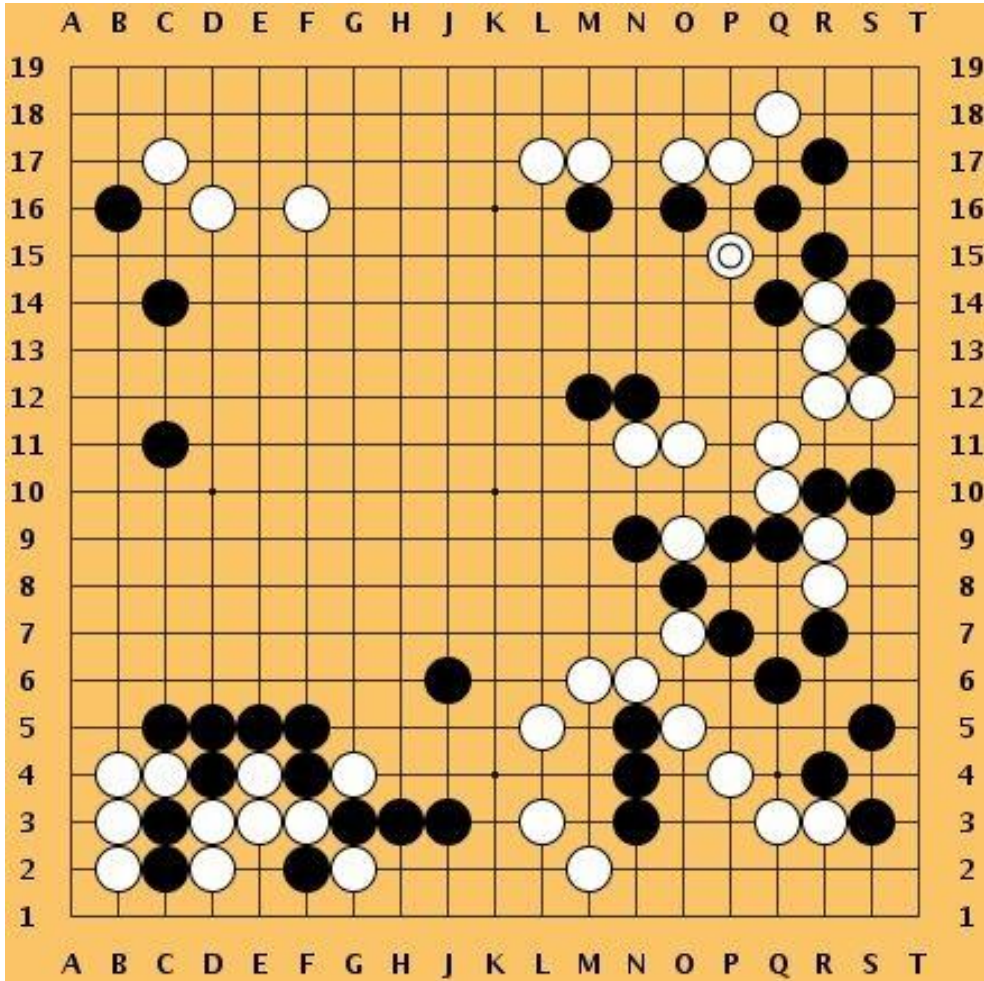
Objective: Complete the game with the highest score

State: Raw pixel inputs of the game state

Action: Game controls e.g. Left, Right, Up, Down

Reward: Score increase/decrease at each time step

Go



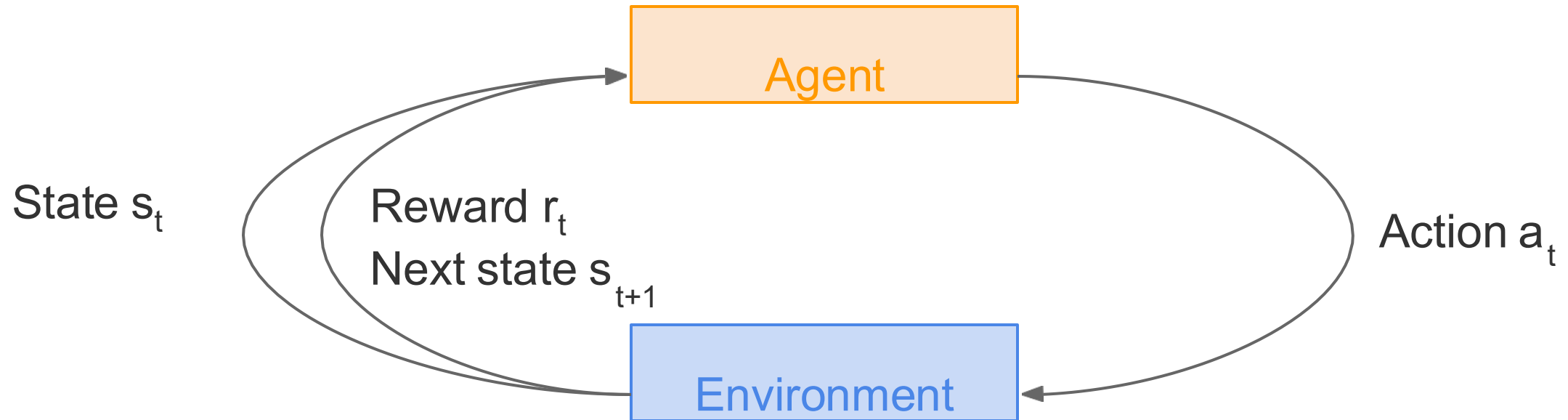
Objective: Win the game!

State: Position of all pieces

Action: Where to put the next piece down

Reward: 1 if win at the end of the game, 0 otherwise

How can we mathematically formalize the RL problem?



Markov Decision Process

- Mathematical formulation of the RL problem
- **Markov property**: Current state completely characterises the state of the world

Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

\mathcal{S} : set of possible states

\mathcal{A} : set of possible actions

\mathcal{R} : distribution of reward given (state, action) pair

\mathbb{P} : transition probability i.e. distribution over next state given (state, action) pair

γ : discount factor

Markov Decision Process

- At time step $t=0$, environment samples initial state $s_0 \sim p(s_0)$
- Then, for $t=0$ until done:
 - Agent selects action a_t
 - Environment samples reward $r_t \sim R(\cdot | s_t, a_t)$
 - Environment samples next state $s_{t+1} \sim P(\cdot | s_t, a_t)$
 - Agent receives reward r_t and next state s_{t+1}
- A policy π is a function from S to A that specifies what action to take in each state
- Following a policy produces sample *trajectories* (or paths) $s_0, a_0, r_0, s_1, a_1, r_1, \dots$
- **Objective:** find policy π^* that maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$

$$(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$$

A simple MDP: Grid World

actions = {

1. right 

2. left 

3. up 

4. down 

}

states

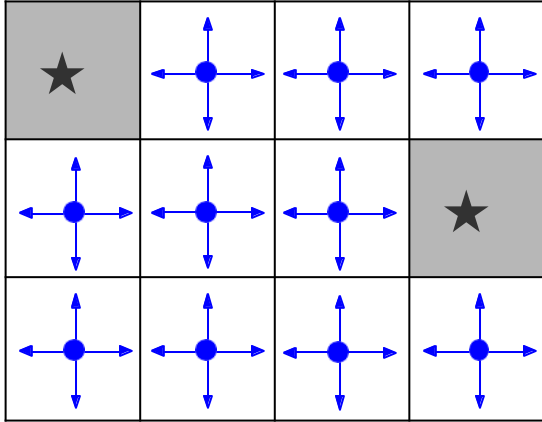
★			
			★

Set a negative “reward”
for each transition
(e.g. $r = -1$)

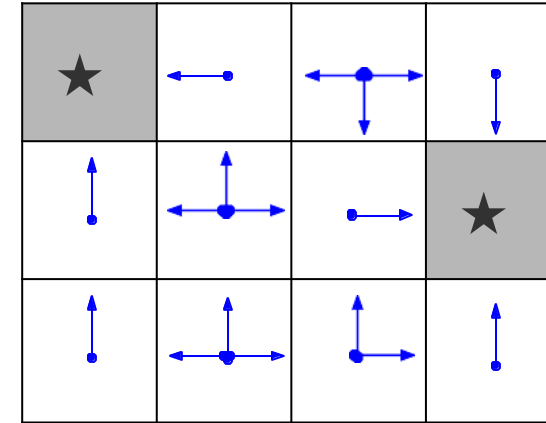
Objective: reach one of terminal states (greyed out) in
least number of actions

$$(S, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$$

A simple MDP: Grid World



Random Policy



Optimal Policy

$$(S, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$$

MDP for language generation

- (Autoregressive) text generation

Sentence $\mathbf{y} = (y_0, \dots, y_T)$

$$\pi_{\theta}(y_t \mid \mathbf{y}_{<t}) = \text{softmax}(f_{\theta}(y_t \mid \mathbf{y}_{<t}))$$

The optimal policy π^*

We want to find optimal policy π^* that maximizes the sum of rewards.

How do we handle the randomness (initial state, transition probability...)?

The optimal policy π^*

We want to find optimal policy π^* that maximizes the sum of rewards.

How do we handle the randomness (initial state, transition probability...)?

Maximize the **expected sum of rewards!**

$$\text{Formally: } \pi^* = \arg \max_{\pi} \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t | \pi \right] \quad \text{with } s_0 \sim p(s_0), a_t \sim \pi(\cdot | s_t), s_{t+1} \sim p(\cdot | s_t, a_t)$$

Definitions: Value function and Q-value function

Following a policy produces sample trajectories (or paths) $s_0, a_0, r_0, s_1, a_1, r_1, \dots$

Definitions: Value function and Q-value function

Following a policy produces sample trajectories (or paths) $s_0, a_0, r_0, s_1, a_1, r_1, \dots$

How good is a state?

The **value function** at state s , is the expected cumulative reward from following the policy from state s :

$$V^\pi(s) = \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, \pi \right]$$

Definitions: Value function and Q-value function

Following a policy produces sample trajectories (or paths) $s_0, a_0, r_0, s_1, a_1, r_1, \dots$

How good is a state?

The **value function** at state s , is the expected cumulative reward from following the policy from state s :

$$V^\pi(s) = \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, \pi \right]$$

How good is a state-action pair?

The **Q-value function** at state s and action a , is the expected cumulative reward from taking action a in state s and then following the policy:

$$Q^\pi(s, a) = \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi \right]$$

Bellman equation

The optimal Q-value function Q^* is the maximum expected cumulative reward achievable from a given (state, action) pair:

$$Q^*(s, a) = \max_{\pi} \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi \right]$$

Bellman equation

The optimal Q-value function Q^* is the maximum expected cumulative reward achievable from a given (state, action) pair:

$$Q^*(s, a) = \max_{\pi} \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi \right]$$

Q^* satisfies the following **Bellman equation**:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right]$$

Intuition: if the optimal state-action values for the next time-step $Q^*(s', a')$ are known, then the optimal strategy is to take the action that maximizes the expected value of $r + \gamma Q^*(s', a')$

Bellman equation

The optimal Q-value function Q^* is the maximum expected cumulative reward achievable from a given (state, action) pair:

$$Q^*(s, a) = \max_{\pi} \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi \right]$$

Q^* satisfies the following **Bellman equation**:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right]$$

Intuition: if the optimal state-action values for the next time-step $Q^*(s', a')$ are known, then the optimal strategy is to take the action that maximizes the expected value of $r + \gamma Q^*(s', a')$

The optimal policy π^* corresponds to taking the best action in any state as specified by Q^*

Solving for the optimal policy

Value iteration algorithm: Use Bellman equation as an iterative update

$$Q_{i+1}(s, a) = \mathbb{E} \left[r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$

Q_i will converge to Q^* as $i \rightarrow \text{infinity}$

Solving for the optimal policy

Value iteration algorithm: Use Bellman equation as an iterative update

$$Q_{i+1}(s, a) = \mathbb{E} \left[r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$

Q_i will converge to Q^* as $i \rightarrow \text{infinity}$

Question: What's the problem with this?

Solving for the optimal policy

Value iteration algorithm: Use Bellman equation as an iterative update

$$Q_{i+1}(s, a) = \mathbb{E} \left[r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$

Q_i will converge to Q^* as $i \rightarrow \text{infinity}$

Question: What's the problem with this?

Not scalable. Must compute $Q(s, a)$ for every state-action pair. If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!

Solving for the optimal policy

Value iteration algorithm: Use Bellman equation as an iterative update

$$Q_{i+1}(s, a) = \mathbb{E} \left[r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$

Q_i will converge to Q^* as $i \rightarrow \text{infinity}$

Question: What's the problem with this?

Not scalable. Must compute $Q(s, a)$ for every state-action pair. If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!

Question: how would you solve the issue?

Solving for the optimal policy

Value iteration algorithm: Use Bellman equation as an iterative update

$$Q_{i+1}(s, a) = \mathbb{E} \left[r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$

Q_i will converge to Q^* as $i \rightarrow \text{infinity}$

Question: What's the problem with this?

Not scalable. Must compute $Q(s, a)$ for every state-action pair. If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!

Question: how would you solve the issue?

Solution: use a function approximator to estimate $Q(s, a)$. E.g. a neural network!

Solving for the optimal policy: Q-learning

Q-learning: Use a function approximator to estimate the action-value function

$$Q(s, a; \theta) \approx Q^*(s, a)$$

Solving for the optimal policy: Q-learning

Q-learning: Use a function approximator to estimate the action-value function

$$Q(s, a; \theta) \approx Q^*(s, a)$$

If the function approximator is a deep neural network => **deep q-learning!**

Solving for the optimal policy: Q-learning

Q-learning: Use a function approximator to estimate the action-value function

$$Q(s, a; \theta) \approx Q^*(s, a)$$

function parameters (weights)

If the function approximator is a deep neural network => **deep q-learning!**

Solving for the optimal policy: Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

Solving for the optimal policy: Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

Forward Pass

Loss function: $L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot)} [(y_i - Q(s, a; \theta_i))^2]$

where $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$

Solving for the optimal policy: Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

Forward Pass

Loss function: $L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot)} [(y_i - Q(s, a; \theta_i))^2]$

where $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$

Backward Pass

Gradient update (with respect to Q-function parameters θ):

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) \right] \nabla_{\theta_i} Q(s, a; \theta_i)$$

Case Study: Playing Atari Games



Objective: Complete the game with the highest score

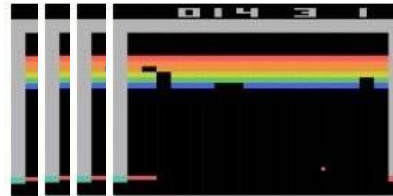
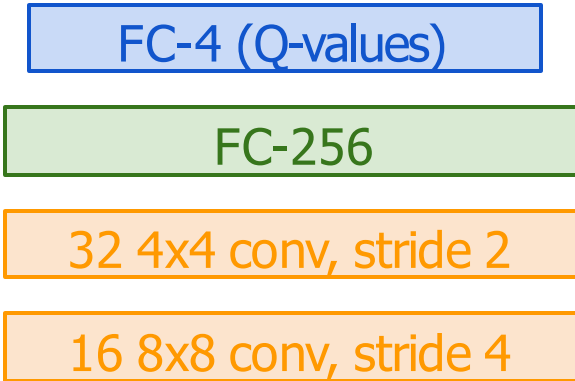
State: Raw pixel inputs of the game state

Action: Game controls e.g. Left, Right, Up, Down

Reward: Score increase/decrease at each time step

Q-network Architecture

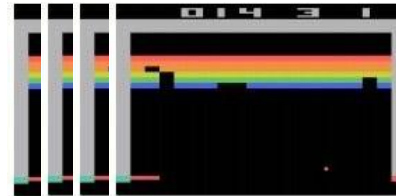
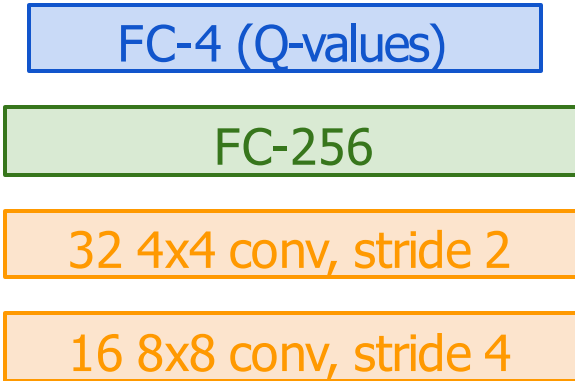
$Q(s, a; \theta)$:
neural network
with weights θ



Current state s_t : 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

Q-network Architecture

$Q(s, a; \theta)$:
neural network
with weights θ

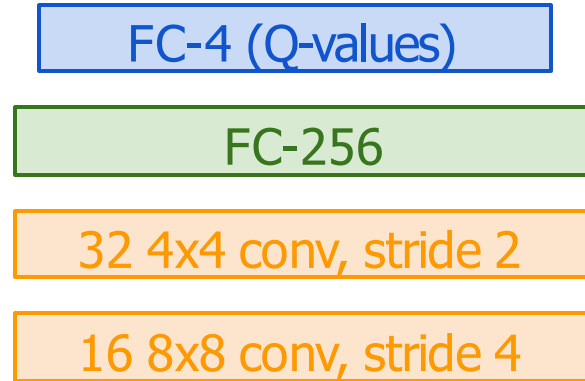


← Input: state s_t

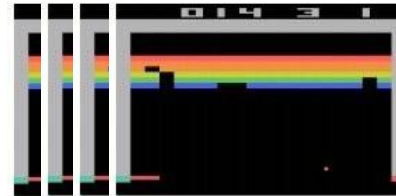
Current state s_t : 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

Q-network Architecture

$Q(s, a; \theta)$:
neural network
with weights θ



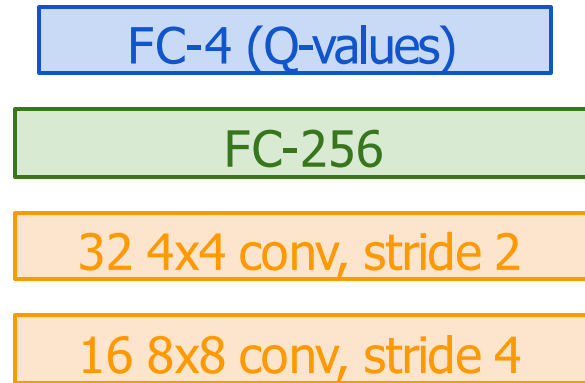
← Familiar conv layers,
FC layer



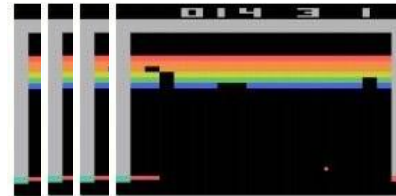
Current state s_t : 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

Q-network Architecture

$Q(s, a; \theta)$:
neural network
with weights θ



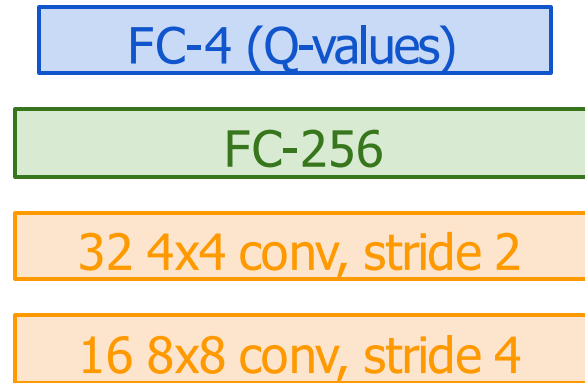
← Last FC layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t, a_4)$



Current state s_t : 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

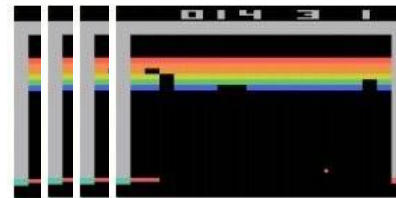
Q-network Architecture

$Q(s, a; \theta)$:
neural network
with weights θ



Last FC layer has 4-d
output (if 4 actions),
corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$,
 $Q(s_t, a_4)$

Number of actions between 4-18
depending on Atari game

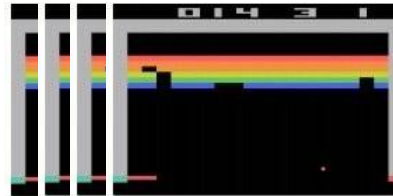
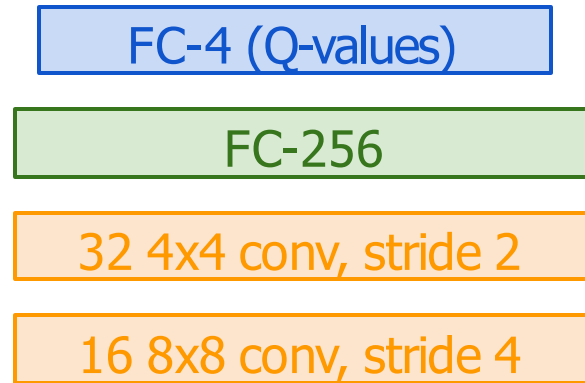


Current state s_t : 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

Q-network Architecture

$Q(s, a; \theta)$:
neural network
with weights θ

A single feedforward pass
to compute Q-values for all
actions from the current
state => efficient!



Current state s_t : 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

← Last FC layer has 4-d
output (if 4 actions),
corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$,
 $Q(s_t, a_4)$

Number of actions between 4-18
depending on Atari game

Summary so far

- Q-learning:
 - Bellman equation
 - Value-based RL
 - Off-policy RL

Loss function: $L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} [(y_i - Q(s, a; \theta_i))^2]$

where $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$

- Next: Policy gradient
 - Policy-based RL
 - On-policy RL

Questions?