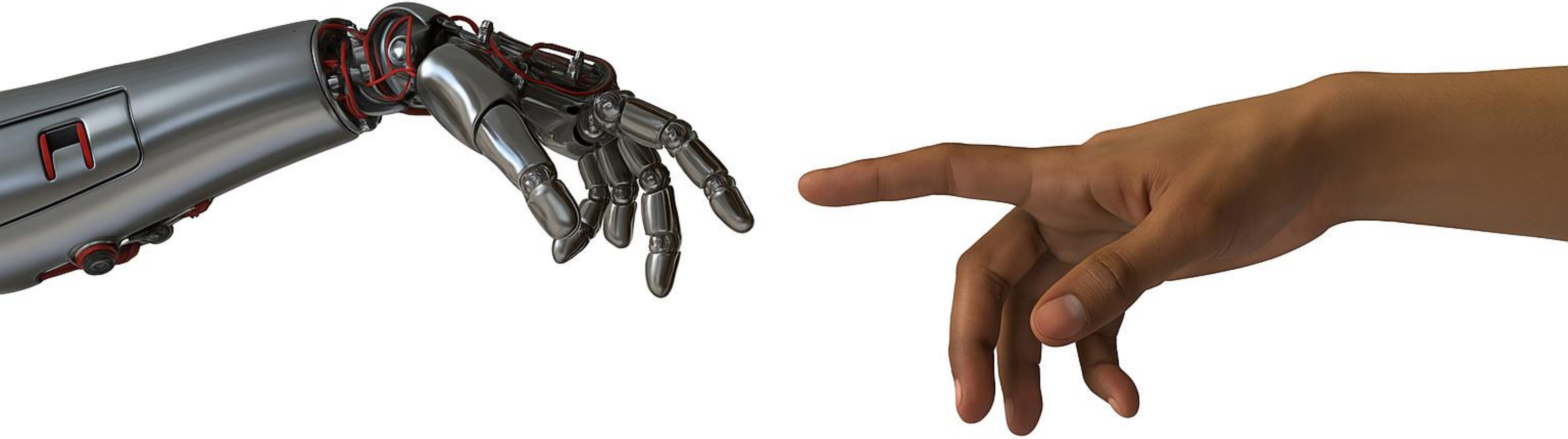
Reinforcement Learning

Richard Brooks

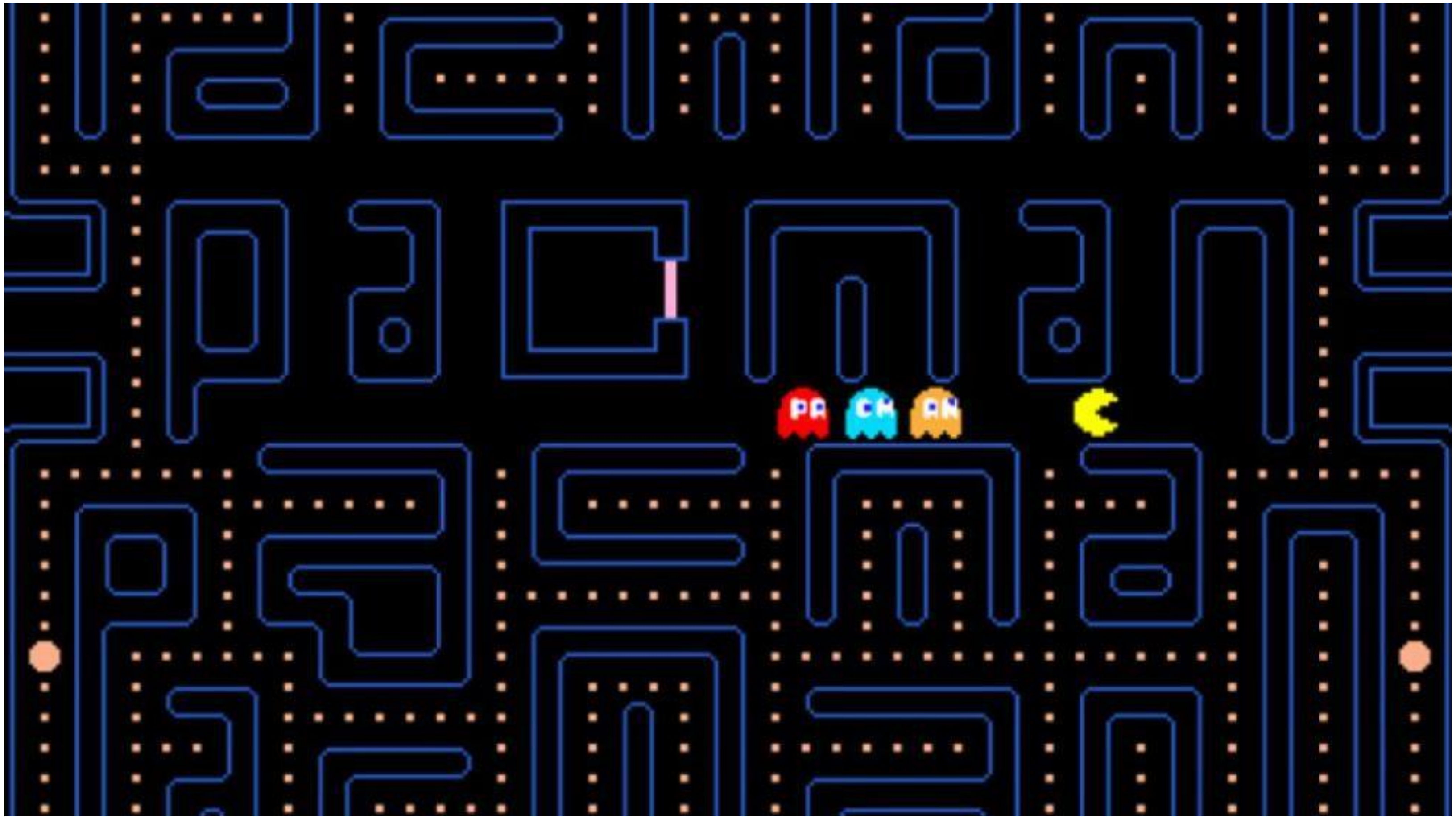
MAL2 – Spring 2025

Credits: [MIT 6.S191 Spring 2025]

Learning in Dynamic Environments



What is it?



An **agent**
makes **observations**
and takes **actions**
within an **environment**
and in return receives **rewards**

What is it?

agent ∴ observations ∴ actions ∴ environment ∴ rewards

Classes of Learning Problems

Supervised Learning

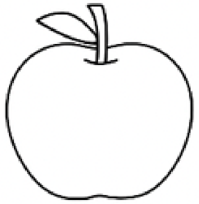
Data: (x, y)

$\boldsymbol{x}$ is data, $\boldsymbol{y}$ is label

Goal: Learn function to map

$$x \rightarrow y$$

Apple example:



This thing is an apple.

Classes of Learning Problems

Supervised Learning

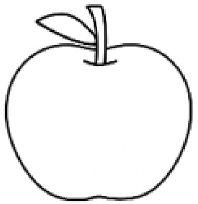
Data: (x, y)

$\boldsymbol{x}$ is data, $\boldsymbol{y}$ is label

Goal: Learn function to map

$$x \to y$$

Apple example:



This thing is an apple.

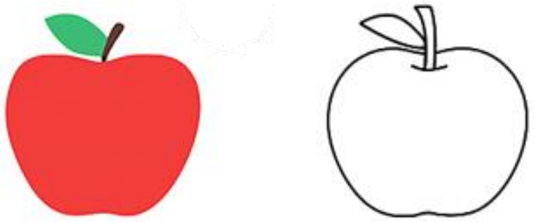
Unsupervised Learning

Data: x

$\boldsymbol{x}$ is data, no labels!

Goal: Learn underlying structure

Apple example:



This thing is like the other thing

Classes of Learning Problems

Supervised Learning

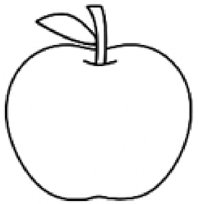
Data: (x, y)

$\boldsymbol{x}$ is data, $\boldsymbol{y}$ is label

Goal: Learn function to map

$$x \rightarrow y$$

Apple example:



This thing is an apple.

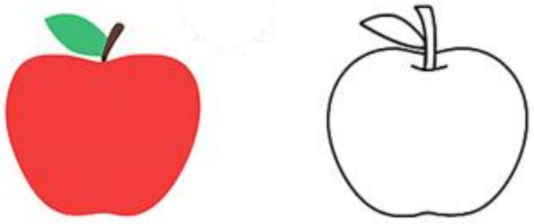
Unsupervised Learning

Data: x

$\boldsymbol{x}$ is data, no labels!

Goal: Learn underlying structure

Apple example:



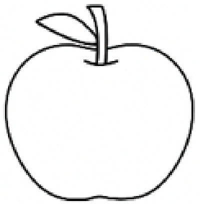
This thing is like the other thing

Reinforcement Learning

Data: State-action pairs

Goal: Maximize future rewards over many time steps

Apple example:



Eat this thing because it keeps the doctor away.

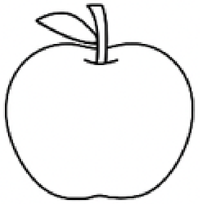
RL: Our focus today

Reinforcement Learning

Data: State-action pairs

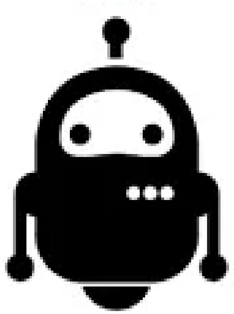
Goal: Maximize future rewards over many time steps

Apple example:



Eat this thing because it keeps the doctor away.

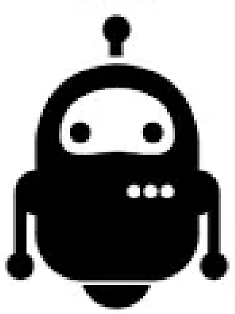
Key Concepts



AGENT

Agent: takes actions

Key Concepts

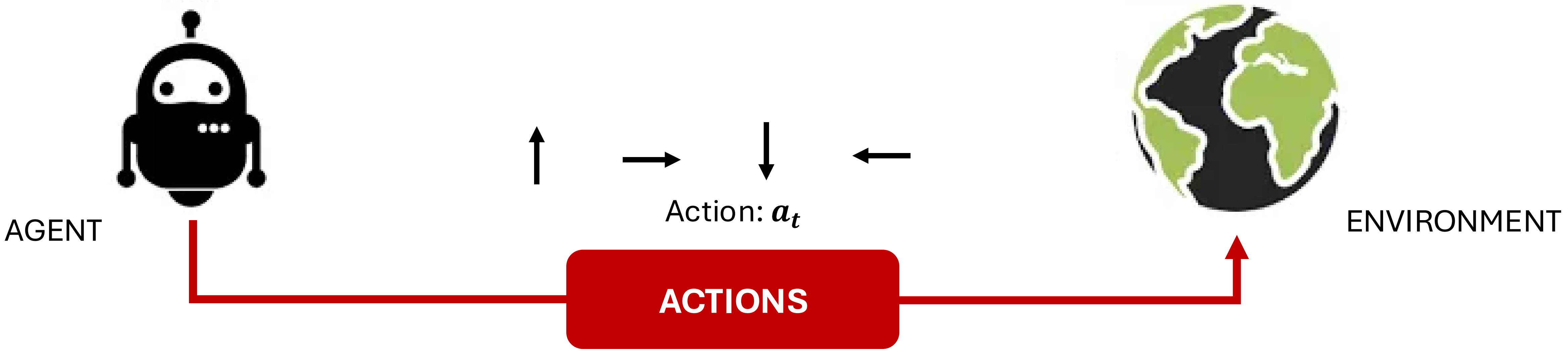


AGENT

ENVIRONMENT

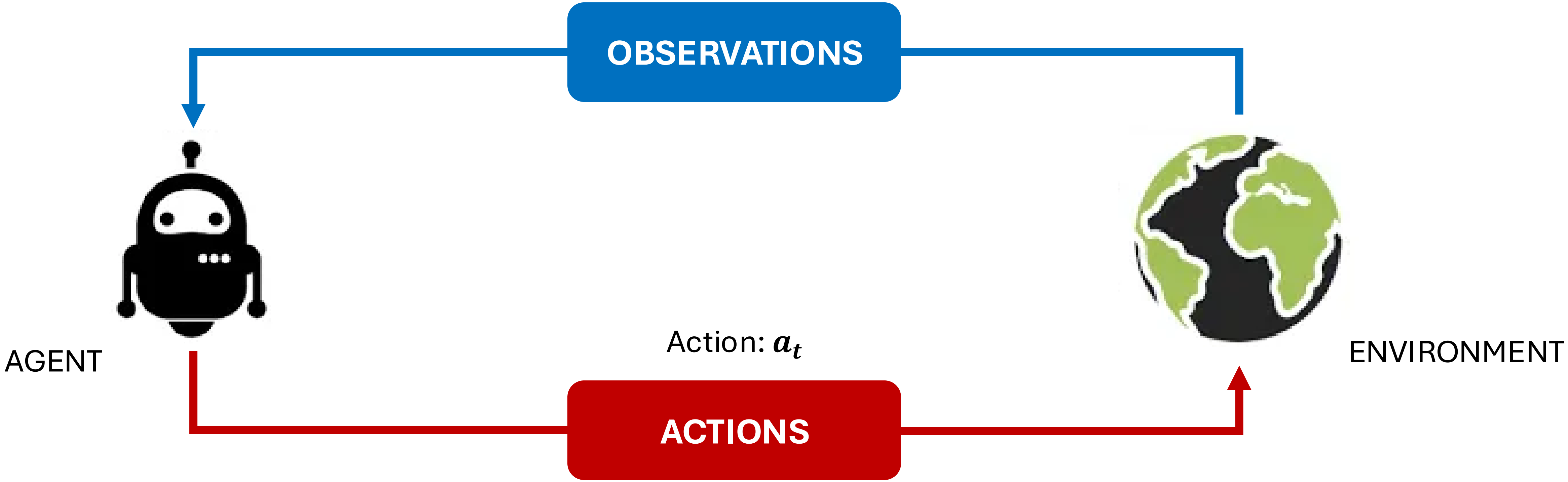
Environment: the world in which the agent exists and operate

Key Concepts



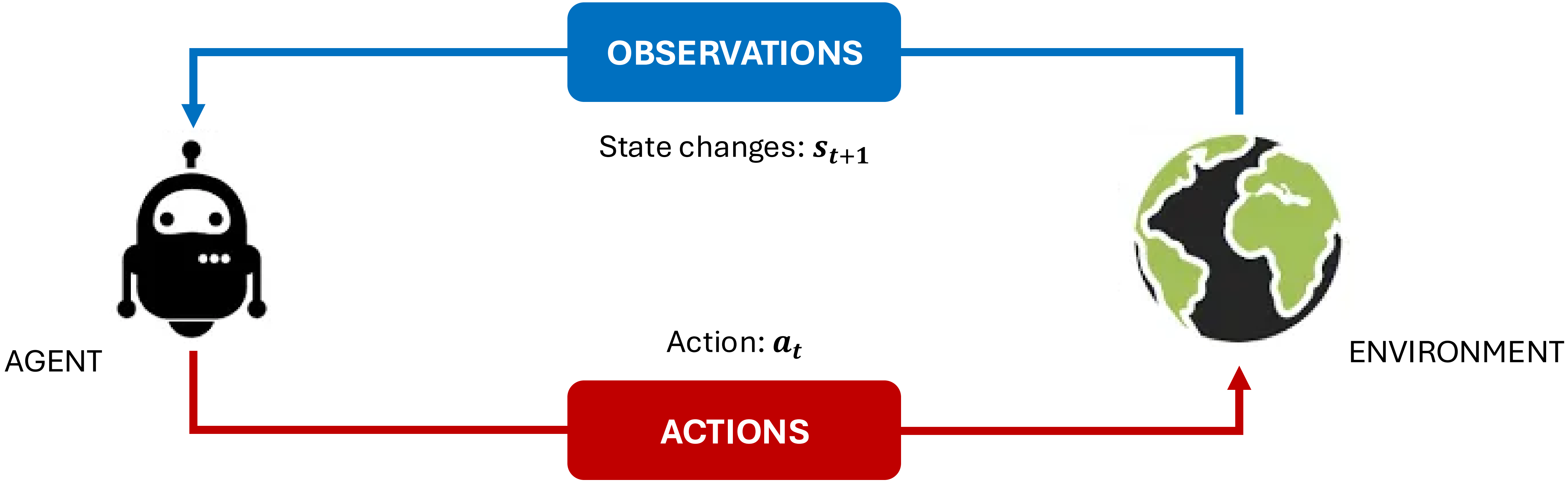
Action: a move the agent can make in the environment.
Action space A : the set of possible actions an agent can make in the environment

Key Concepts



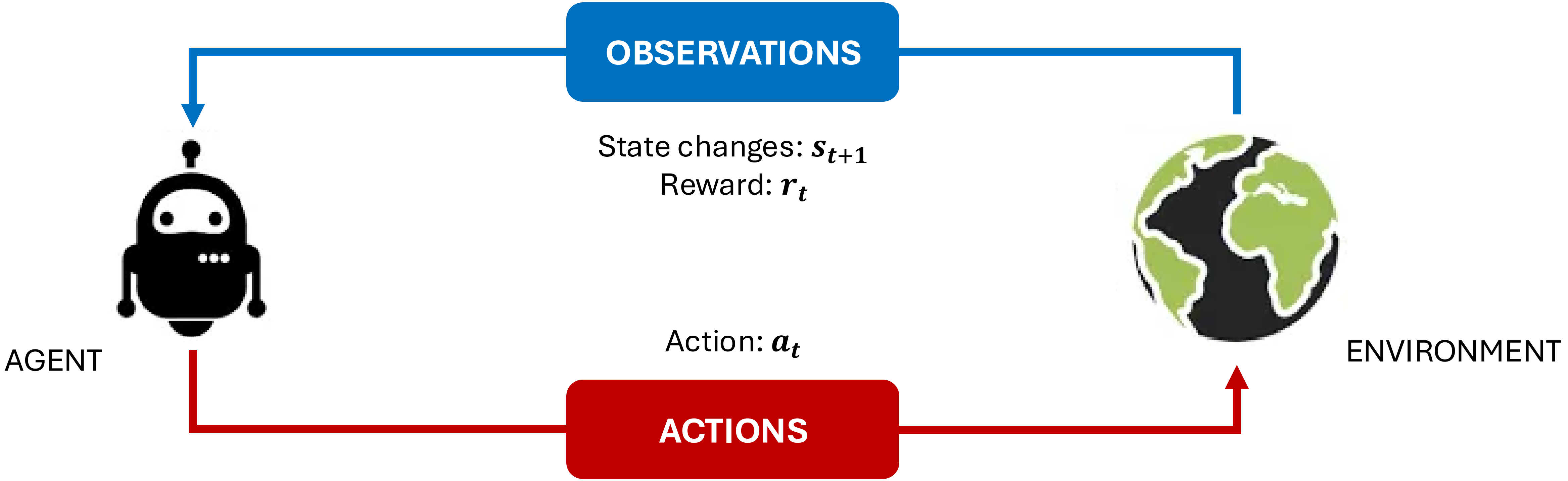
Observations: of the environment after taking actions.

Key Concepts



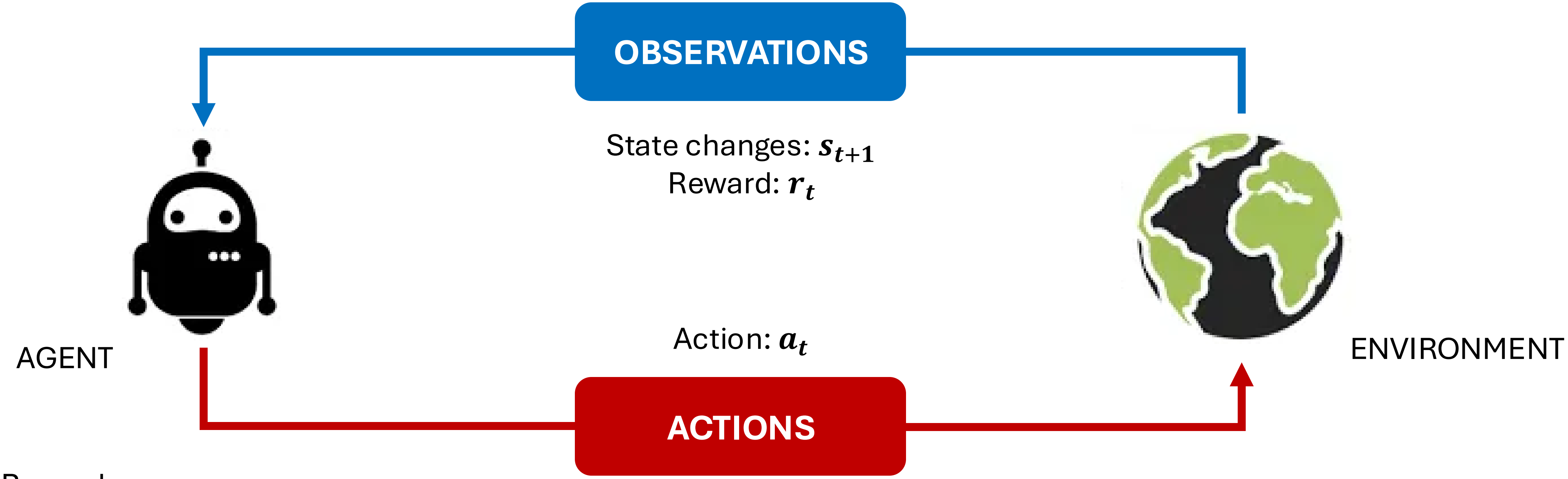
State: a situation which the agent perceives.

Key Concepts



Reward: feedback that measures the success or failure of the agent's action.

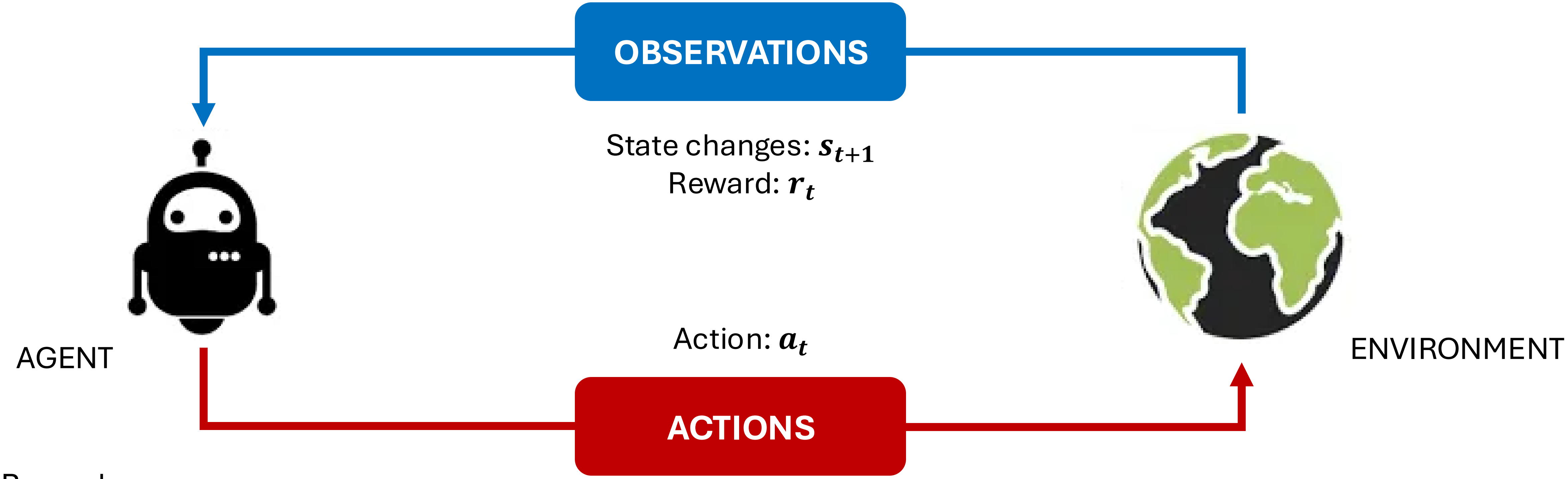
Key Concepts



Total Reward
(Return)

$$R_t = \sum_{i=t}^{\infty} r_i$$

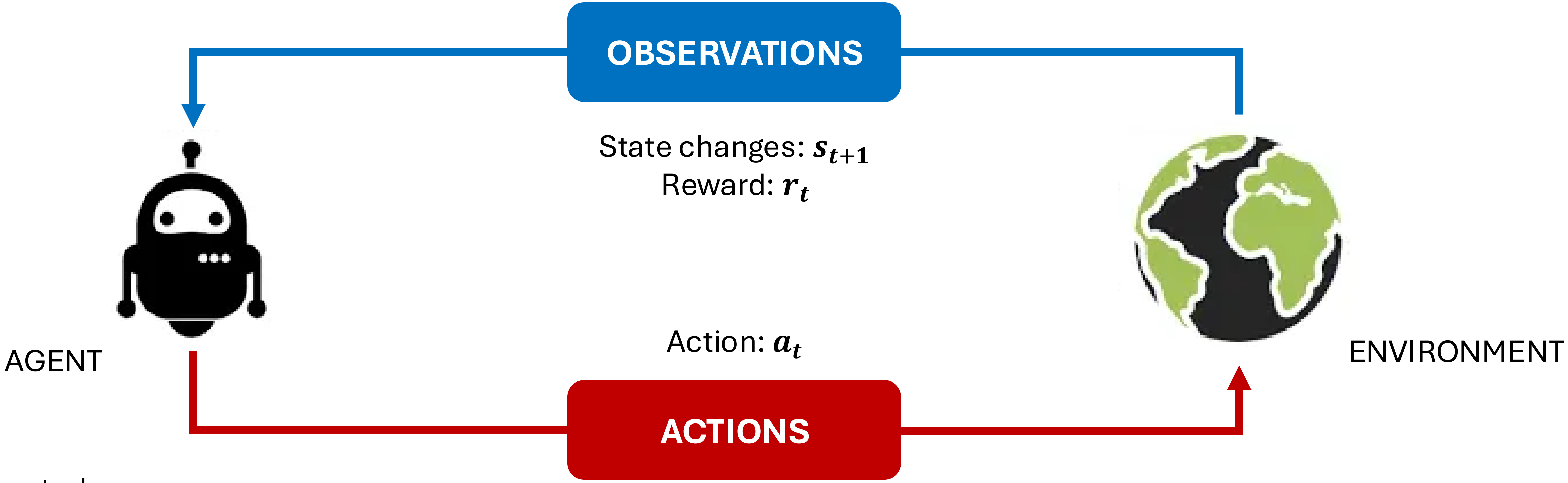
Key Concepts



Total Reward
(Return)

$$R_t = \sum_{i=t}^{\infty} r_i = r_t + r_{t+1} \ldots + r_{t+n} + \cdots$$

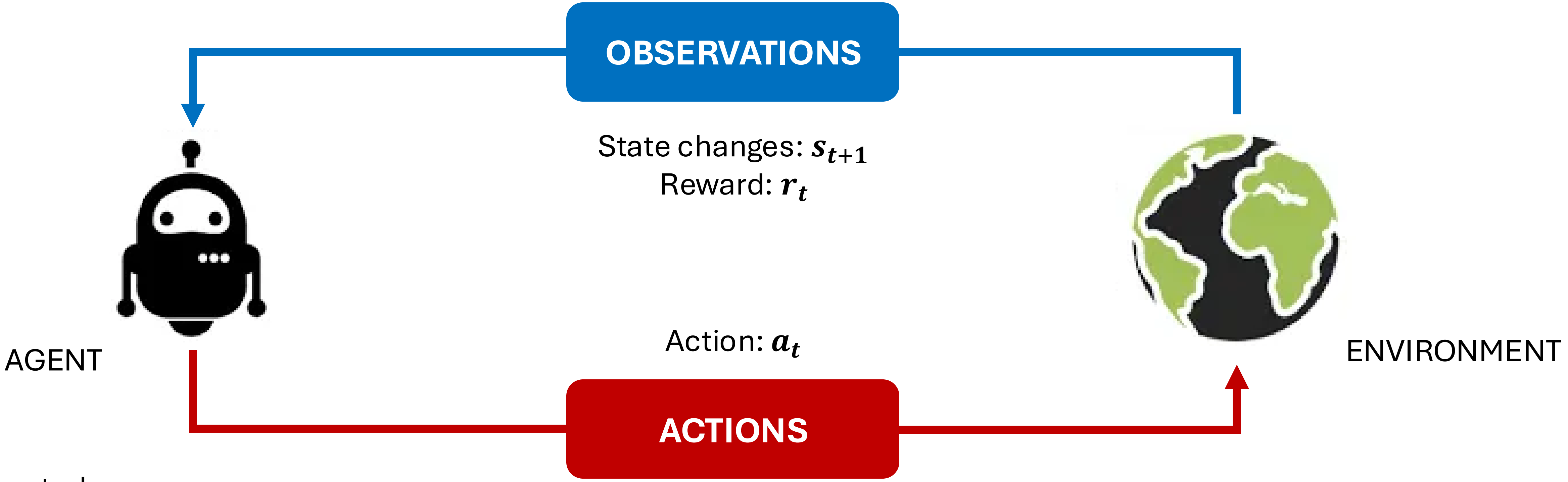
Key Concepts



Discounted
Total Reward
(Return)

$$R_t = \sum_{i=t}^{\infty} \gamma^i r_i$$

Key Concepts



Discounted Total Reward **(Return)**

$$R_t = \sum_{i=t}^{\infty} \gamma^i r_i = \gamma^t r_t + \gamma^{t+1} r_{t+1} \ldots + \gamma^{t+n} r_{t+n} + \cdots$$

$$\gamma: \text{ discount factor; } 0 < \gamma < 1$$

Defining the Q-function

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

Total reward, $\boldsymbol{R_t}$, is the discounted sum of all rewards obtained from time $\boldsymbol{t}$

Defining the Q-function

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

Total reward, ***R_t***, is the discounted sum of all rewards obtained from time ***t***

$$Q\left(s_t, a_t\right) = \mathbb{E}\left[R_t \mid s_t, a_t\right]$$

The Q-function captures the **expected total future reward** an agent in state, ***s***, can receive by executing a certain action, ***a***

How to take actions given a Q-function?

$$Q\left(s_t, a_t\right) = \mathbb{E}\left[R_t \mid s_t, a_t\right]$$

(State, action)

Ultimately, the agent needs a policy $\boldsymbol{\pi}(\boldsymbol{s})$, to infer the best action to take at its state, $\boldsymbol{s}$

How to take actions given a Q-function?

$$Q\left(s_t, a_t\right) = \mathbb{E}\left[R_t \mid s_t, a_t\right]$$

(State, action)

Ultimately, the agent needs a policy $\boldsymbol{\pi(s)}$, to infer the **best action to take** at its state, ***s***

Strategy: the policy should choose an action that maximizes future rewards

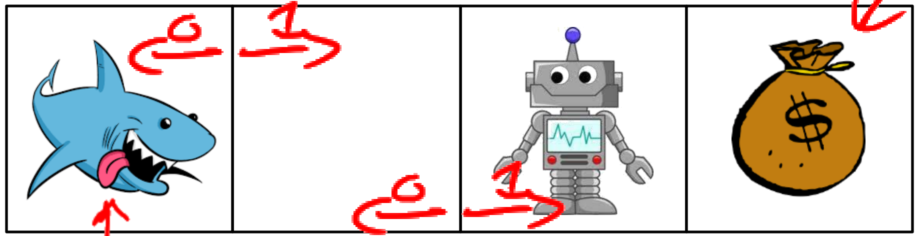
$$\pi^*(s) = \underset{a}{\operatorname{argmax}} Q(s, a)$$

Shark Tank: The Game

state
0 1

environment

r=+1



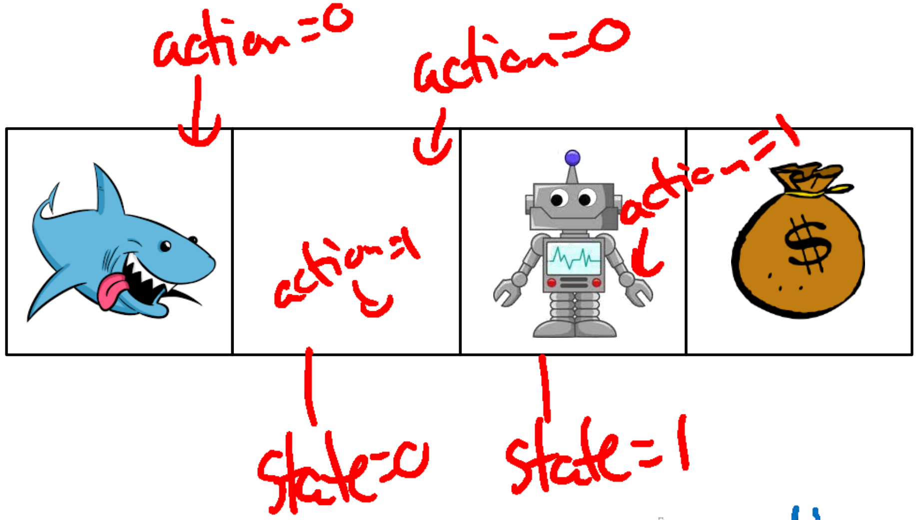
r=-1

action

current
state: position of robot

Shark Tank: The Game

We need a **policy**, π, that informs us which **action**, a, to take in a given **state**, s.



π : Q-learning

in a state s
choose action a
with highest quality Q

state-action pair

discount factor

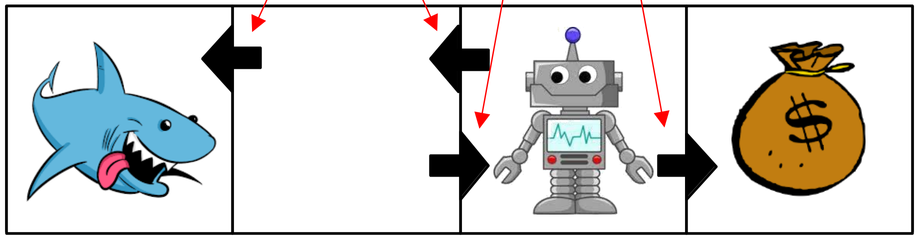
$$Q(s,a) = \sum_{t=0}^{\infty} \gamma^t r_{t+1}$$

expected future rewards if acting optimally

time steps

Shark Tank: The Game

$$Q(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \,\middle|\, s_0 = s, a_0 = a\right]$$



$$Q(s,a) = \sum_{t=0}^{\infty} \gamma^t r_{t+1}$$

say $\gamma = 0.95$

$Q(1,1) = 0.95^0 \cdot 1 = 1$

$Q(0,0) = 0.95^0 \cdot (-1) = -1$

$Q(0,1) = \boxed{0.95^0 \cdot 0 + 0.95^1 \cdot 1} = 0.95$ $\quad ? = \gamma \cdot$

$Q(1,0) = 0.95^0 \cdot 0 + \boxed{0.95^1 \cdot 0 + 0.95^2 \cdot 1} = 0.9025$

policy transform a to s

$a = \pi(s) = \arg\max_a Q(s,a)$

$\pi(1) = \arg\max_a [0.9025, 1]$ $\quad$ index 0 $\quad$ index 1

$= \underline{\underline{1}}$

There is a smarter way

$$Q(s,a) = r + \gamma \max_{a'} Q\left(s', a'\right)$$

← initialize all Q's to zero and update according to this

In practice, the above method will diverge, so we usually introduce a learning rate to slow things down:

learning rate

$$Q(s,a) = (1-\alpha)\,Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a')\right)$$

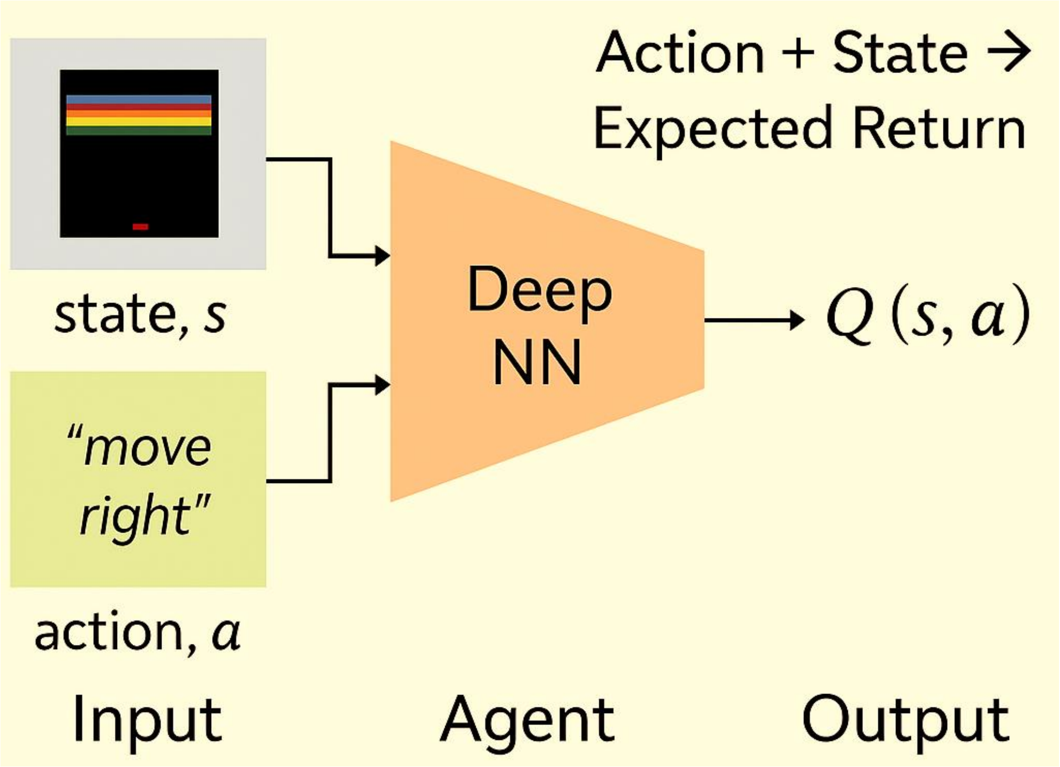
Quickly becomes near impossible

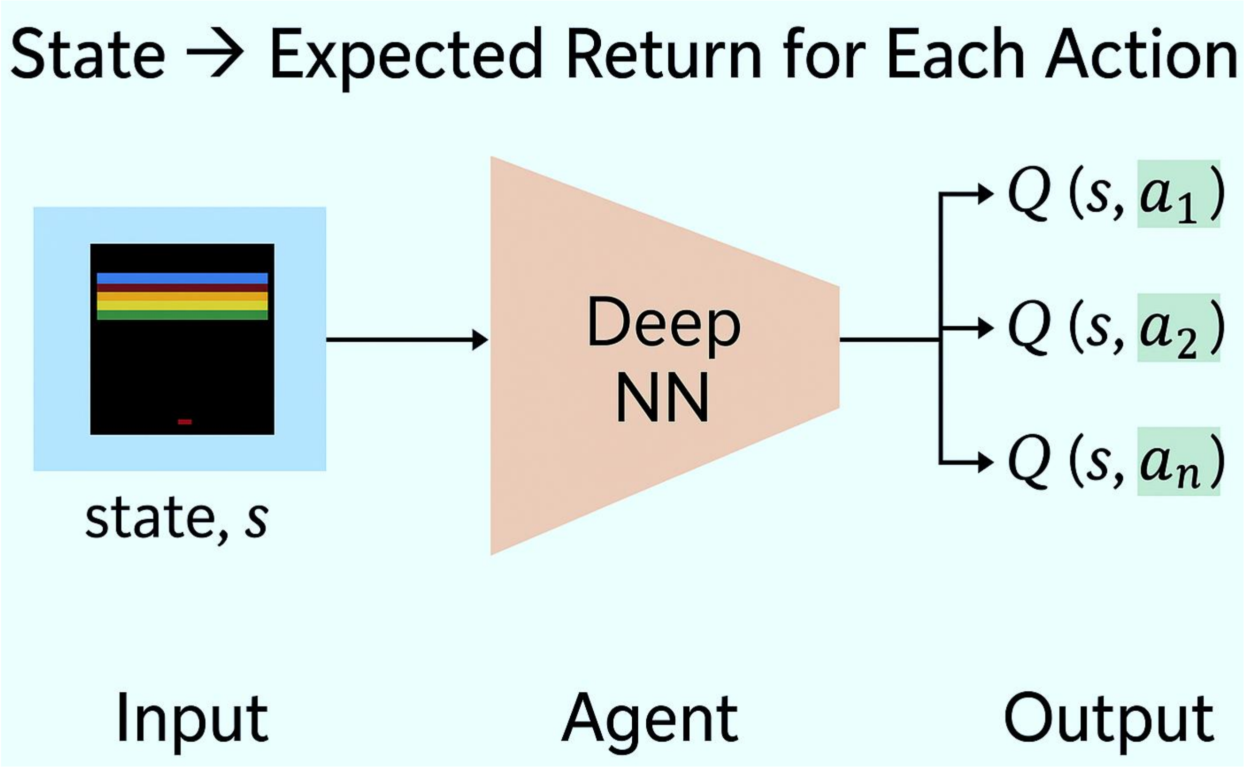
The states are continuous, there's essentially infinitely many

Approximately 10.000.000.000.000.000.000.000.000.000.000.000.000.000.000.000 different states

Deep Q Networks (DQN)

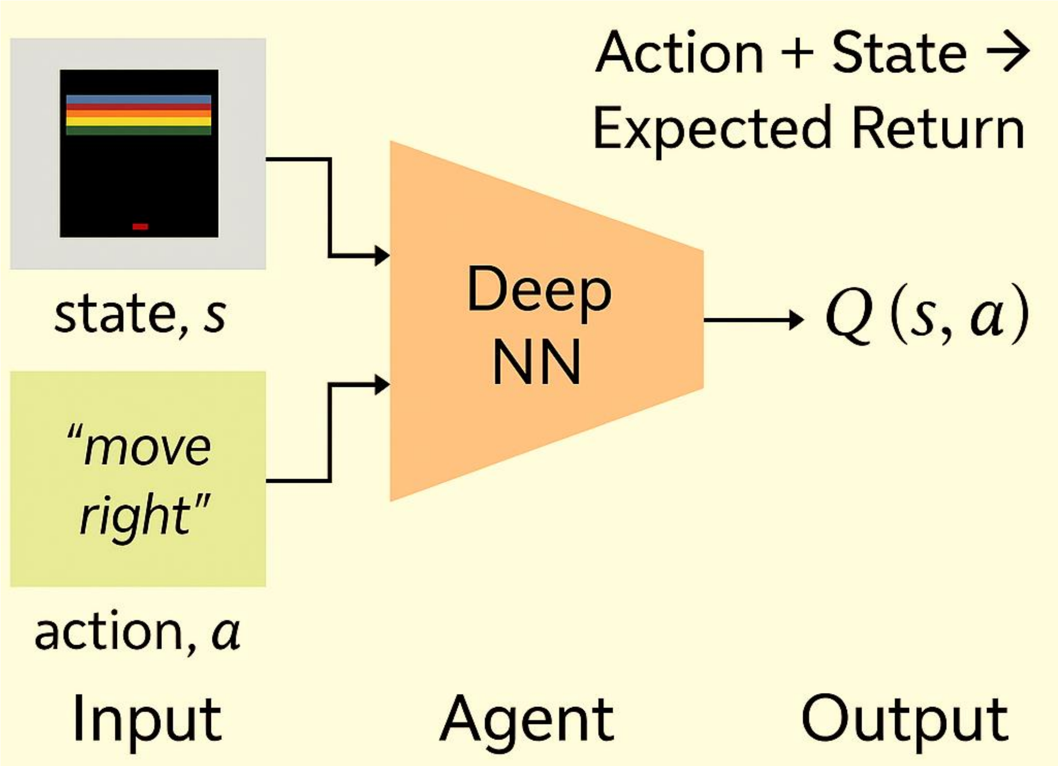
How can we use deep neural networks to model Q-functions?

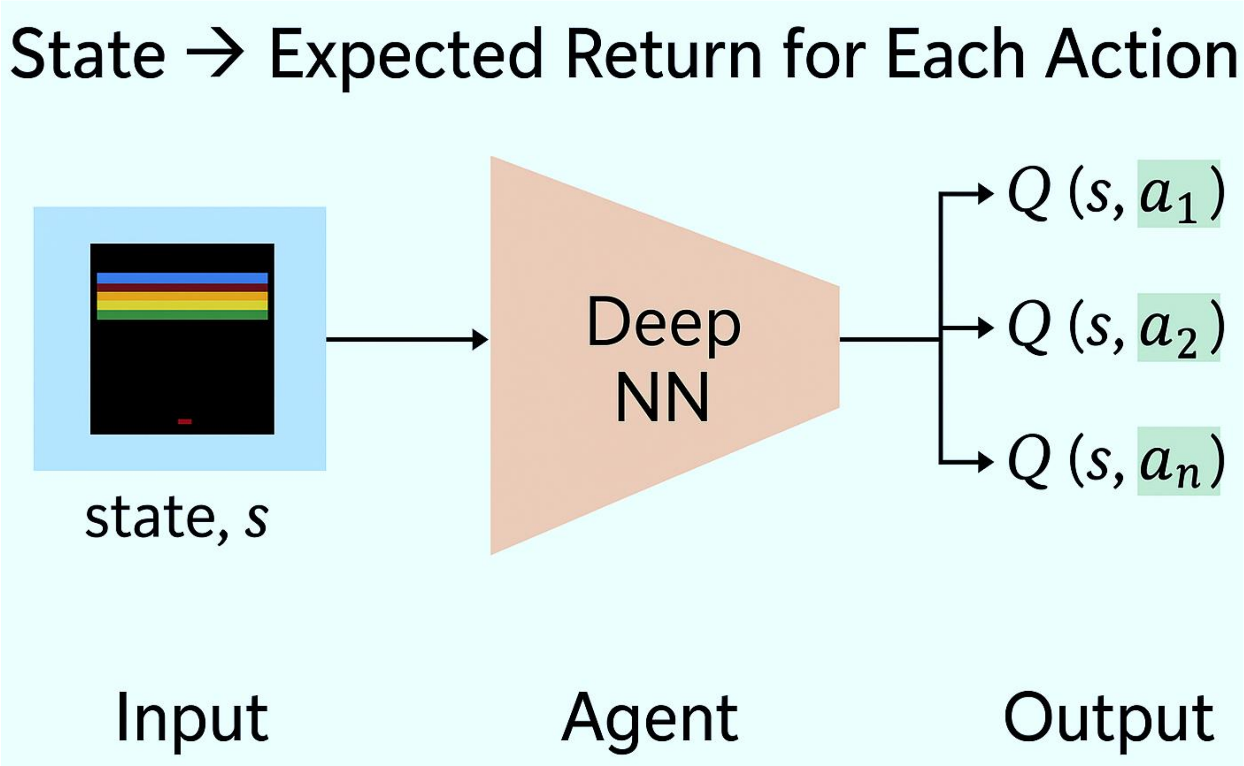




Deep Q Networks (DQN): Training

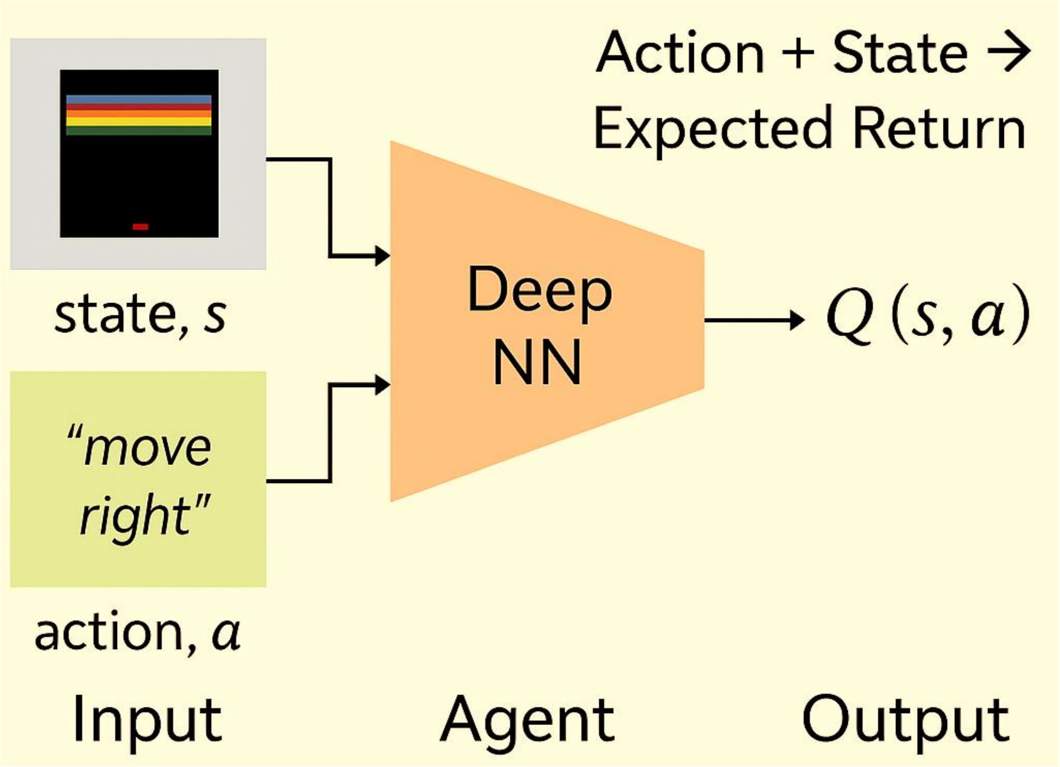
How can we use deep neural networks to model Q-functions?

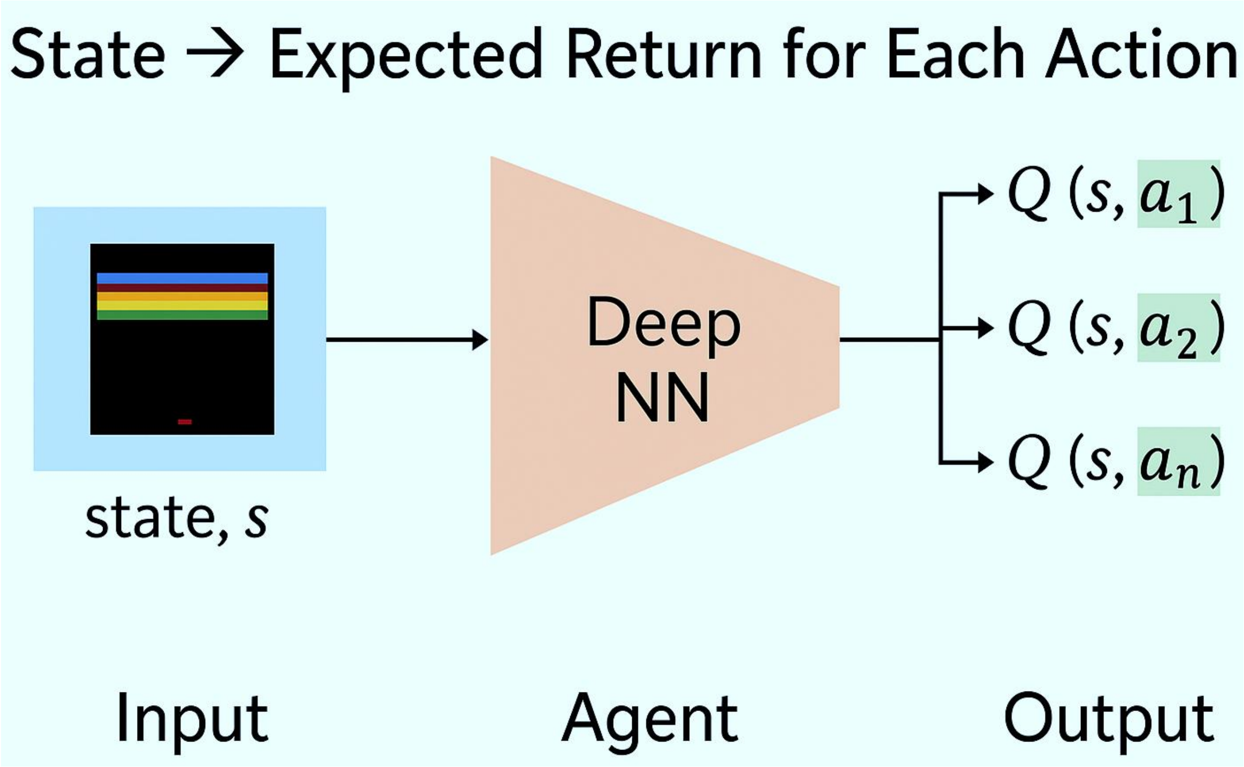




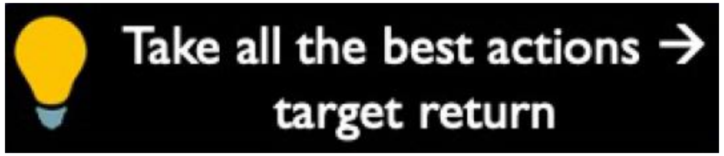
Deep Q Networks (DQN): Training

How can we use deep neural networks to model Q-functions?



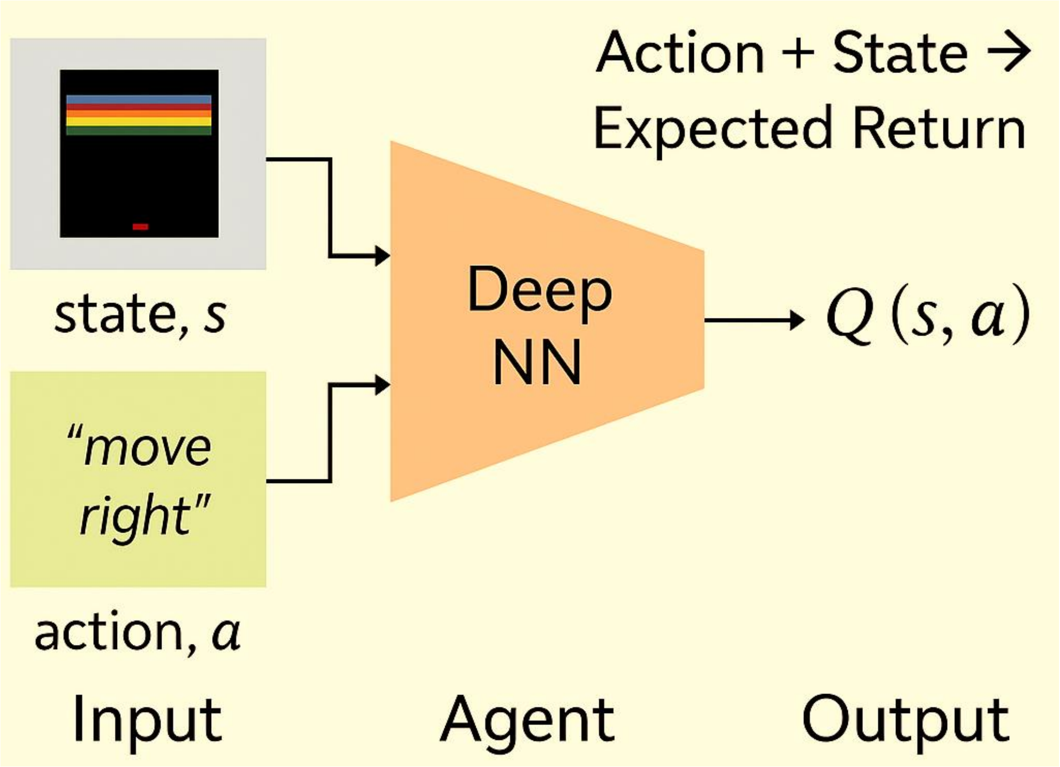


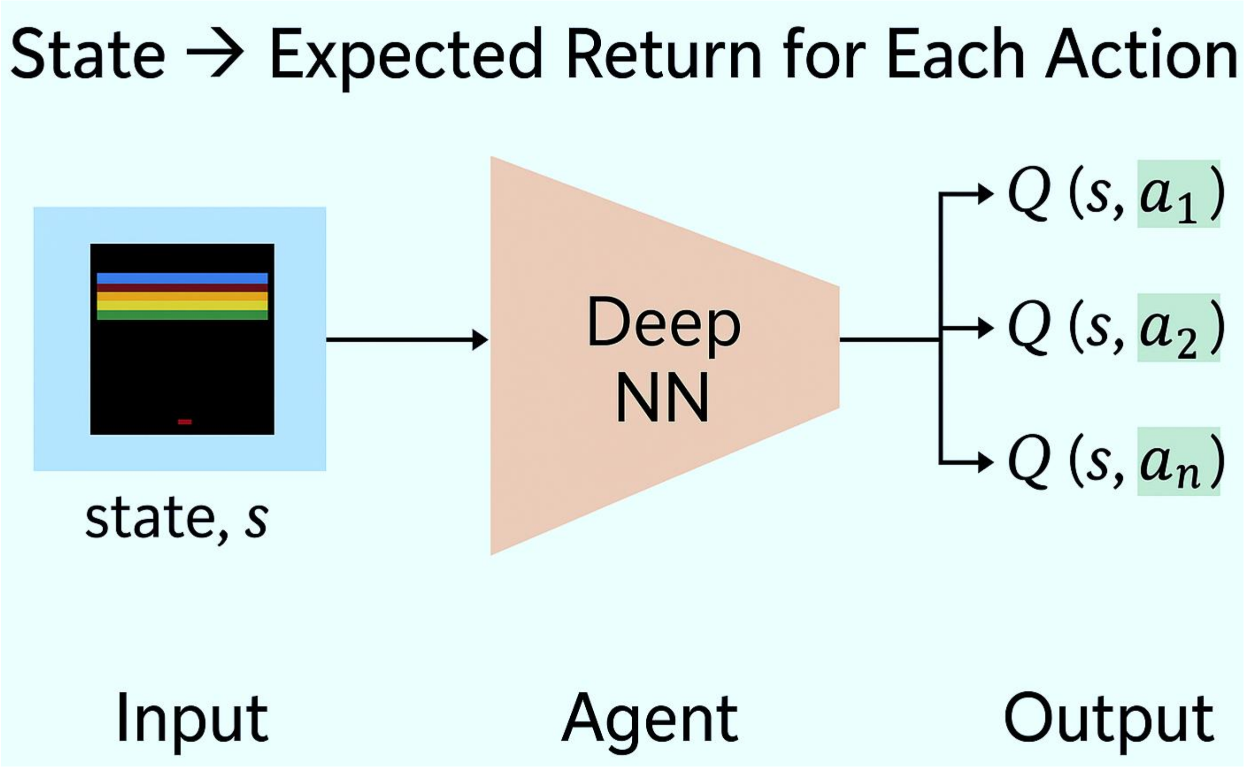
$$\overbrace{\left(r + \gamma \max_{a'} Q\left(s', a'\right)\right)}^{\text{target}}$$



Deep Q Networks (DQN): Training

How can we use deep neural networks to model Q-functions?

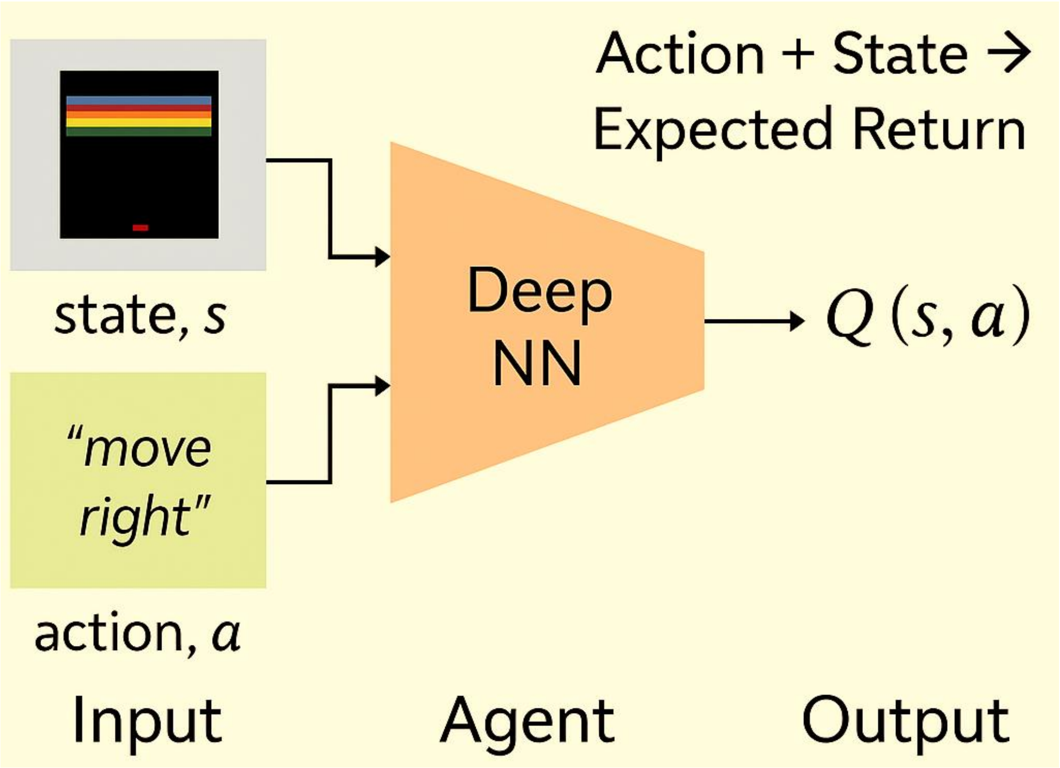


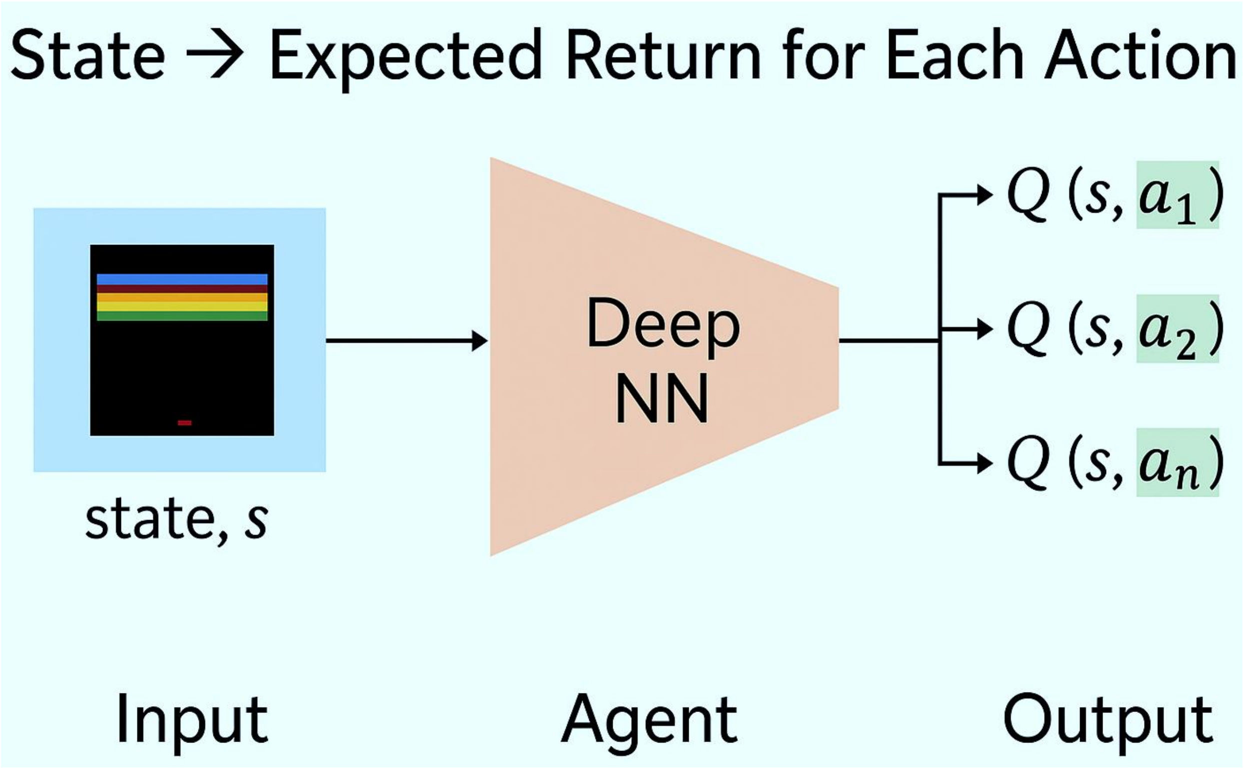


$$\overbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}^{\text{target}} \qquad \overbrace{Q(s, a)}^{\text{predicted}}$$

Deep Q Networks (DQN): Training

How can we use deep neural networks to model Q-functions?



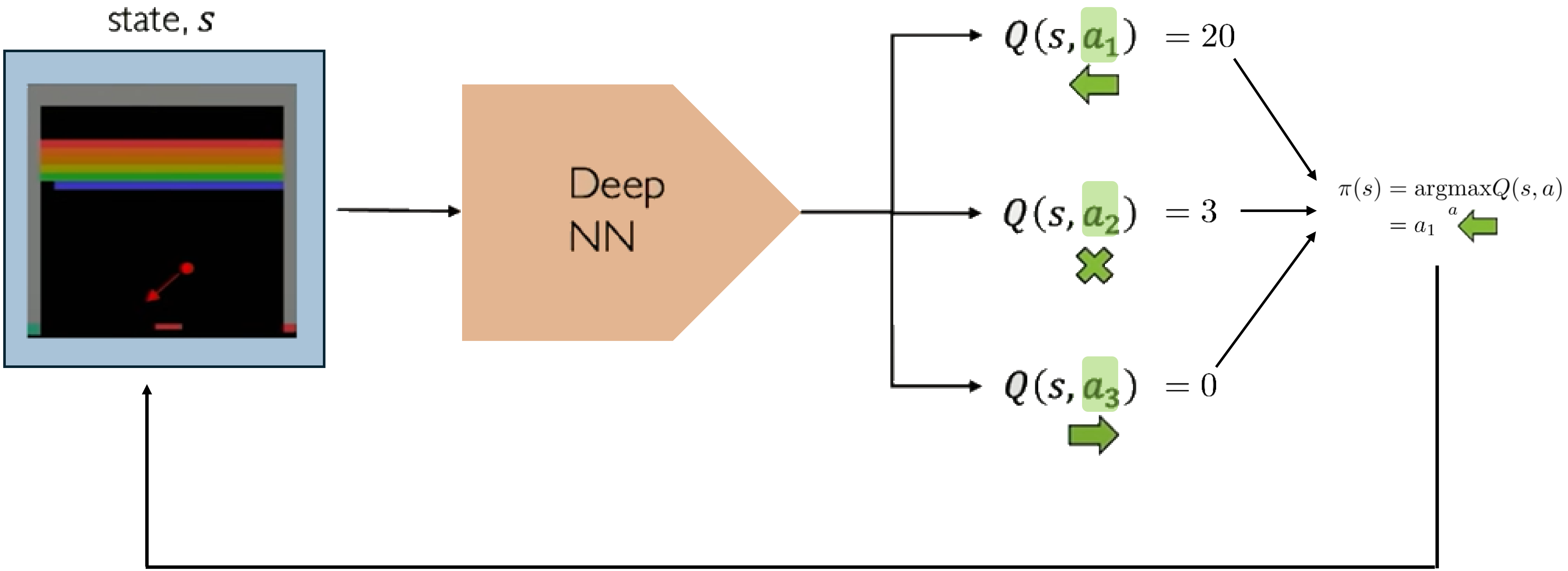


$$\mathcal{L} = \mathbb{E}[\|(\overbrace{\left(r + \gamma \max_{a'} Q\left(s', a'\right)\right)}^{\text{target}} - \overbrace{Q(s,a)}^{\text{predicted}} \|^2]$$

Q-Loss

Deep Q Network Summary

Use NN to learn Q -function and then use to infer the optimal policy, $\boldsymbol{\pi(s)}$



Send action back to environment and receive next state

Two Families of RL Algorithms

Value Learning

Find $Q(s, a)$

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

Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

Two Families of RL Algorithms

Value Learning

Find $Q(s, a)$

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

Policy Learning

Find $\pi(s)$

Sample $a \backsim \pi(s)$

Downsides of Q-Learning

- **Complexity:**
  - Can model scenarios where the action space is discrete and small
  - Cannot handle continuous action spaces

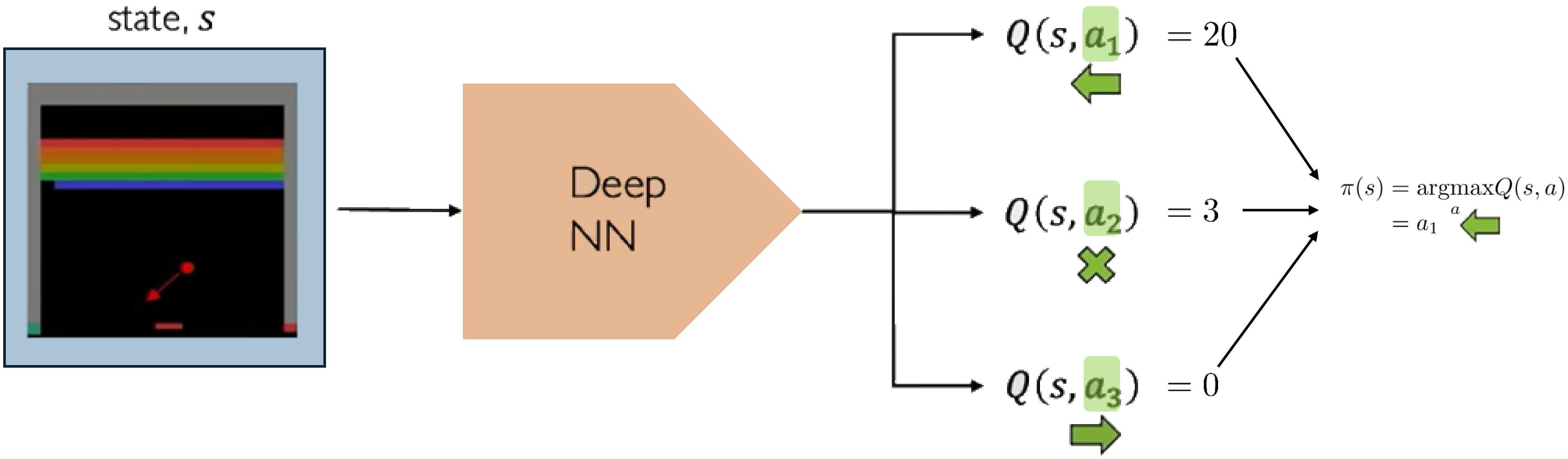
- **Flexibility:**
  - Policy is deterministically computed from the Q function by maximizing the reward → cannot learn stochastic policies

To address these, consider a new class of RL training algorithms:

Policy gradient methods

Deep Q Network Summary

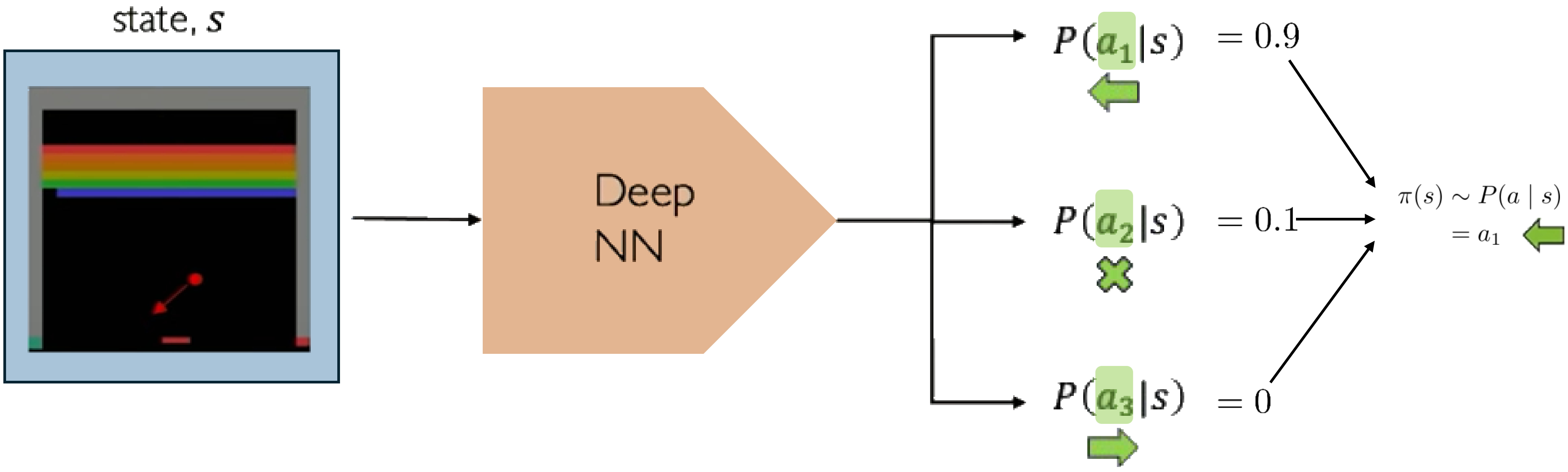
DQN: Approximate Q-function and use to infer the optimal policy, $\boldsymbol{\pi(s)}$



Policy Gradient (PG): Key Idea

DQN: Approximate Q-function and use to infer the optimal policy, $\pi(s)$

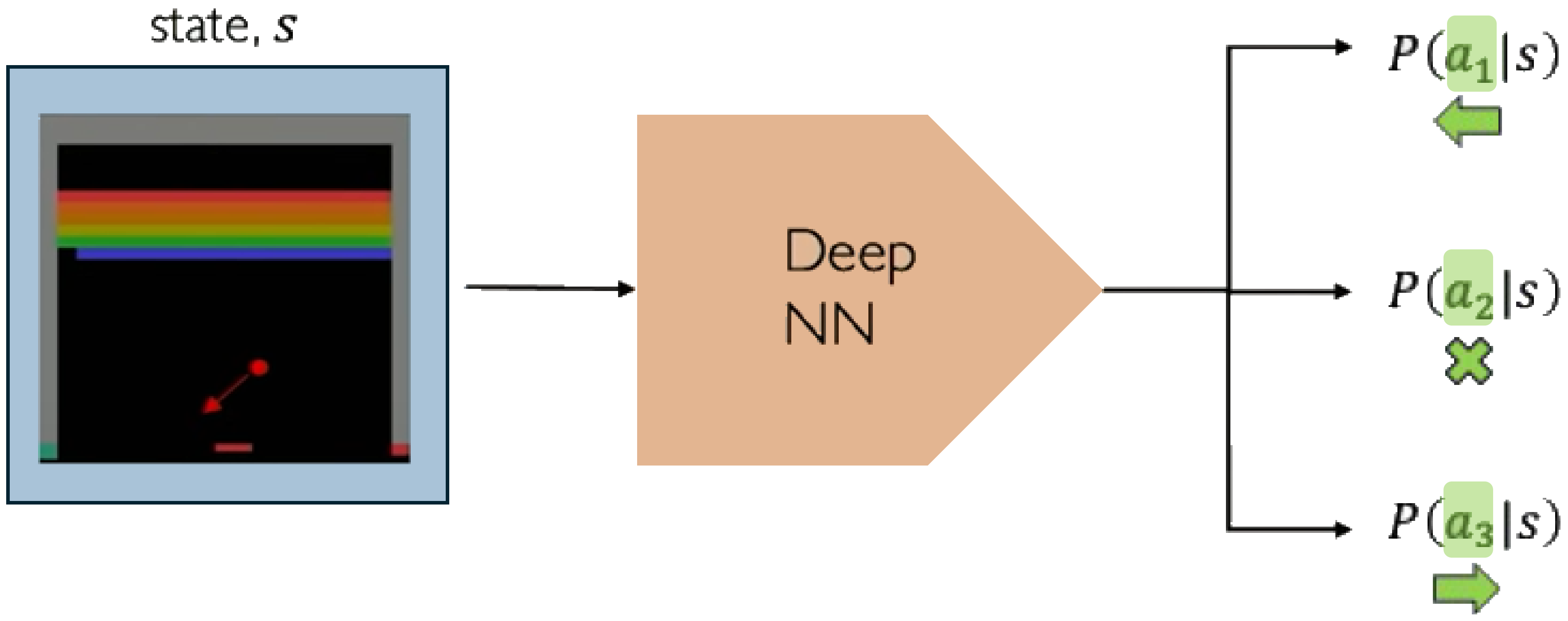
Policy Gradient: Directly optimize the policy, $\pi(s)$



Policy Gradient (PG): Key Idea

DQN: Approximate Q-function and use to infer the optimal policy, $\pi(s)$

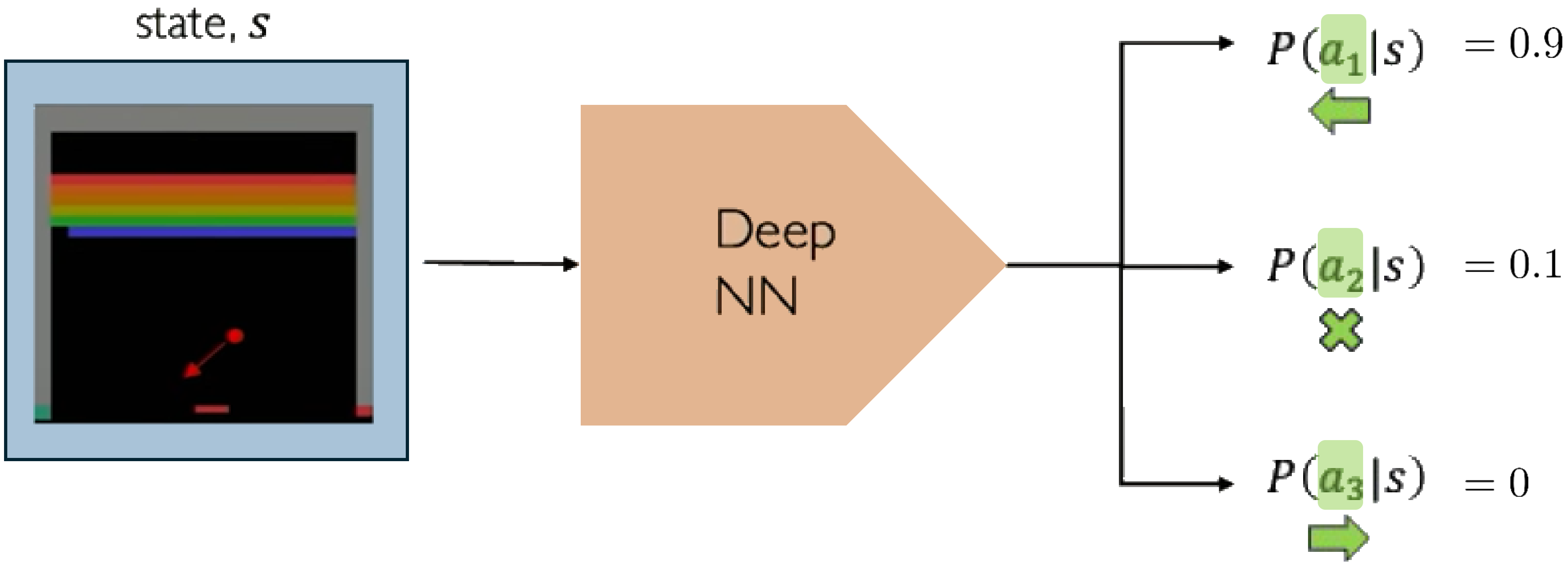
Policy Gradient: Directly optimize the policy, $\boldsymbol{\pi(s)}$



Policy Gradient (PG): Key Idea

DQN: Approximate Q-function and use to infer the optimal policy, $\pi(s)$

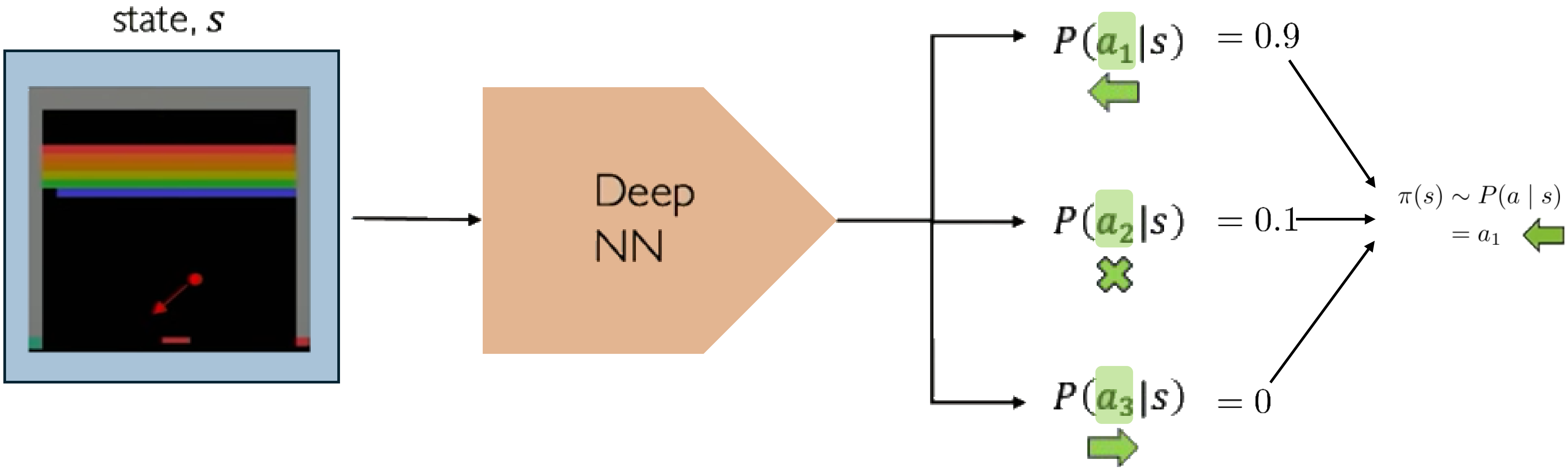
Policy Gradient: Directly optimize the policy, $\boldsymbol{\pi}(\boldsymbol{s})$



Policy Gradient (PG): Key Idea

DQN: Approximate Q-function and use to infer the optimal policy, $\pi(s)$

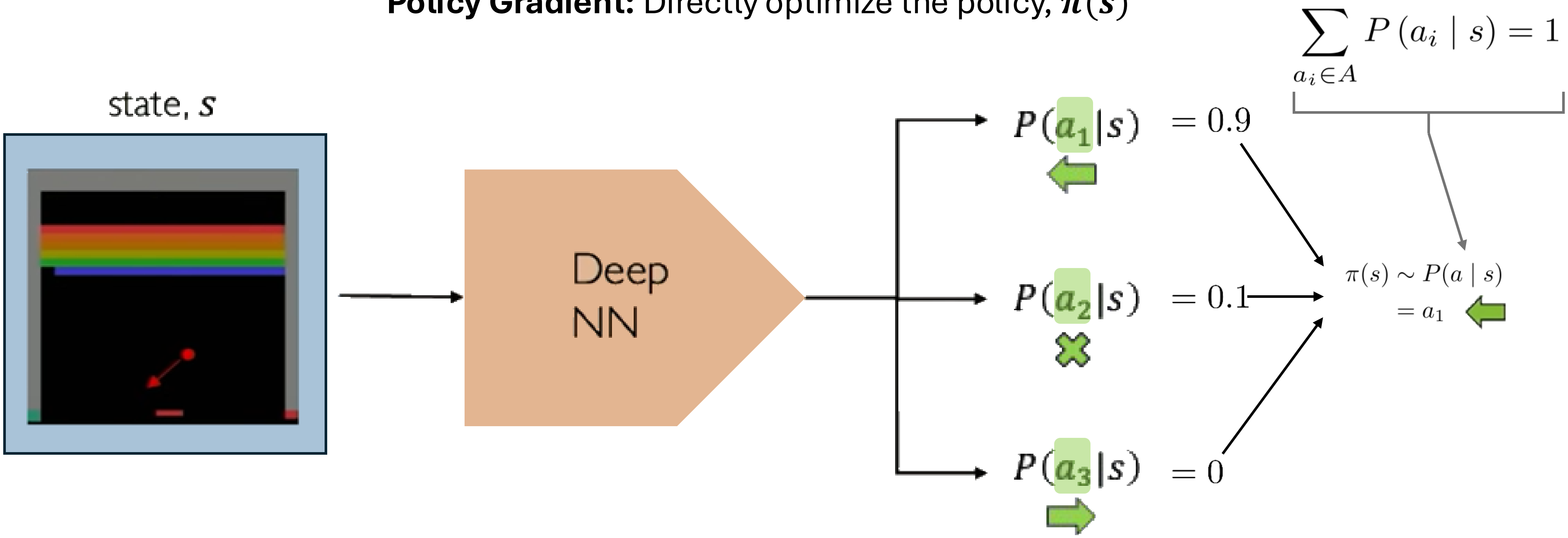
Policy Gradient: Directly optimize the policy, $\pi(s)$



Policy Gradient (PG): Key Idea

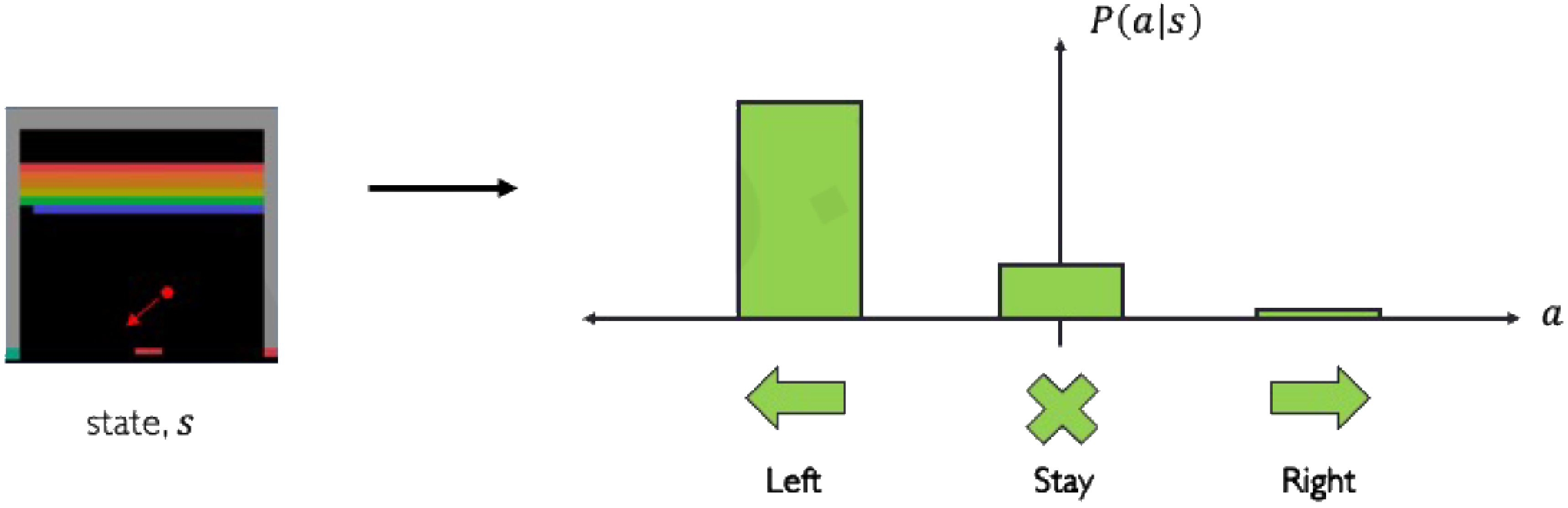
DQN: Approximate Q-function and use to infer the optimal policy, $\pi(s)$

Policy Gradient: Directly optimize the policy, $\pi(s)$



Discrete vs Continuous Action Spaces

Discrete action space: which direction should I move?

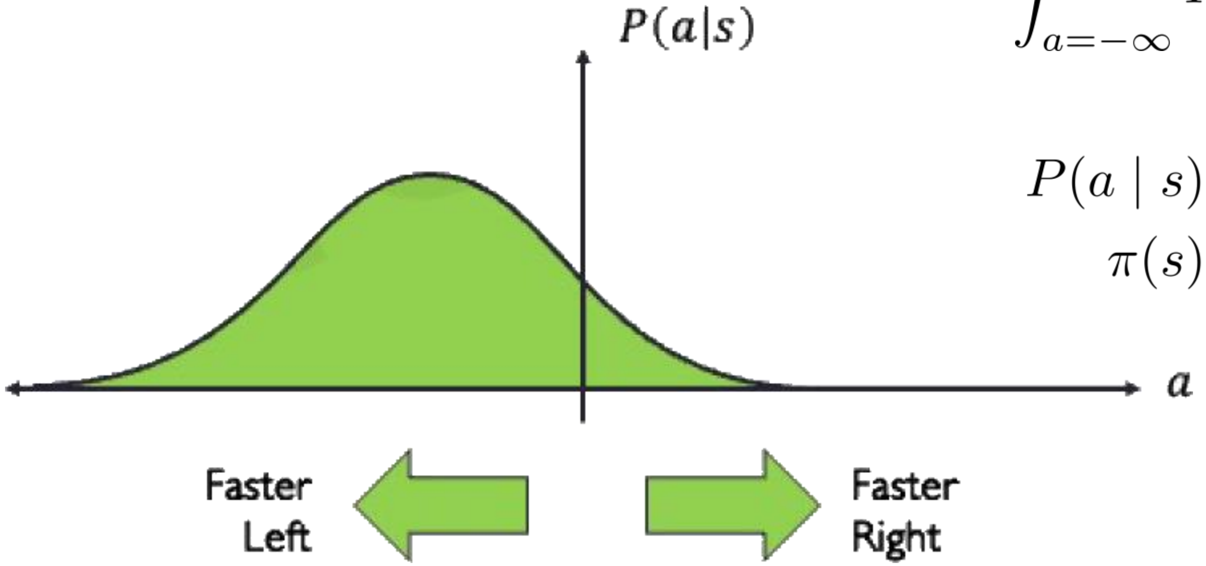


Discrete vs Continuous Action Spaces

Discrete action space: which direction should I move?

Continuous action space: which direction should I move?

state, s



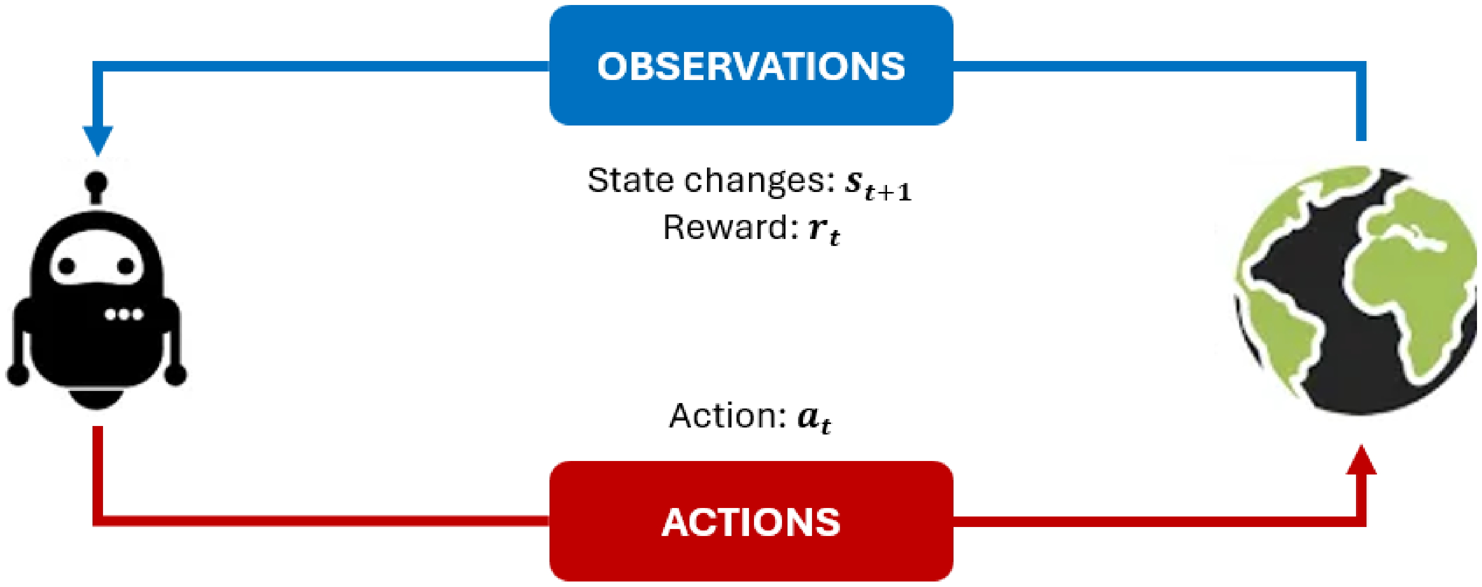
$$\int_{a=-\infty}^{\infty} P(a \mid s) = 1$$

$$P(a \mid s) = N\left(\mu, \sigma^2\right)$$
$$\pi(s) \sim P(a \mid s)$$
$$= -0.8[\mathrm{~m/s}]$$

Policy Gradient: Enables modeling of continuous action space

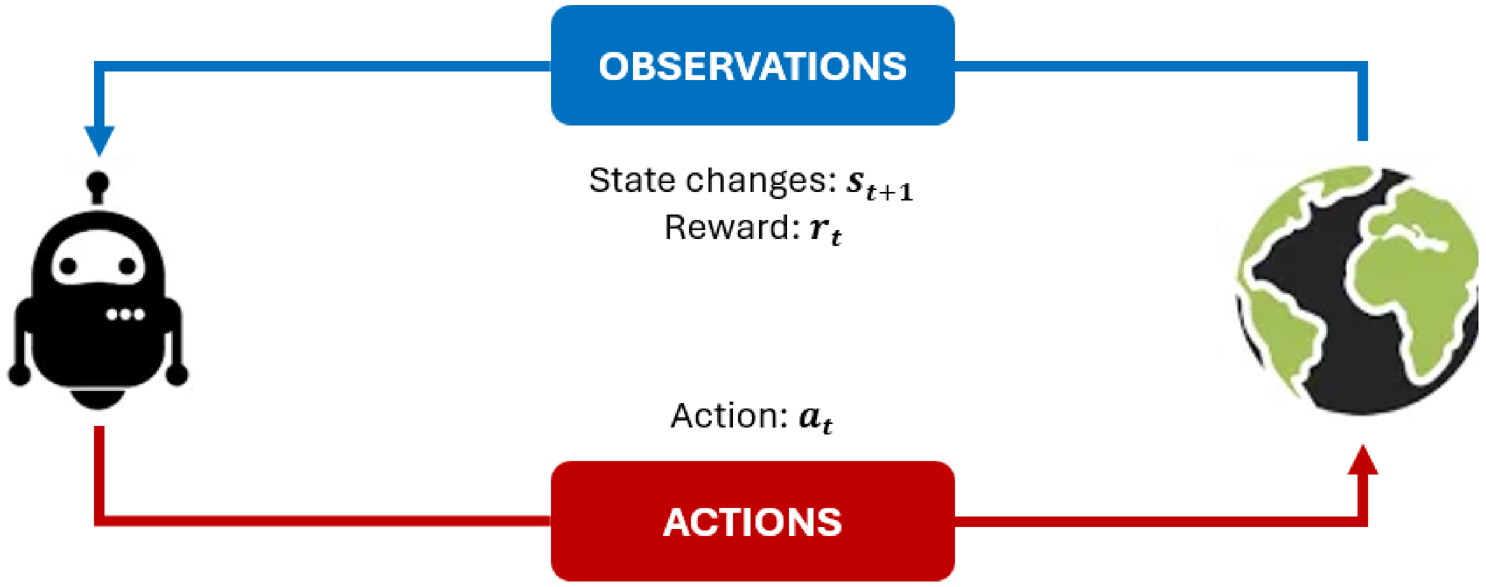
Training Policy Gradients: Case Study

Reinforcement Learning Loop



Training Policy Gradients: Case Study

Reinforcement Learning Loop



Self-Driving Cars

Agent: Vehicle

State: camera, lidar, etc.

Action: steering wheel angle

Reward: distance traveled

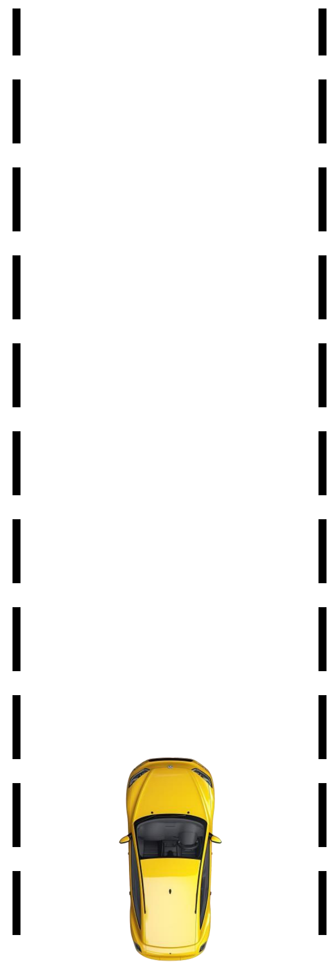
Training Policy Gradients:

Training Algorithm

Training Policy Gradients:

Training Algorithm

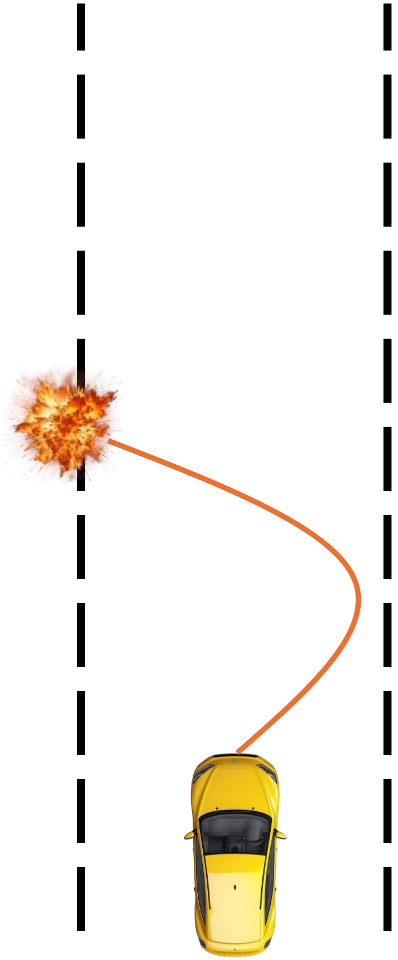
1. Initialize the agent



Training Policy Gradients:

Training Algorithm

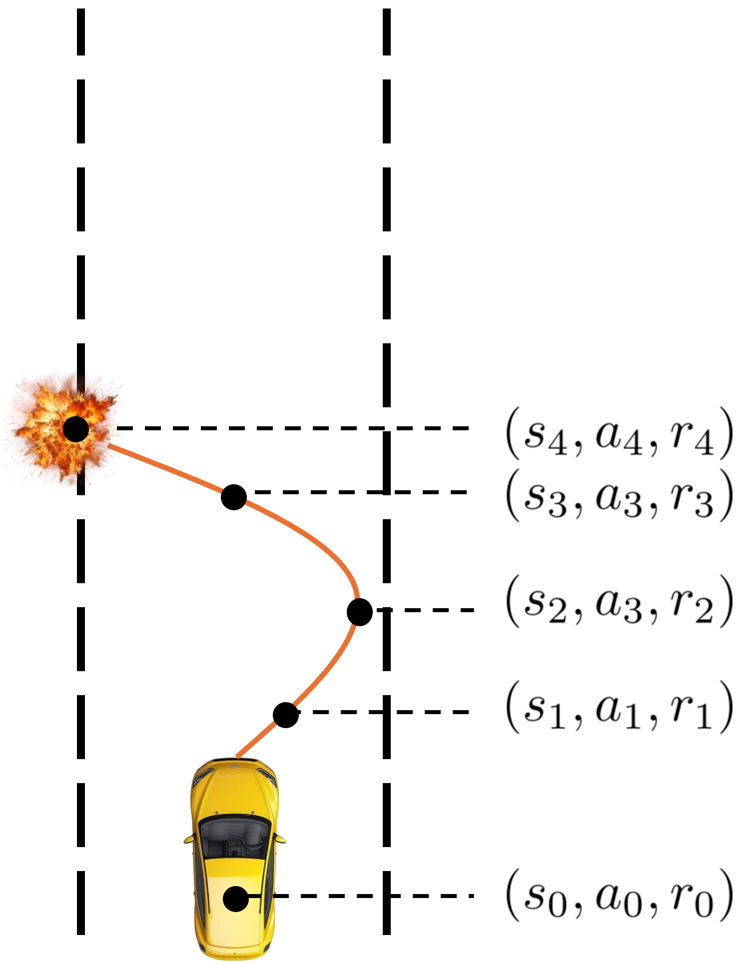
1. Initialize the agent
2. Run a policy until termination



Training Policy Gradients:

Training Algorithm

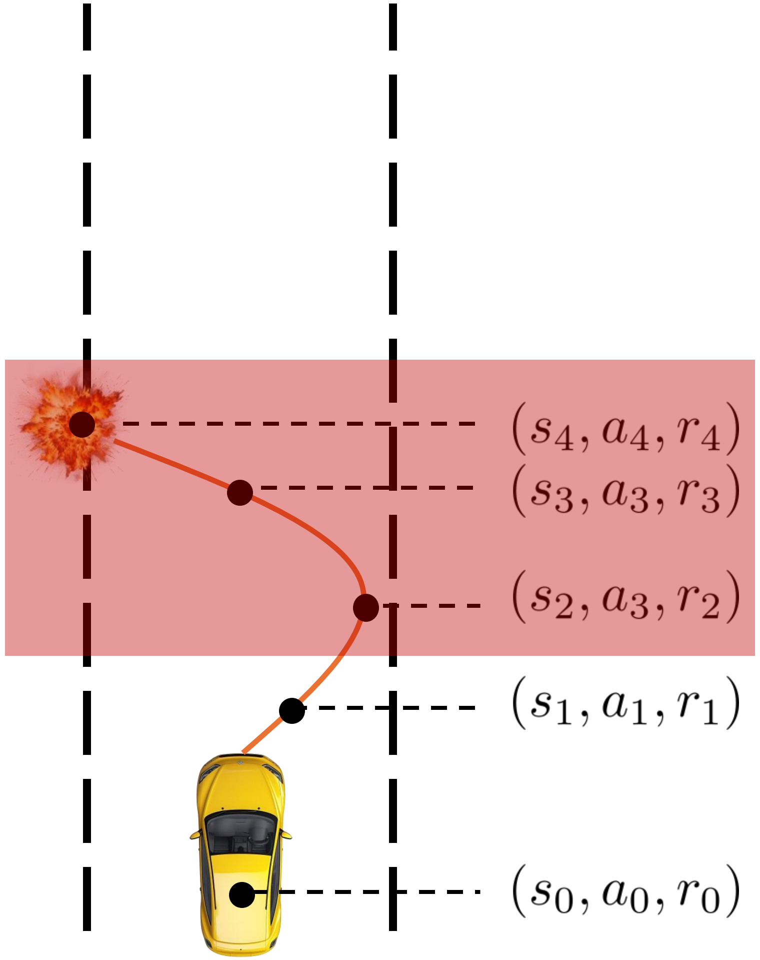
1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards



Training Policy Gradients:

Training Algorithm

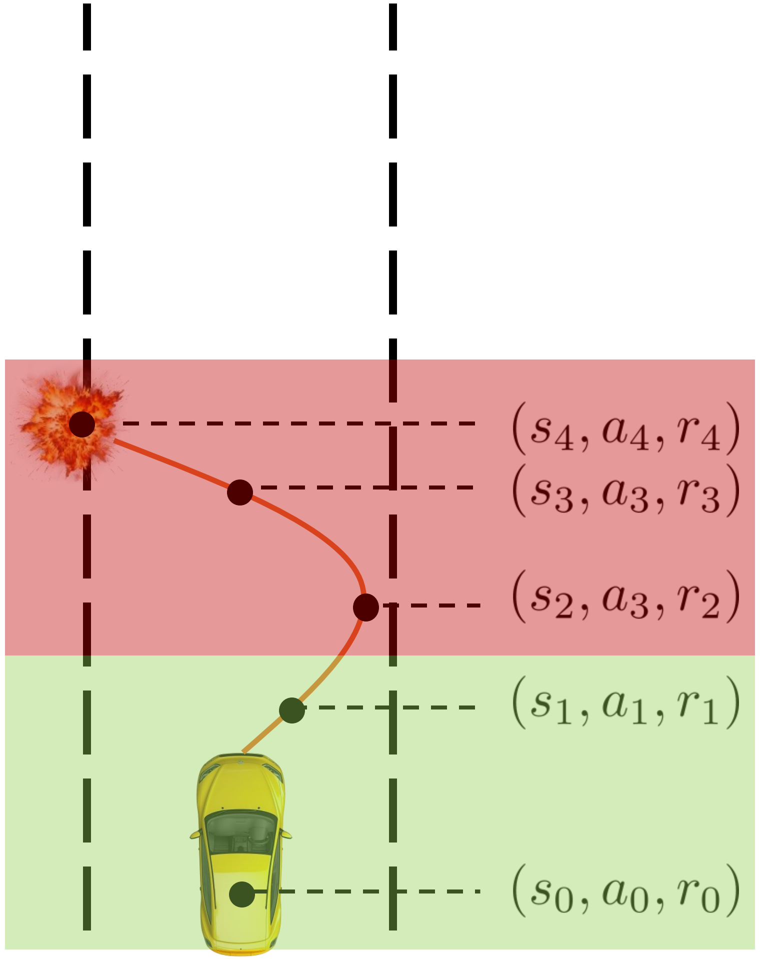
1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrese probability of actions that resulted in low reward



Training Policy Gradients:

Training Algorithm

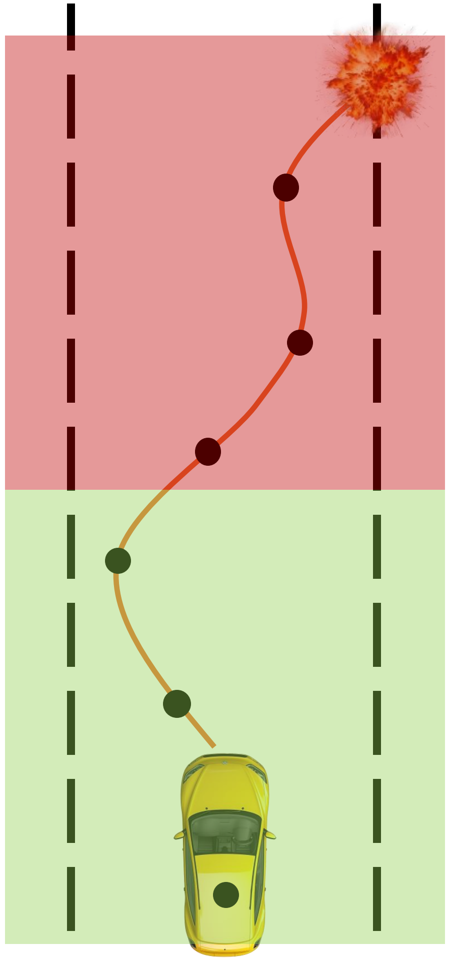
1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrese probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward



Training Policy Gradients:

Training Algorithm

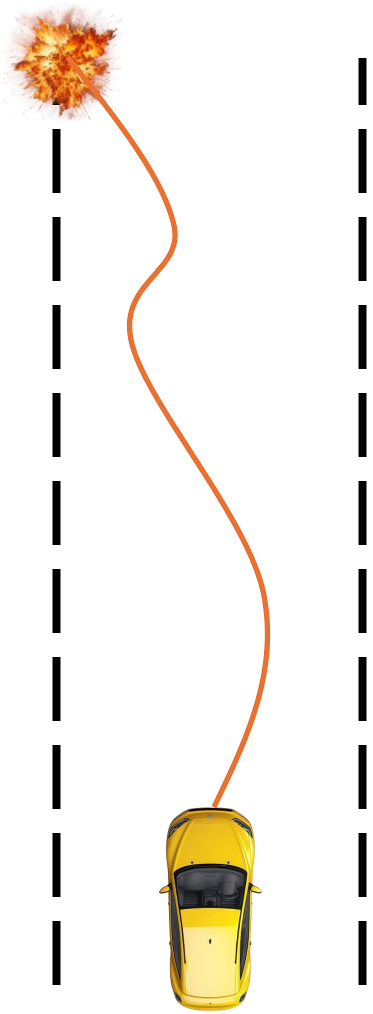
1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrese probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward



Training Policy Gradients:

Training Algorithm

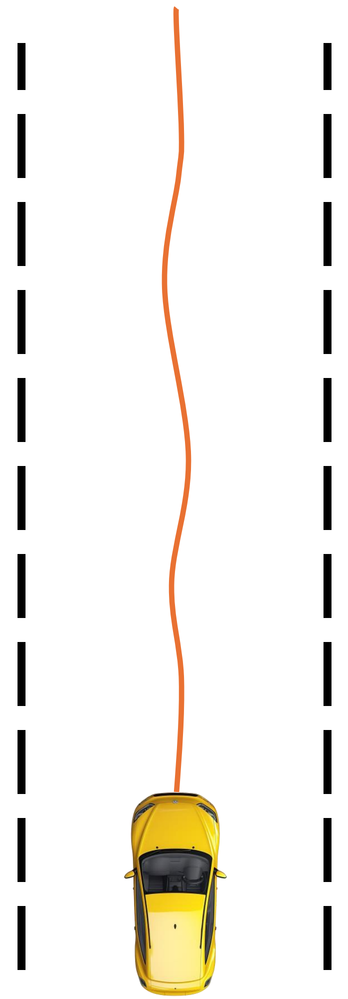
1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrese probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward



Training Policy Gradients:

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrese probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward



Training Policy Gradients:

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrese probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

log-likelihood of action

$$\text{loss} = \underbrace{- \log \mathrm{P}(a_t \mid s_t)}_{} \, R_t$$

reward

Gradient descent update:

$$w' = w - \nabla \text{loss}$$

$$w' = w + \boxed{\nabla \log \mathrm{P}(a_t \mid s_t) \, R_t}$$

Policy gradient!

Reinforcement Learning in Real Life

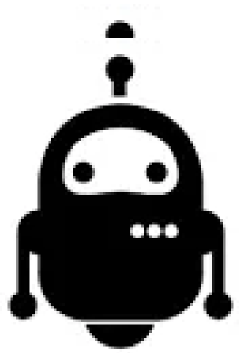
Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrese probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

Reinforcement Learning: Summary

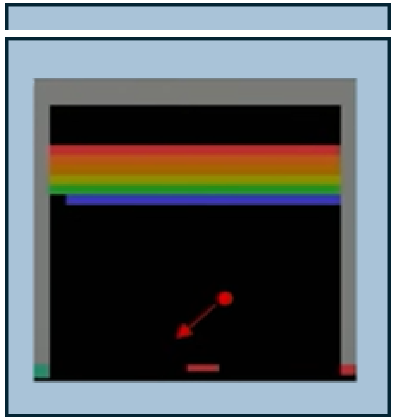
Fondations

- Agents acting in environment
- State-action pairs → maximize future rewards
- Discouting



Q-Learning

- Q function: expected total reward given s, a
- Policy determined by selecting action that maximizes Q function



Policy Gradients

- Learn and optimize the policy directly
- Applicable to continuous action spaces