TDDE78 Reinforcement Learning

Value-Based and Policy Based RL

Amath Sow

PhD student ReaL Lab

LINKÖPINGS UNIVERSITET

Reinforcement Learning: Concepts

Agent: takes actions.

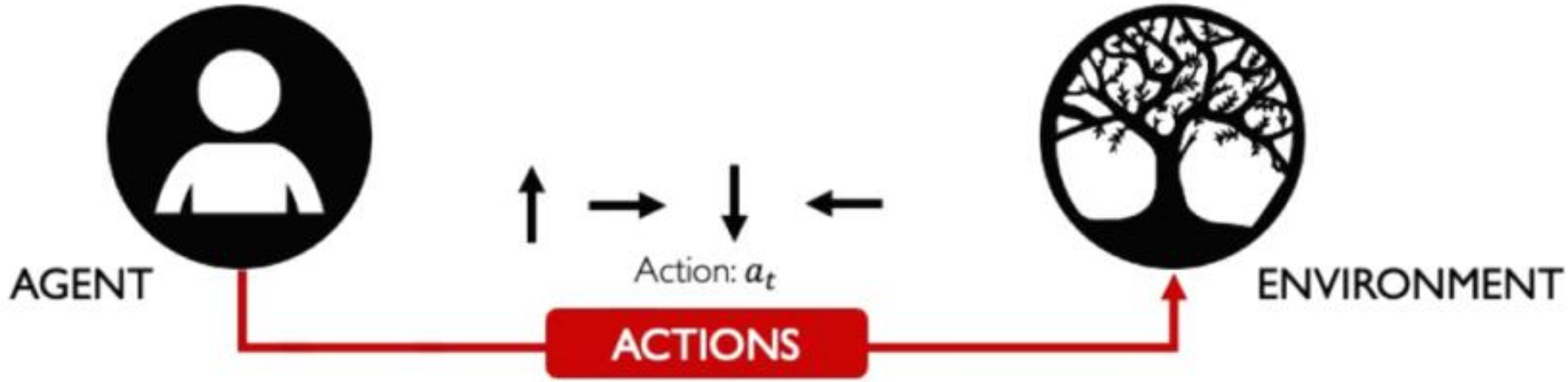
Reinforcement Learning: Concepts

AGENT

ENVIRONMENT

Environment: the world in which the agent exists and operates.

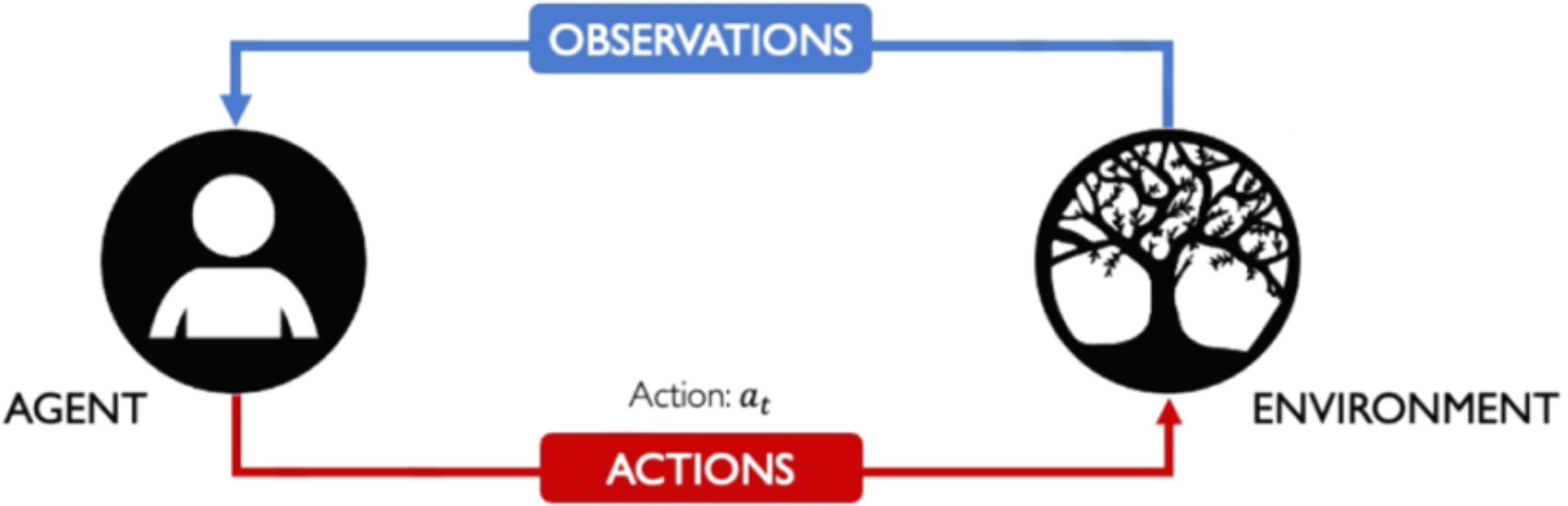
Reinforcement Learning: Concepts



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

Reinforcement Learning: Concepts



Observations: of the environment after taking actions.

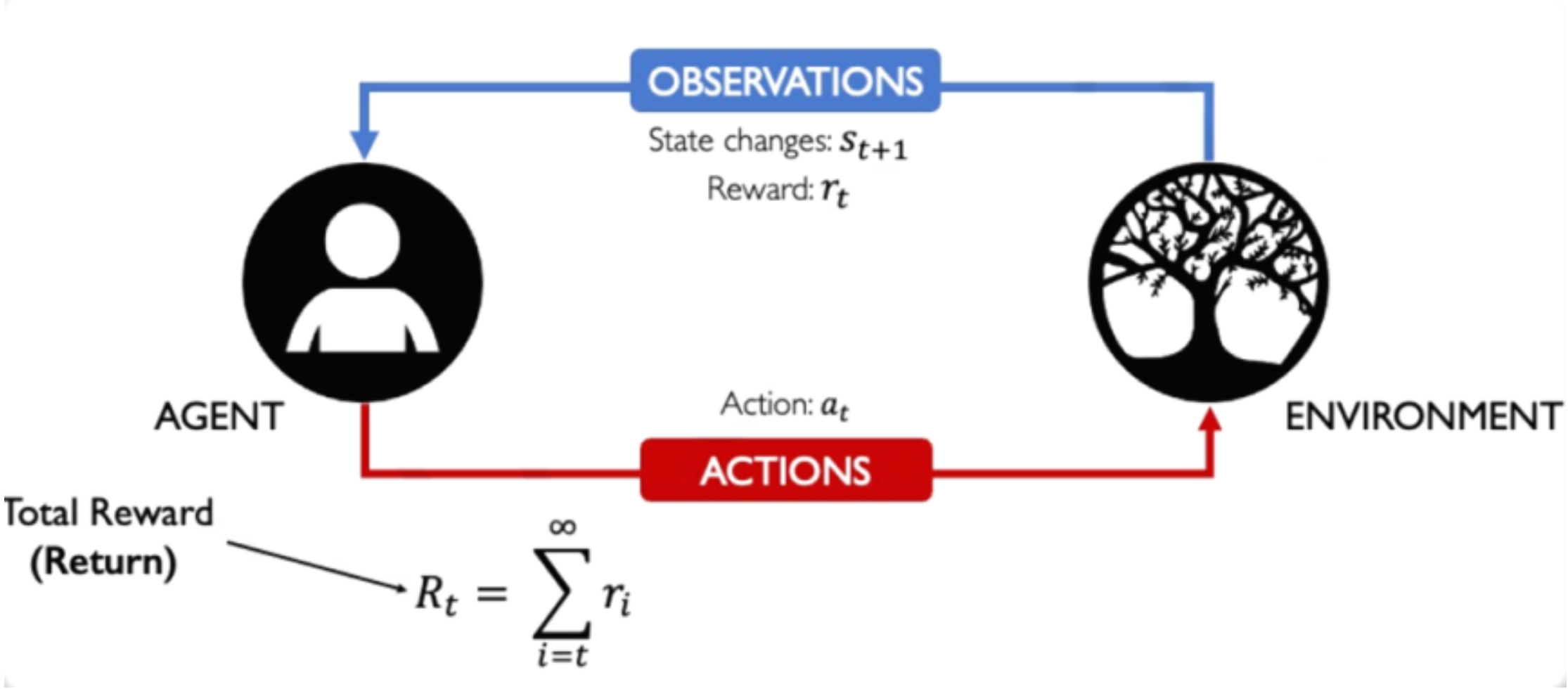
Reinforcement Learning: Concepts

State: a situation which the agent perceives.

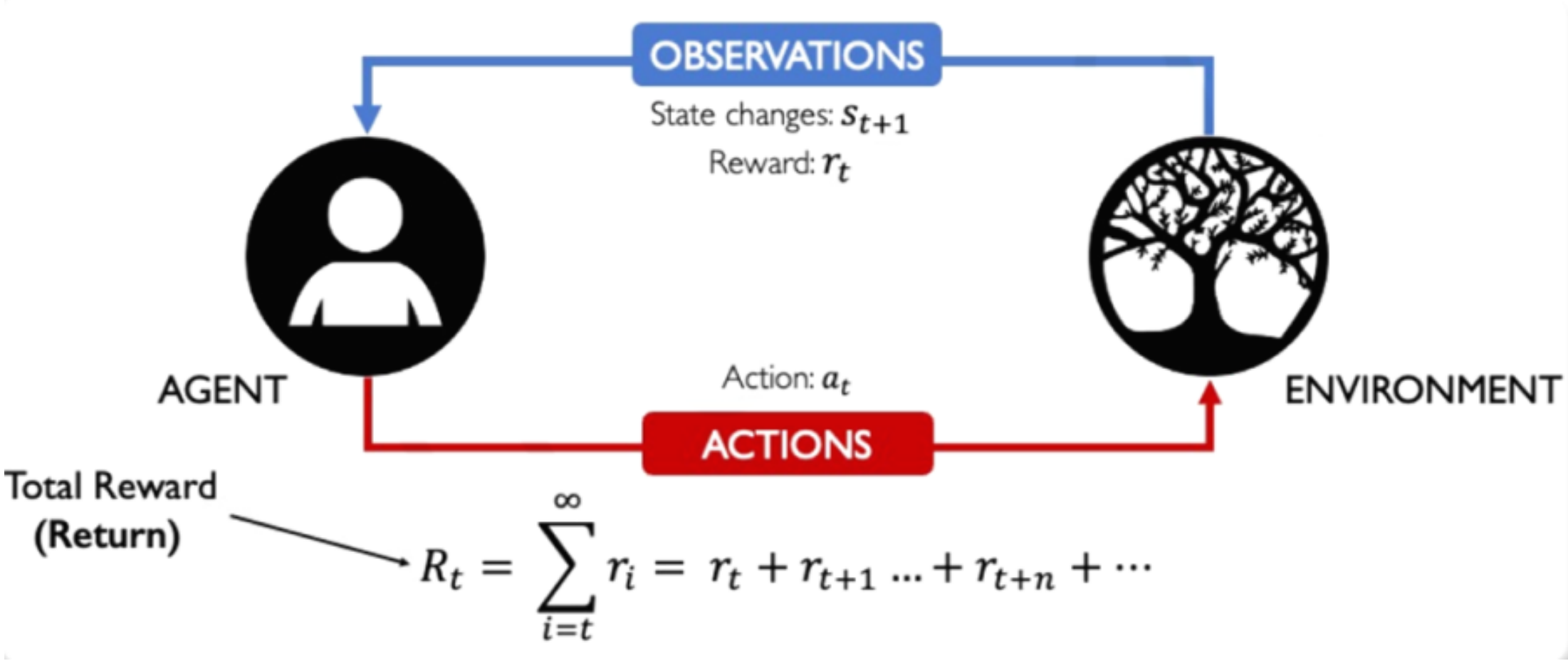
Reinforcement Learning: Concepts

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

Reinforcement Learning: Concepts



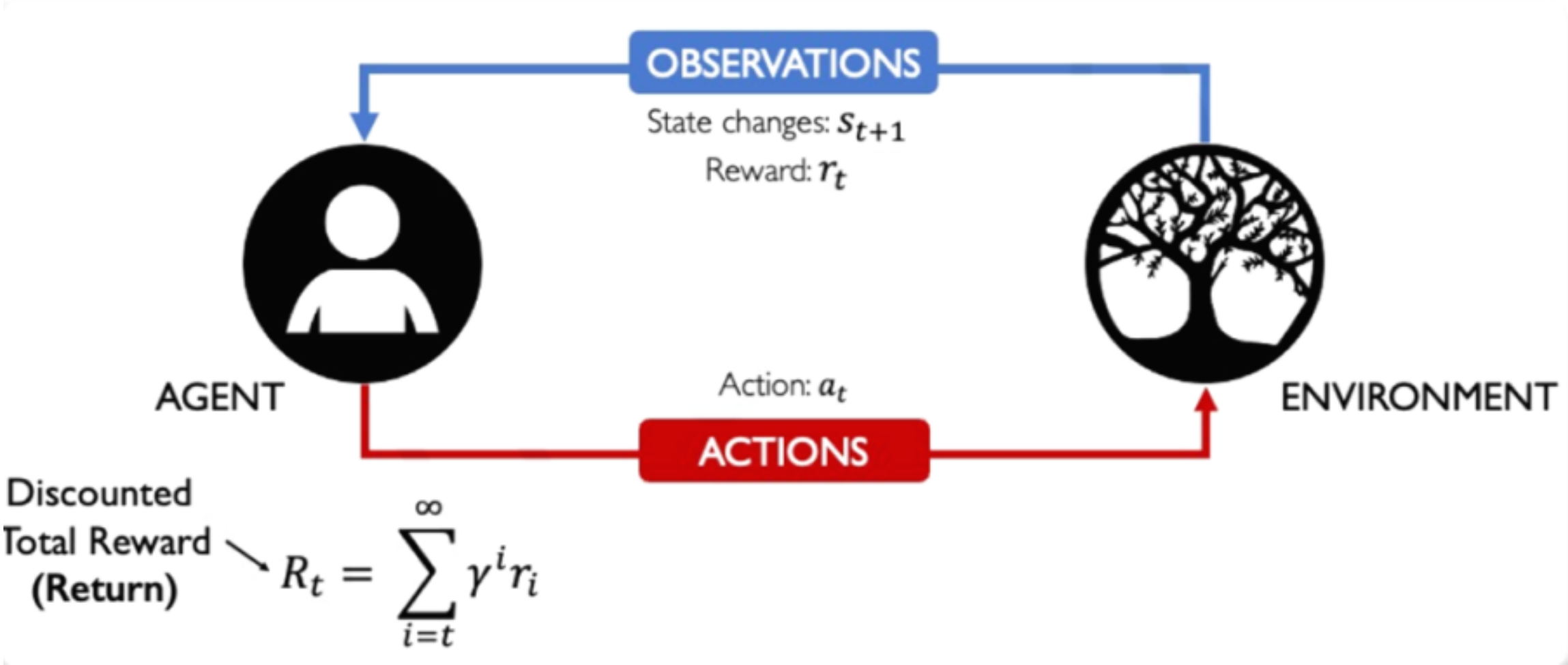
Reinforcement Learning: Concepts



Total Reward (Return)

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

Reinforcement Learning: Concepts



Discounted Total Reward (Return)

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

Reinforcement Learning: Concepts

Discounted Total Reward (Return)

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

γ: discount factor; $0 < \gamma < 1$

Deep Reinforcement Learning Algorithms

Value Learning

Find $Q(s, a)$

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

Policy Learning

Find $\pi(s)$

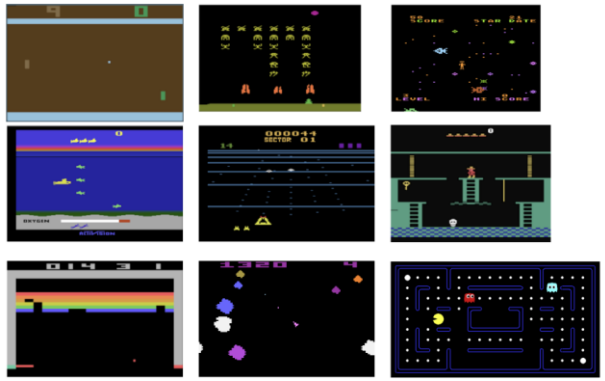
Sample $a \sim \pi(s)$

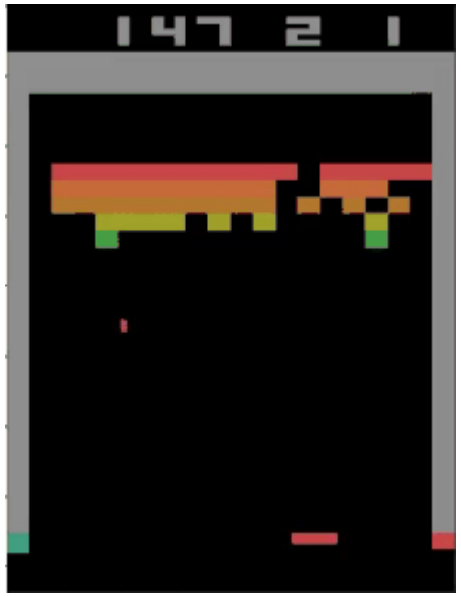
Why Deep Reinforcement Learning?

Reinforcement learning can be used to solve large problems, e.g.

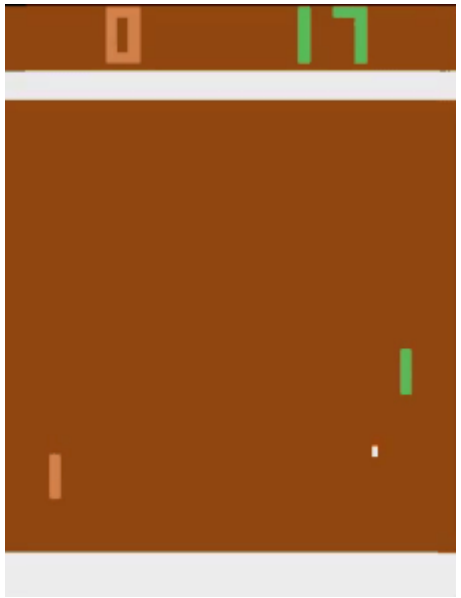
- Backgammon: 10^{20} states
- Go: 10^{170} states
- Helicopter: continuous state space
- Robots: real world

Atari Games

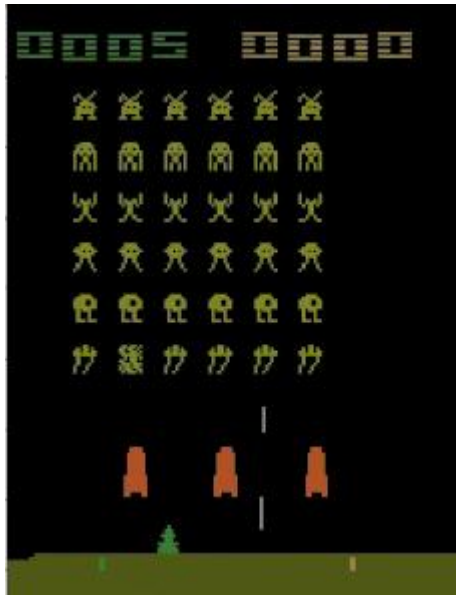




Breakout-v4



Pong-v4



SpaceInvenders-v4

Deep Reinforcement Learning Algorithms

Value Learning

Find $Q(s, a)$

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

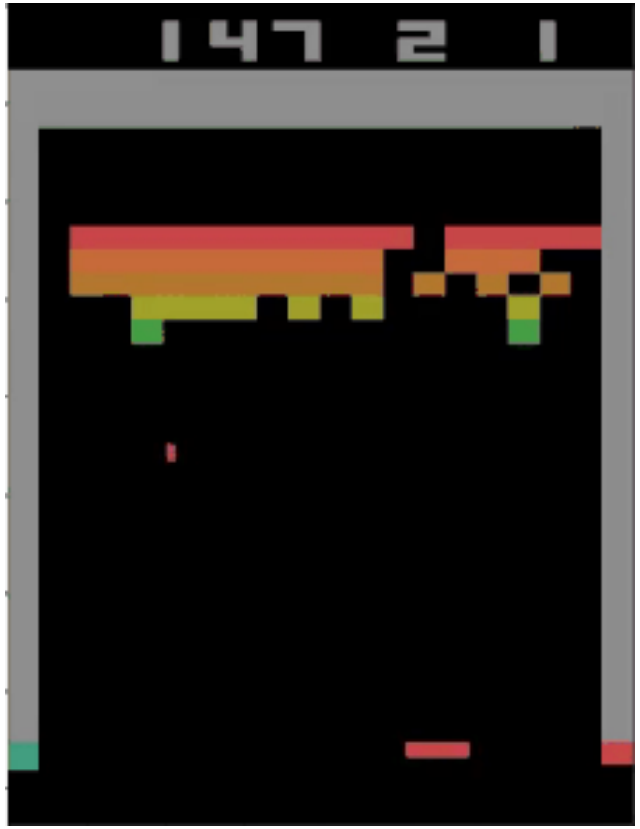
Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

Deep Reinforcement Learning Algorithms

Atari Breakout



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

(state, action)

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

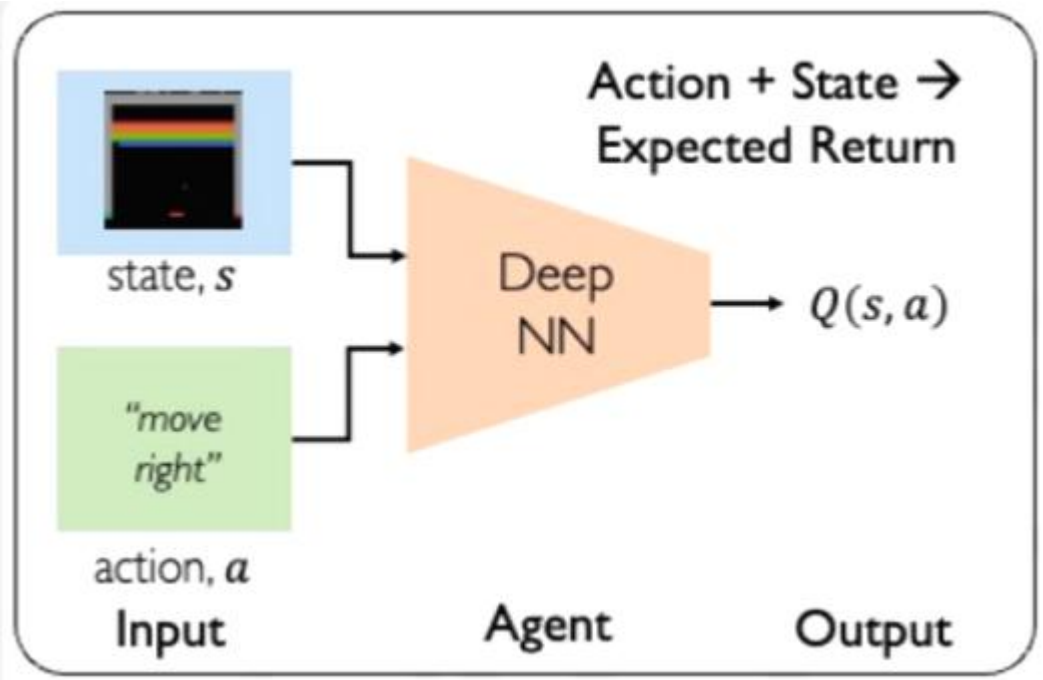
$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

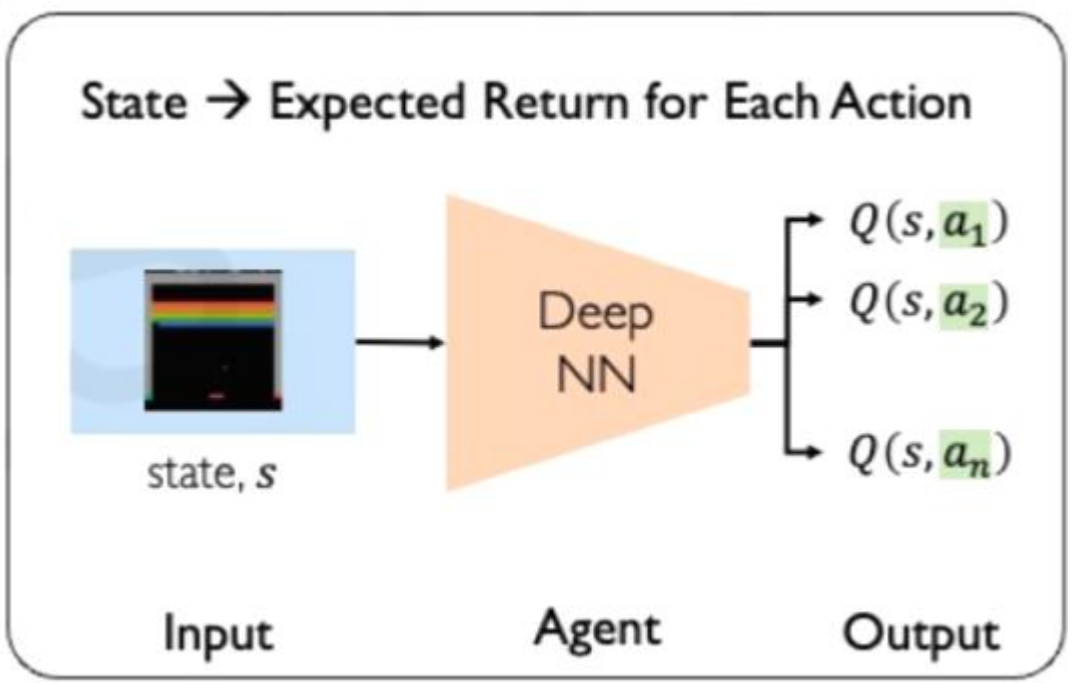
Solution: Non Linear Function approximation

$$Q_w(s, a) \approx Q_\pi(s, a)$$

Deep Q Networks (DQN): Mnih et al. (2013/2015)

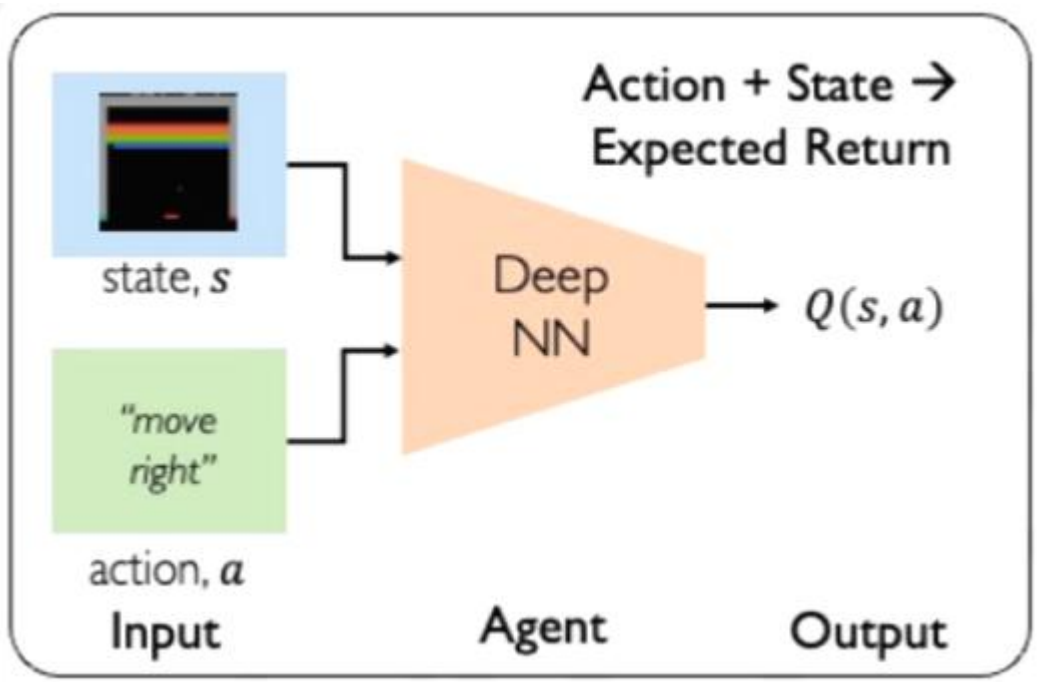
How can we use deep neural networks to model Q-functions?

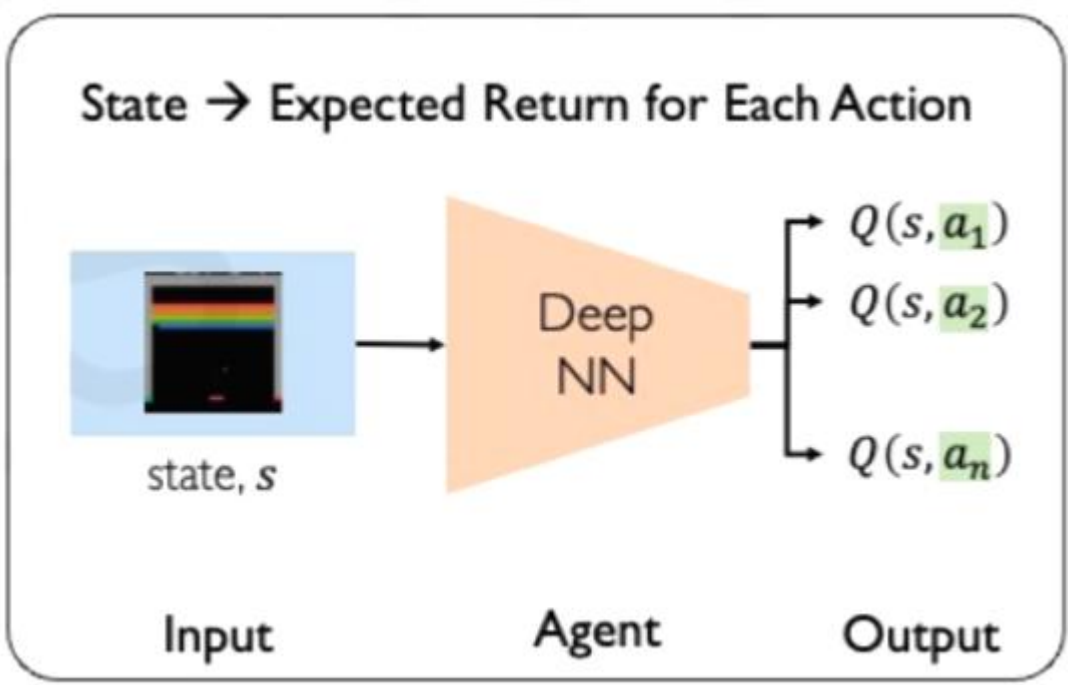




Deep Q Networks (DQN): Training

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



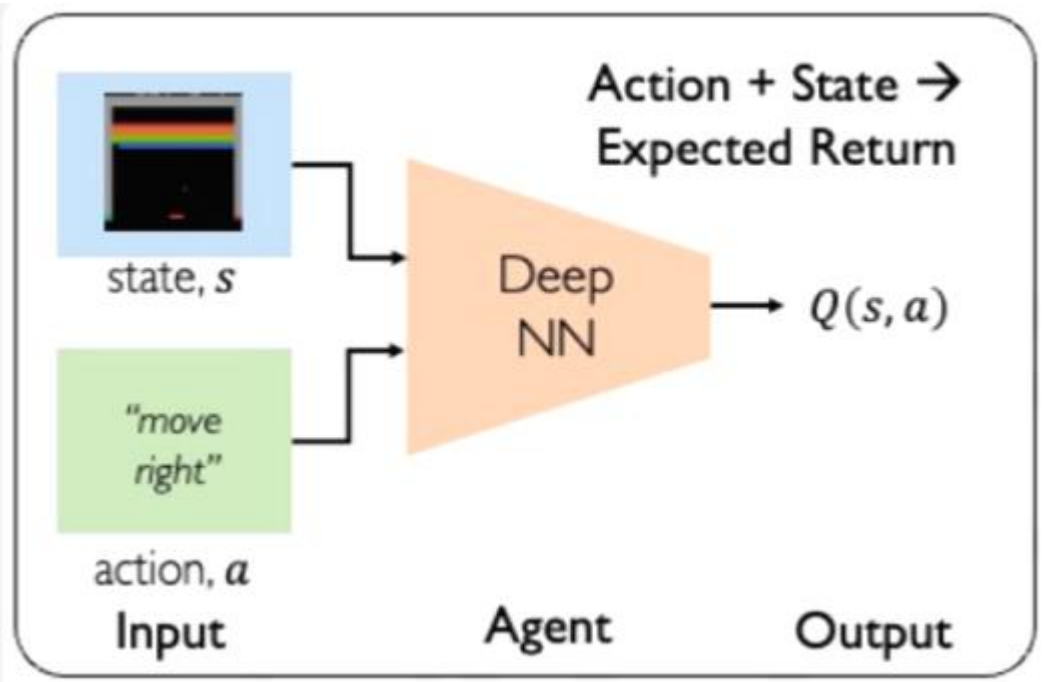


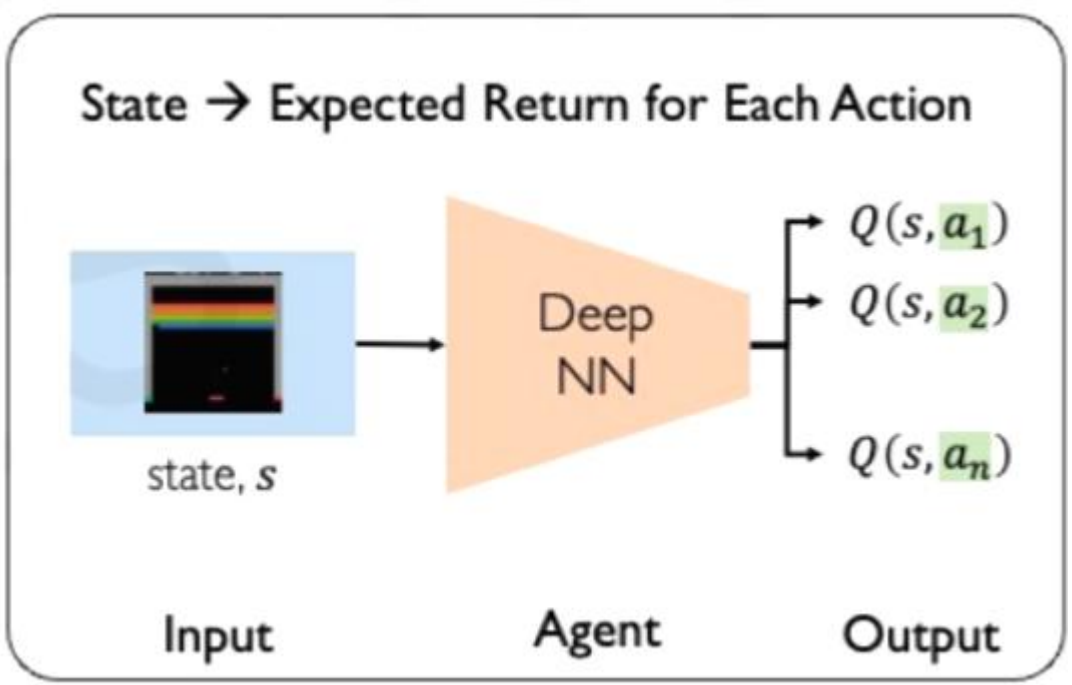
$$\overbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}^{\text{target}}$$

Take all the best actions → target return

Deep Q Networks (DQN): Training

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

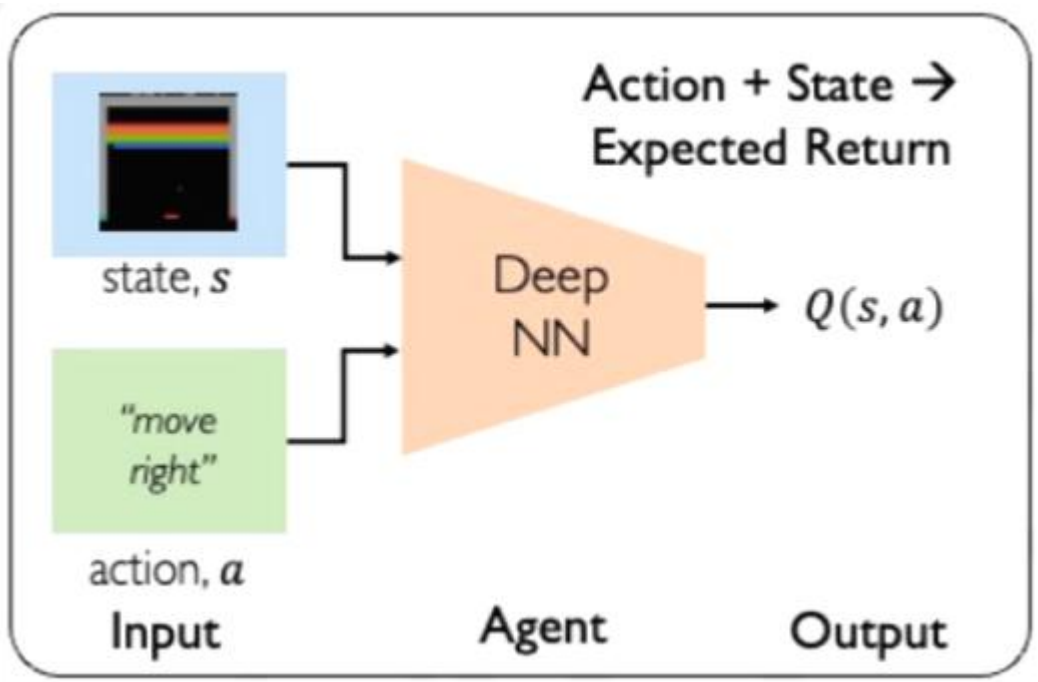


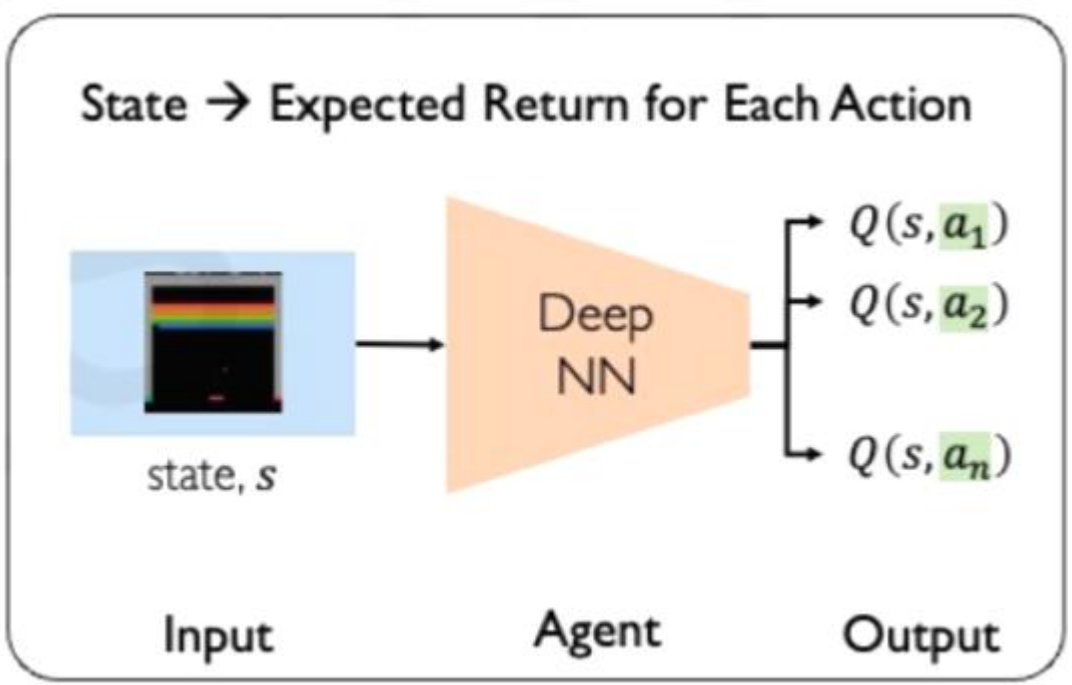


$$\underbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}_{\text{target}} \qquad \underbrace{Q(s, a)}_{\text{predicted}}$$

Deep Q Networks (DQN): Training

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

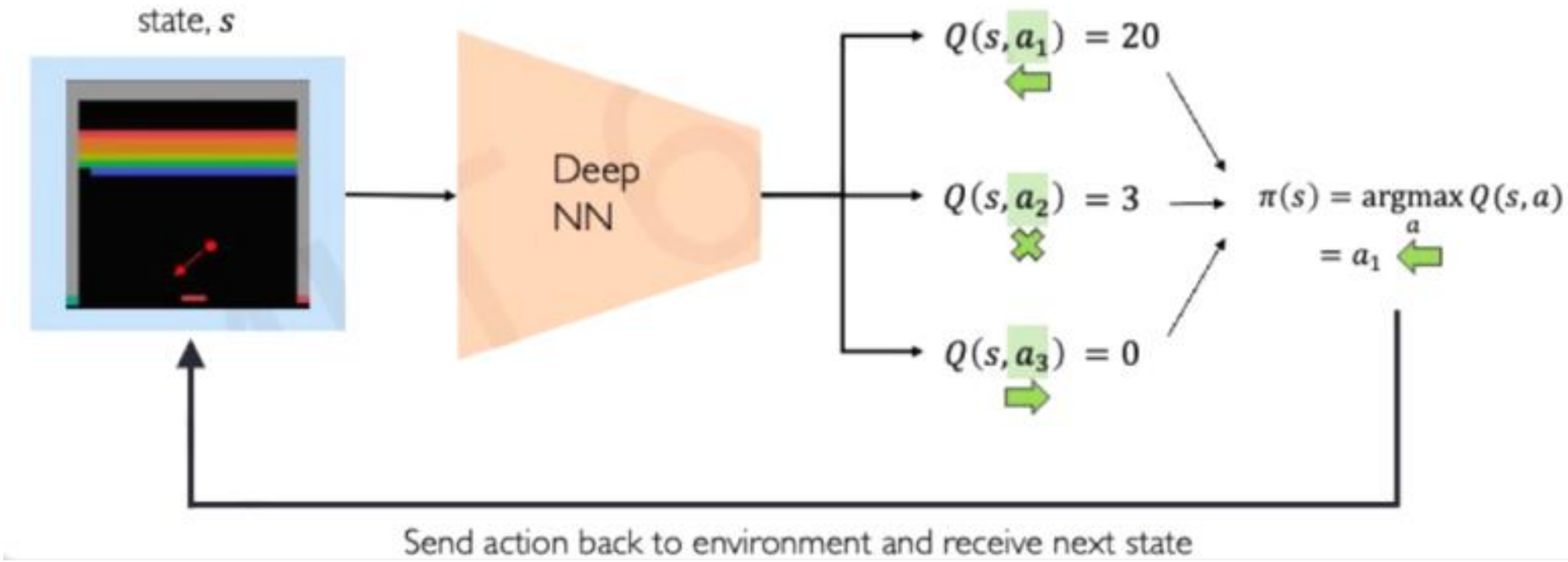




$$\mathcal{L} = \mathbb{E}\left[\left\|\overbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}^{\text{target}} - \overbrace{Q(s, a)}^{\text{predicted}}\right\|^2\right] \quad \text{Q-Loss}$$

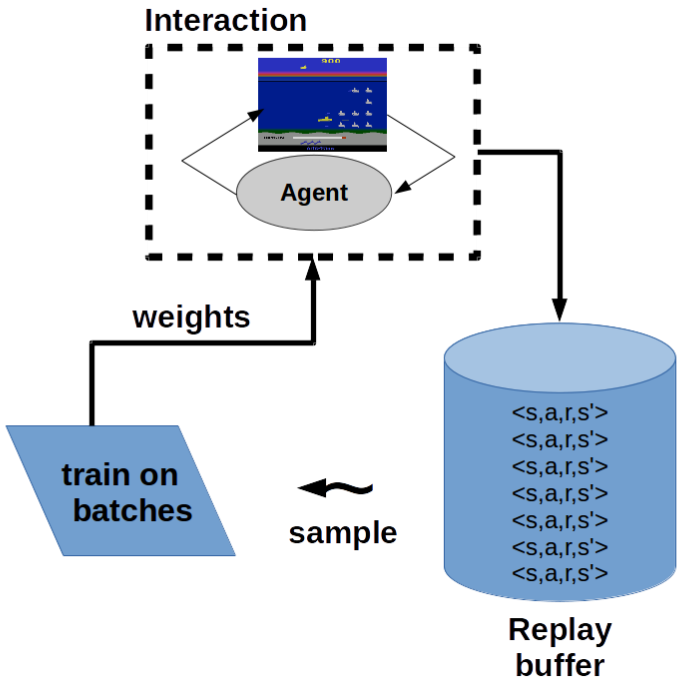
Deep Q Network Summary

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



Deep Q Network with Experiance Replay

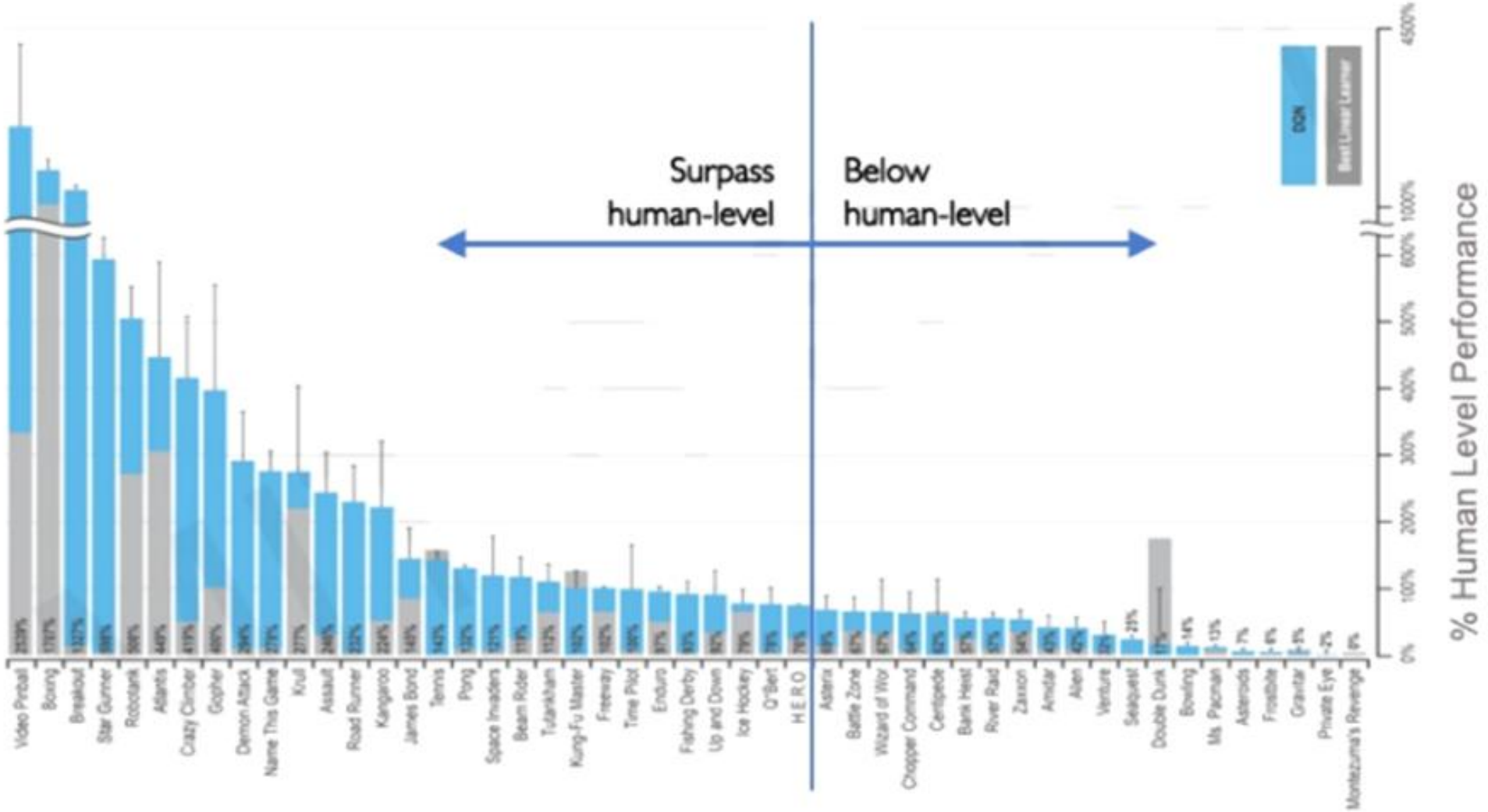
There's a powerful technique that you can use to improve sample efficiency for off-policy algorithms: Experience replay



Training with experience replay

- Play game, sample <s,a,r,s'>.
- Update q-values based on <s,a,r,s'>.
- Store <s,a,r,s'> transition in a buffer.
- If buffer is full, delete earliest data.
- Sample K such transitions from that buffer and update q-values based on them.

DQN Atari Results



Downsides of Q-learning

Complexity:

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

Flexibility:

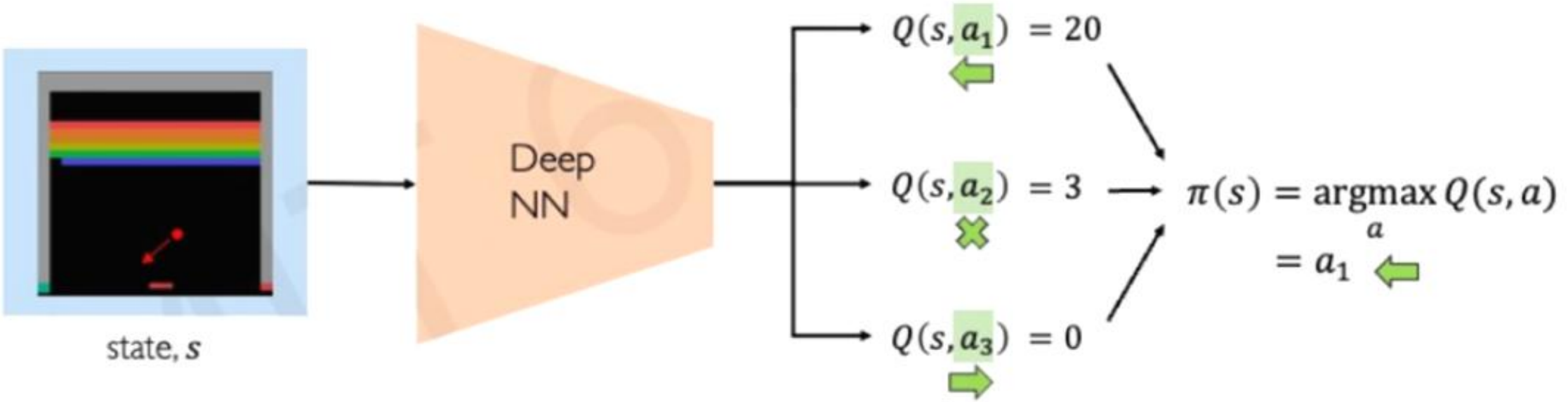
- Policy is deterministically computed from the Q function by maximizing the reward → cannot learn stochastic policies

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

Deep Reinforcement Learning Algorithms

Deep Q Networks (DQN)

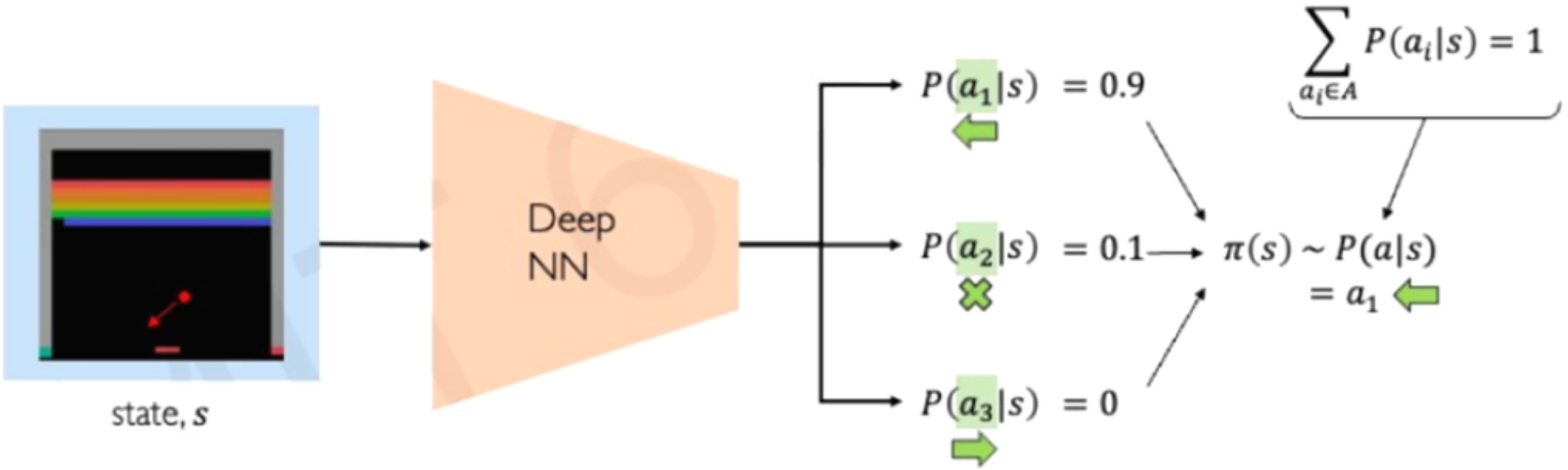
DQN: Approximate Q-function and use to infer the optimal policy, $\pi(s)$



Policy Gradient (PG): Key Idea

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

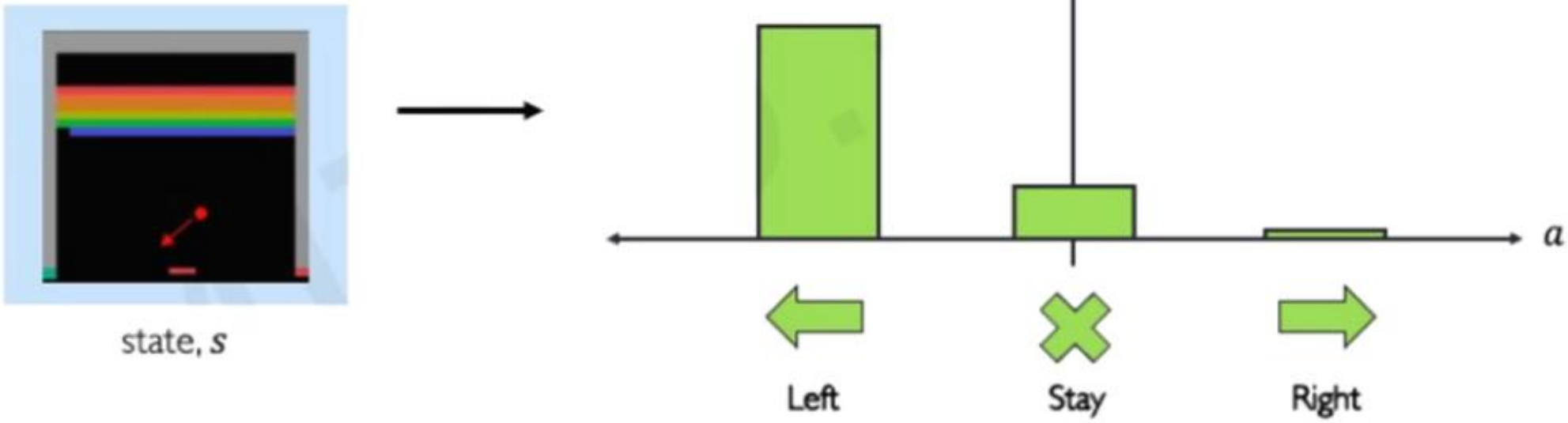
Policy Gradient: Directly optimize the policy $\pi(s)$



What are some advantages of this formulation?

Discrete vs Continuous Action Spaces

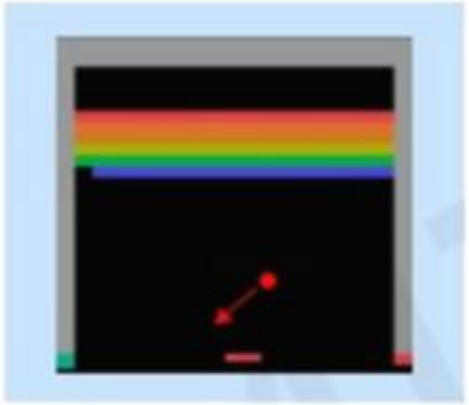
Discrete action space: which direction should I move?



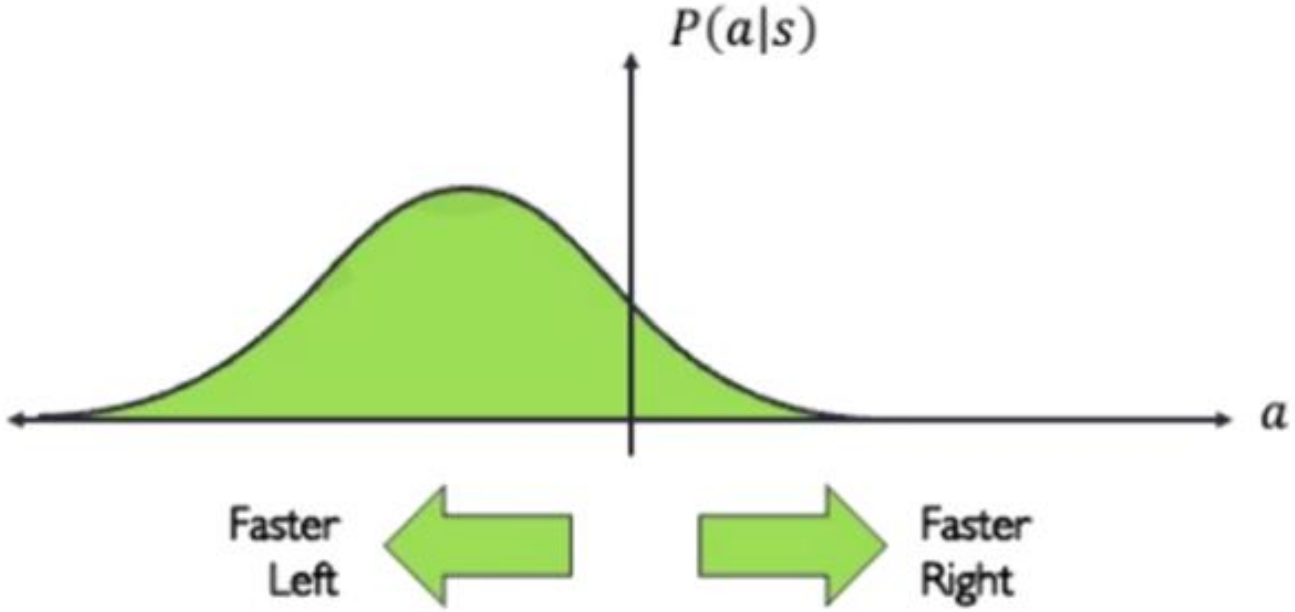
Discrete vs Continuous Action Spaces

Discrete action space: which direction should I move?

Continuous action space: how fast should I move?

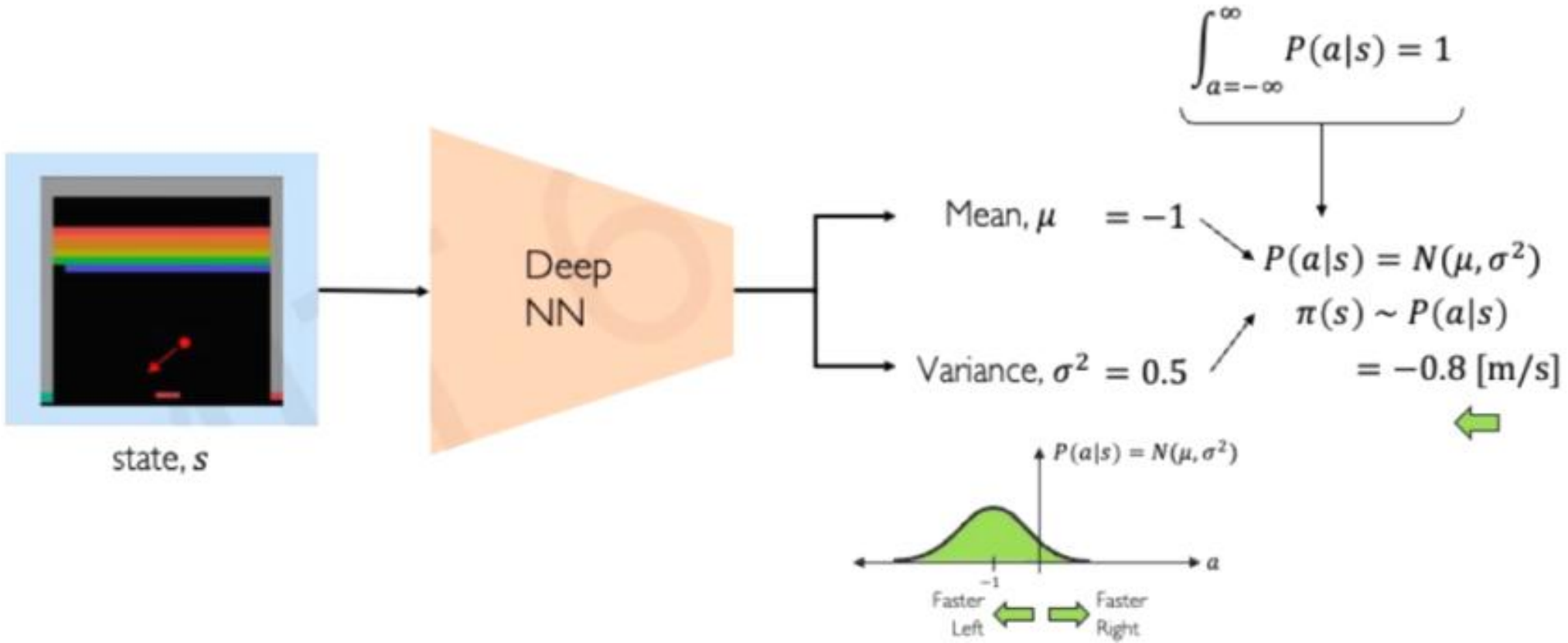


state, s



Policy Gradient (PG): Key Idea

Policy Gradient: Enables modeling of continuous action space



Training Policy Gradients: Case Study

Reinforcement Learning Loop:

Case Study – Self-Driving Cars

Agent: vehicle

State: camera, lidar, etc

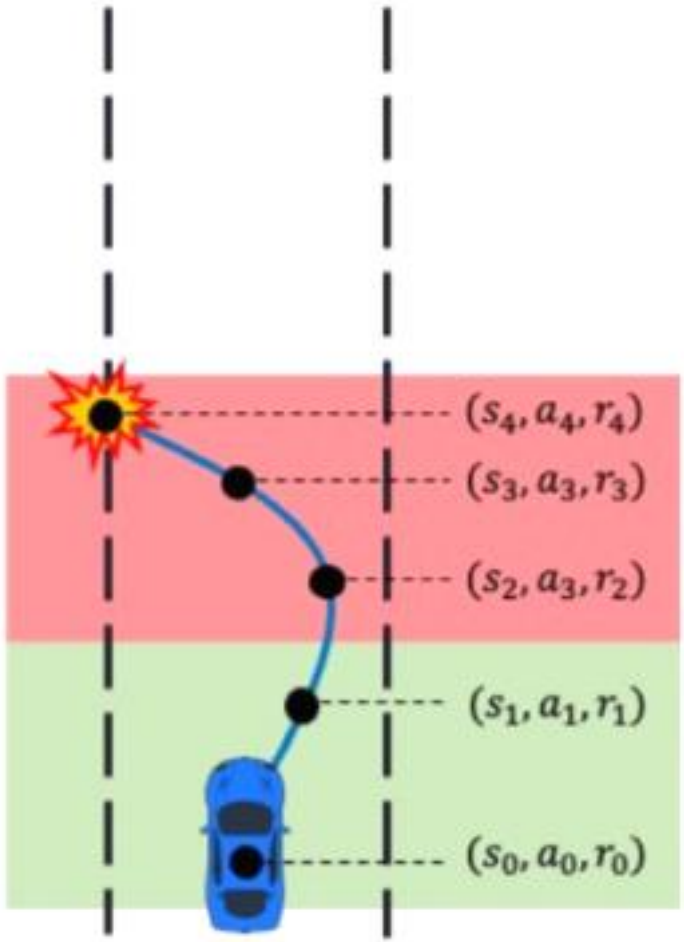
Action: steering wheel angle

Reward: distance traveled

Training Policy Gradients

Training Algorithm

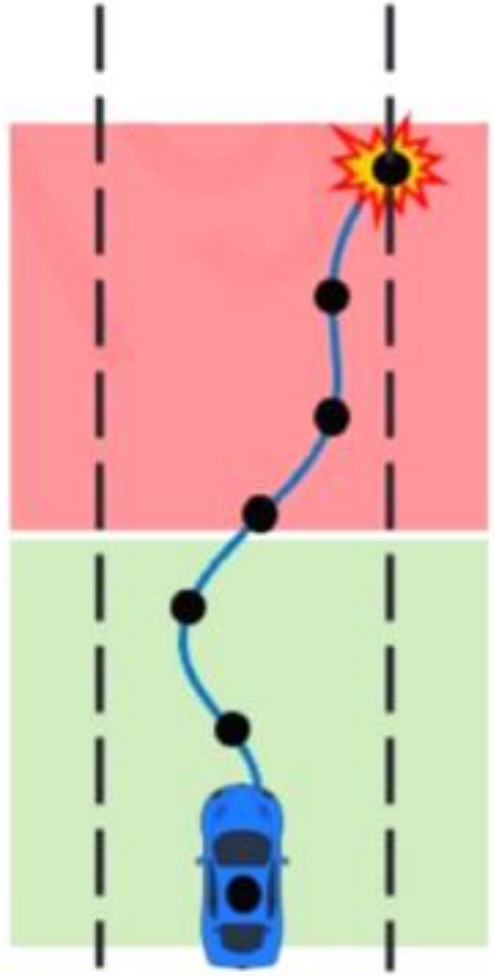
1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward



Training Policy Gradients

Training Algorithm

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



Training Policy Gradients

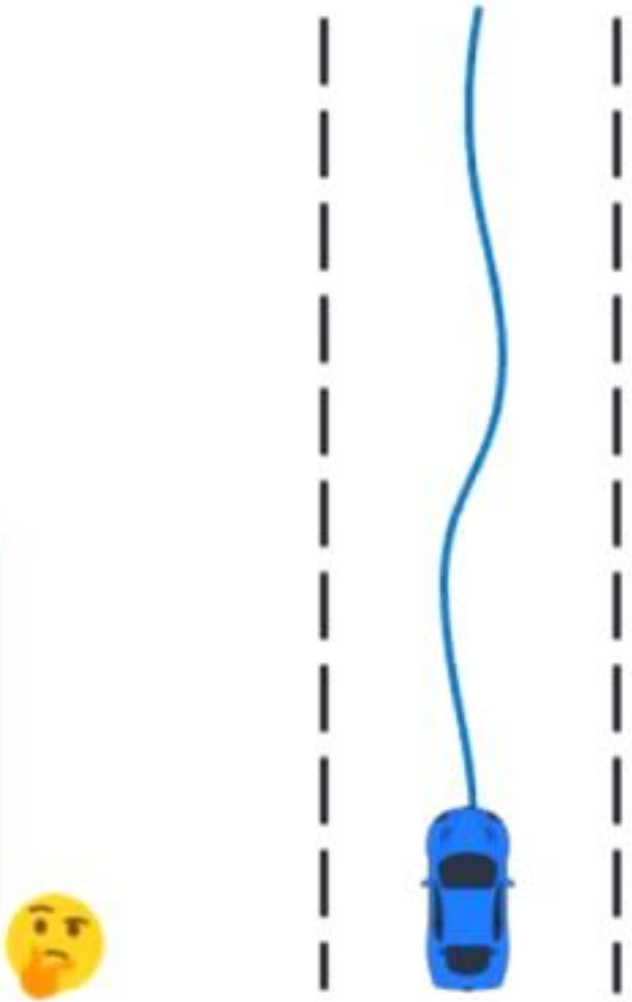
Training Algorithm

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

log-likelihood of action

$$\textbf{loss} = -\log \mathrm{P}(a_t|s_t)\, R_t$$

reward

Gradient descent update:

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

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

Policy gradient!

Policy Gradient Theorem

- The policy gradient approach also applies to (multi-step) MDPs
- Replaces reward R with long-term return G_t or value $q_\pi(s, a)$
- There are actually two policy gradient theorems (Sutton et al., 2000):

average return per episode & average reward per step

Policy Gradient Theorem(Episodic)

Theorem

For any differentiable policy $\pi_{\boldsymbol{\theta}}(s, a)$, let d_0 be the starting distribution over states in which we begin an episode. Then, the policy gradient of $J(\boldsymbol{\theta}) = \mathbb{E}[G_0 \mid S_0 \sim d_0]$ is

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{T} \gamma^t q_{\pi_{\boldsymbol{\theta}}}(S_t, A_t) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t | S_t) \mid S_0 \sim d_0 \right]$$

where

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] \end{aligned}$$

Policy Gradient Theorem(Average Reward)

Theorem

For any differentiable policy $\pi_{\boldsymbol{\theta}}(s, a)$*, the policy gradient of* $J(\boldsymbol{\theta}) = \mathbb{E}[R \mid \pi]$ *is*

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi}[q_{\pi_{\boldsymbol{\theta}}}(S_t, A_t) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t | S_t)]$$

Monte Carlo Policy Gradient (REINFORCE)

Algorithm 1 Episodic REINFORCE

Initialize policy network π_θ
for *iteration* $= 1, 2, \ldots,$ *num_episodes* **do**
    Generate an episode $S_0, A_0, R_1, S_1, \ldots, S_{T-1}, A_{T-1}, R_t$ following π_θ
    for $t = 0, 1, 2, \ldots, T-1$ **do**
        $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    end for
    $L(\theta) = -\frac{1}{T} \sum_{t=0}^{T-1} G_t \ln \pi_\theta(A_t|S_t)$
    Update π_θ using $Adam(\nabla_\theta L(\theta))$
end for

Monte Carlo Policy Gradient (REINFORCE) with baseline

Algorithm 2 Episodic REINFORCE with baseline

Initialize policy network π_θ
Initialize baseline network b_w
for *iteration* $= 1, 2, \ldots,$ *num_episodes* **do**
  Generate an episode $S_0, A_0, R_1, S_1, \ldots, S_{T-1}, A_{T-1}, R_t$ following π_θ
  for $t = 0, 1, 2, \ldots, T-1$ **do**
    $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
  end for
  $L(\theta) = -\frac{1}{T} \sum_{t=0}^{T-1} (G_t - b_w(S_t)) \ln \pi_\theta(A_t|S_t)$
  $L(w) = -\frac{1}{T} \sum_{t=0}^{T-1} (G_t - b_w(S_t))^2$
  Update π_θ using $Adam(\nabla_\theta L(\theta))$
  Update b_w using $Adam(\nabla_w L(w))$
end for

Deep RL: Value and Policy Based Summary

Foundations

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

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