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

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

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?

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

(state, action)

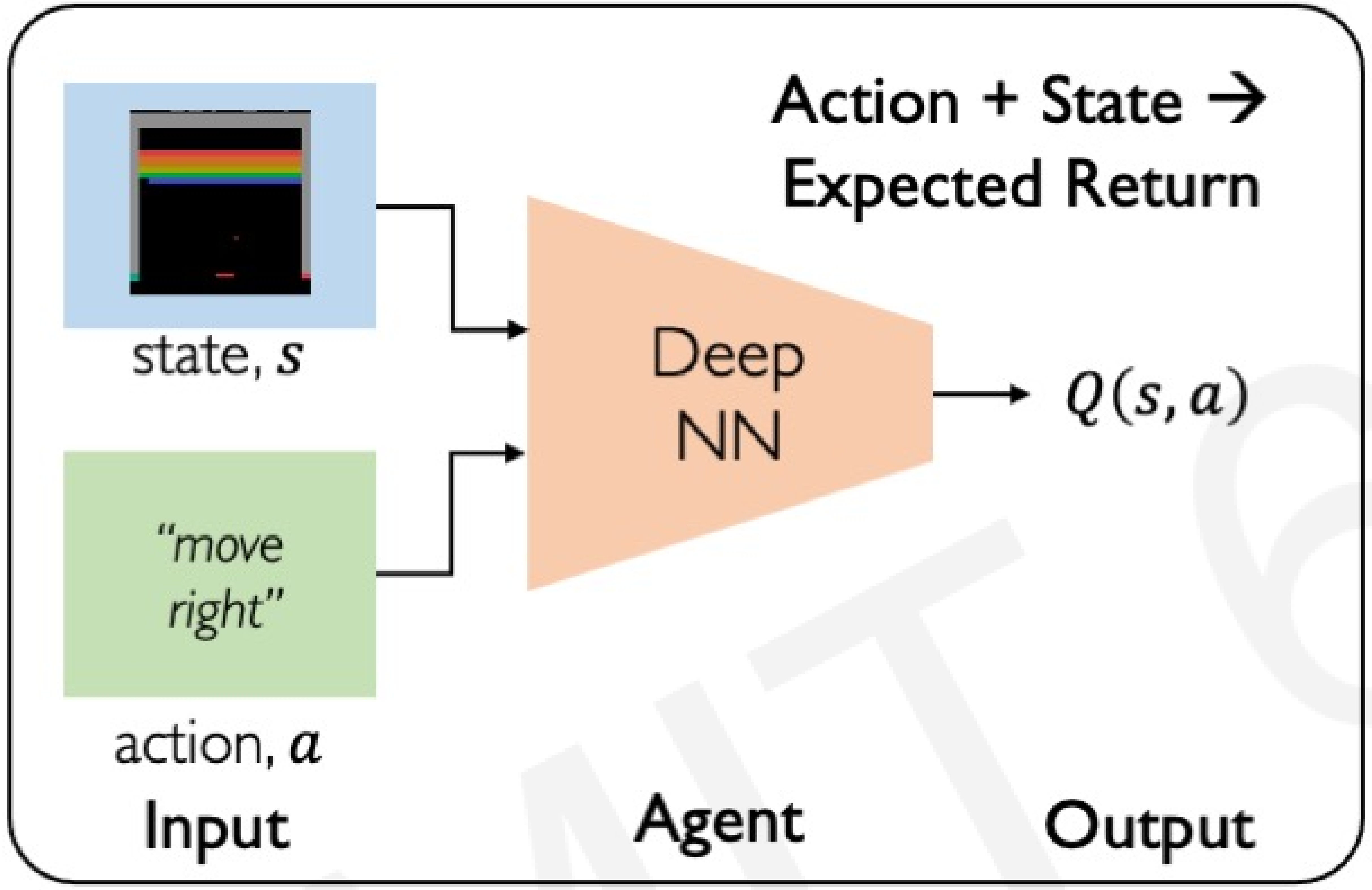
Ultimately, the agent needs a **policy $\boldsymbol{\pi(s)}$**, to infer the **best action to take** at its state, s

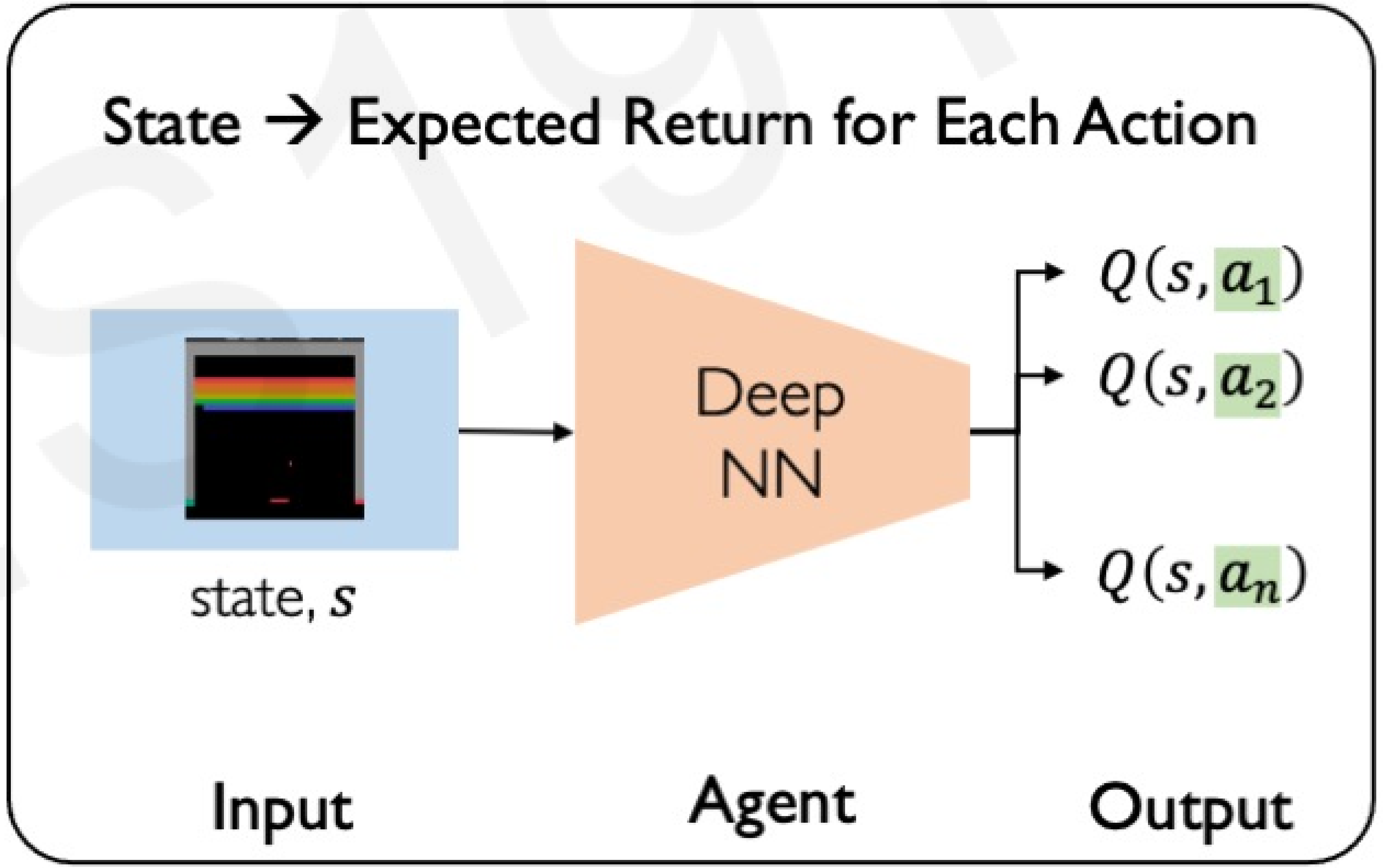
Strategy: the policy should choose an action that maximizes future reward

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

Deep Q Networks (DQN)

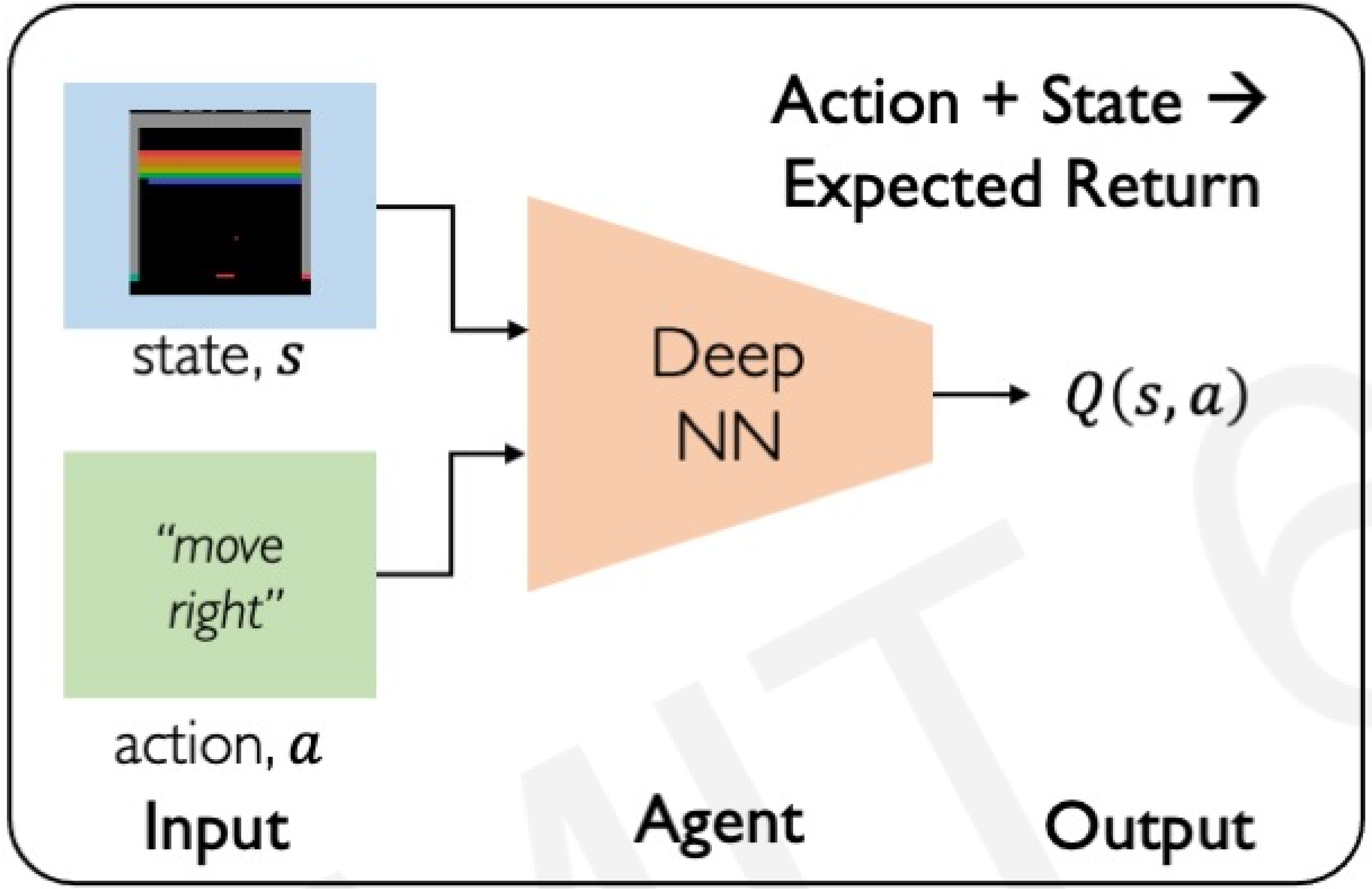
How can we use deep neural networks to model Q-functions?

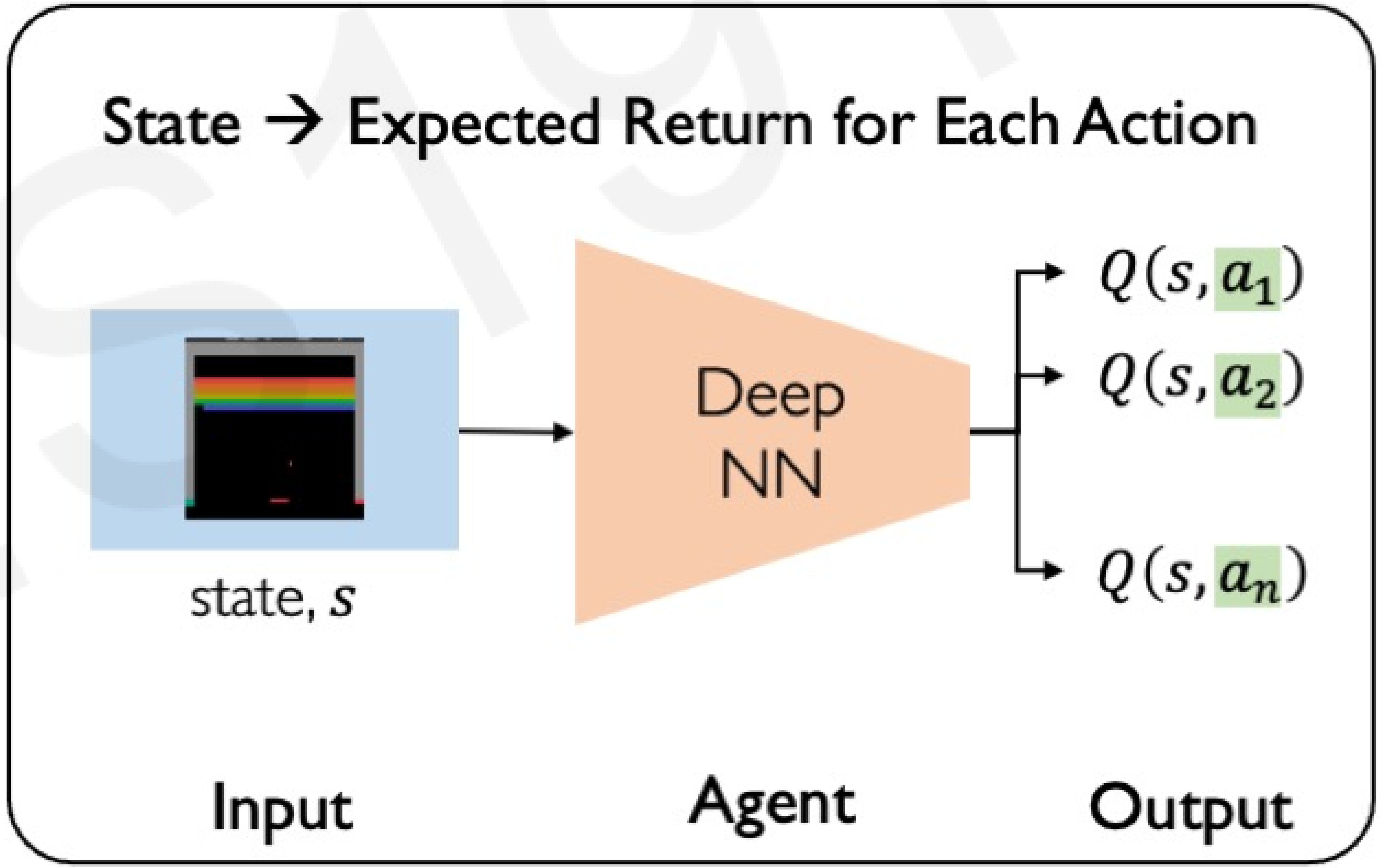




Deep Q Networks (DQN): Training

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

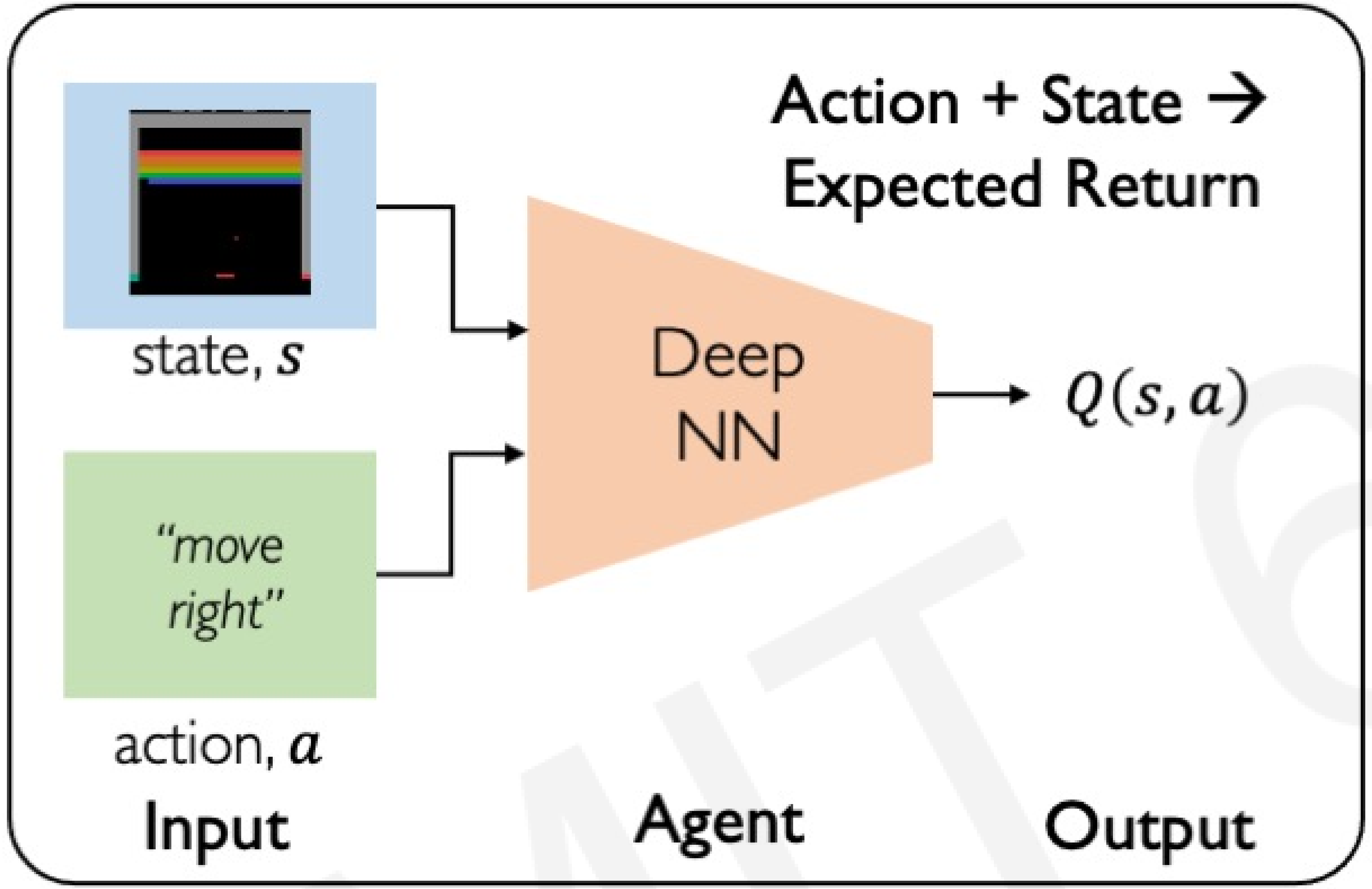


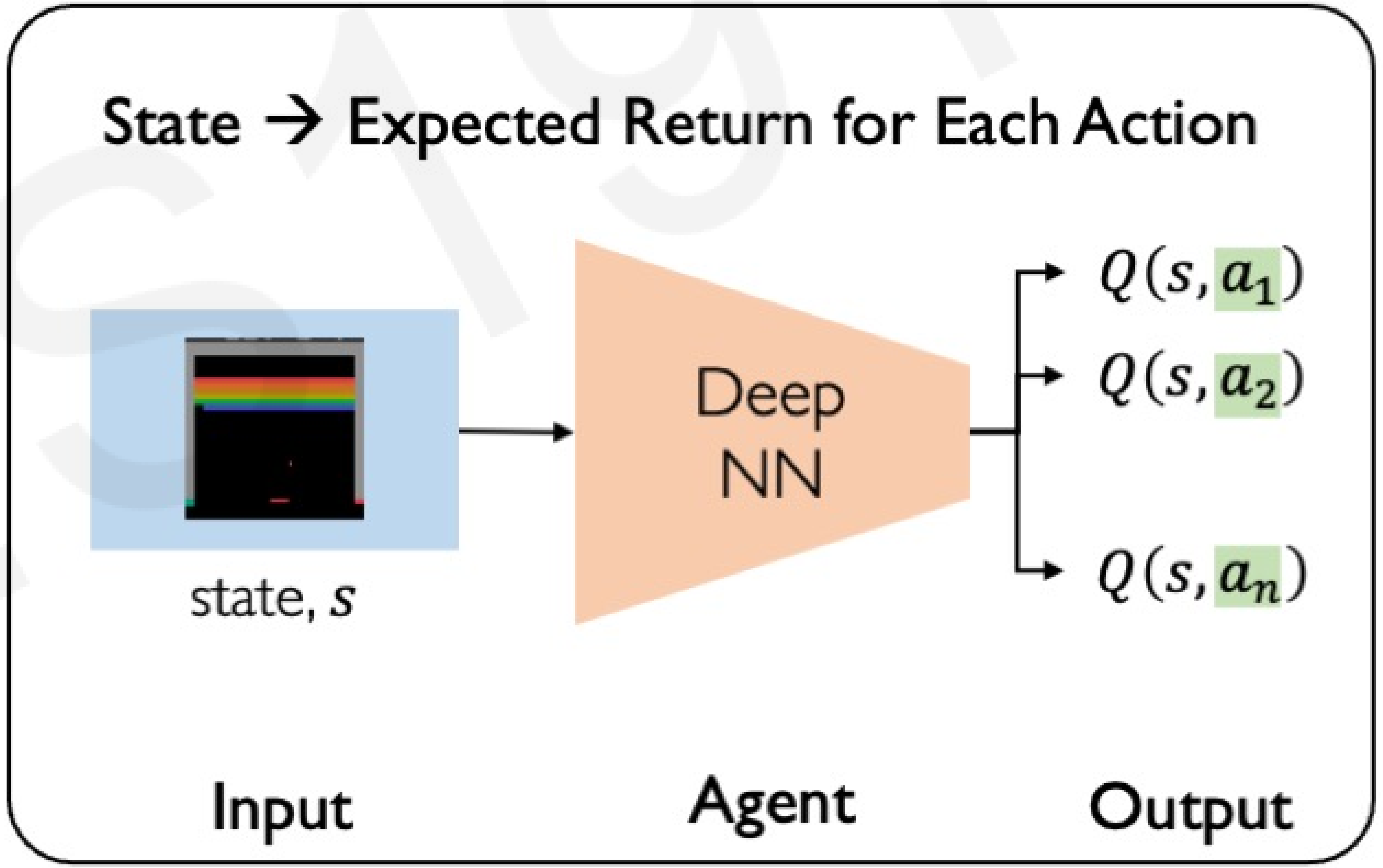


What happens if we take all the best actions?
Maximize target return → train the agent

Deep Q Networks (DQN): Training

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



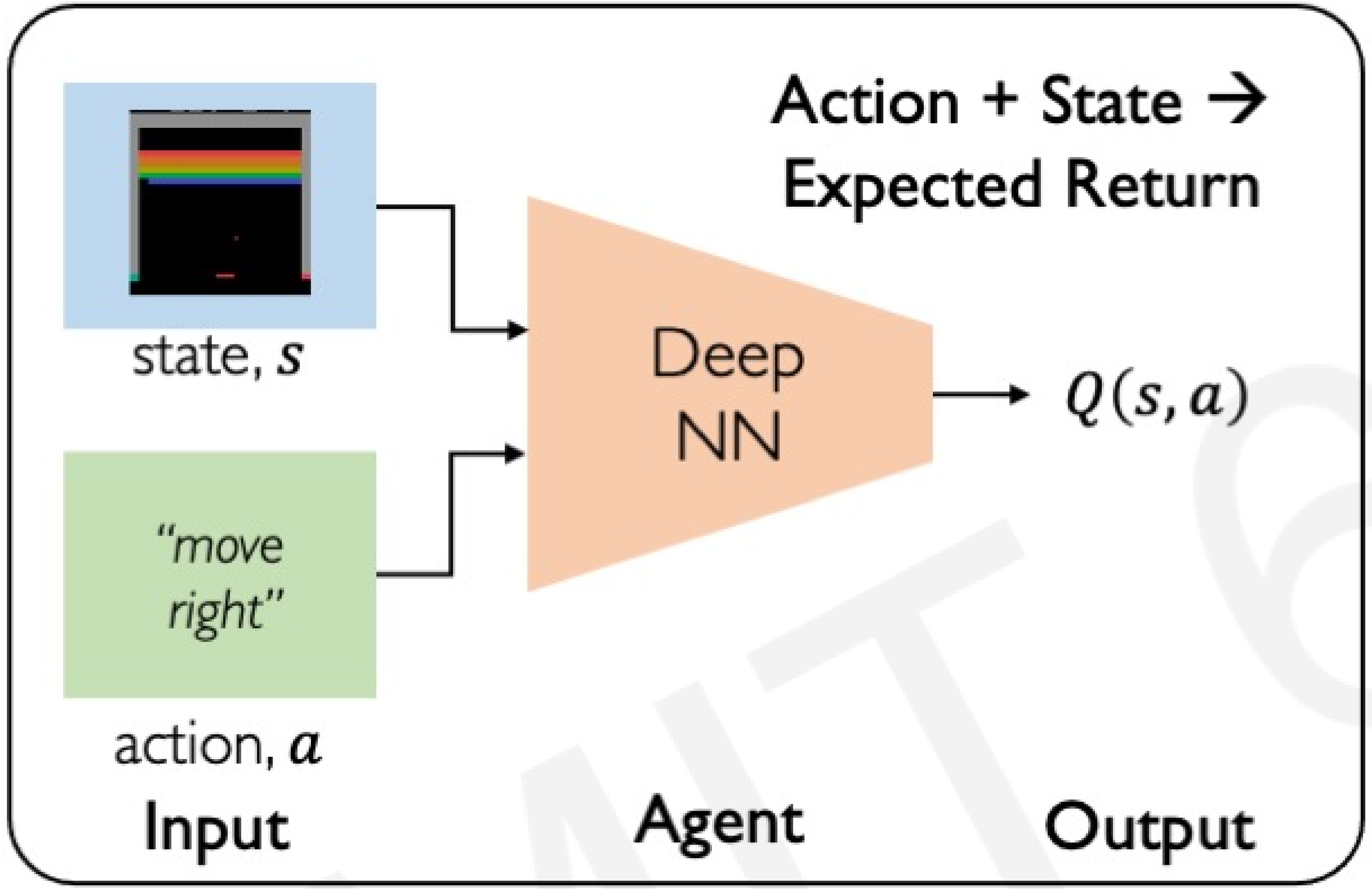


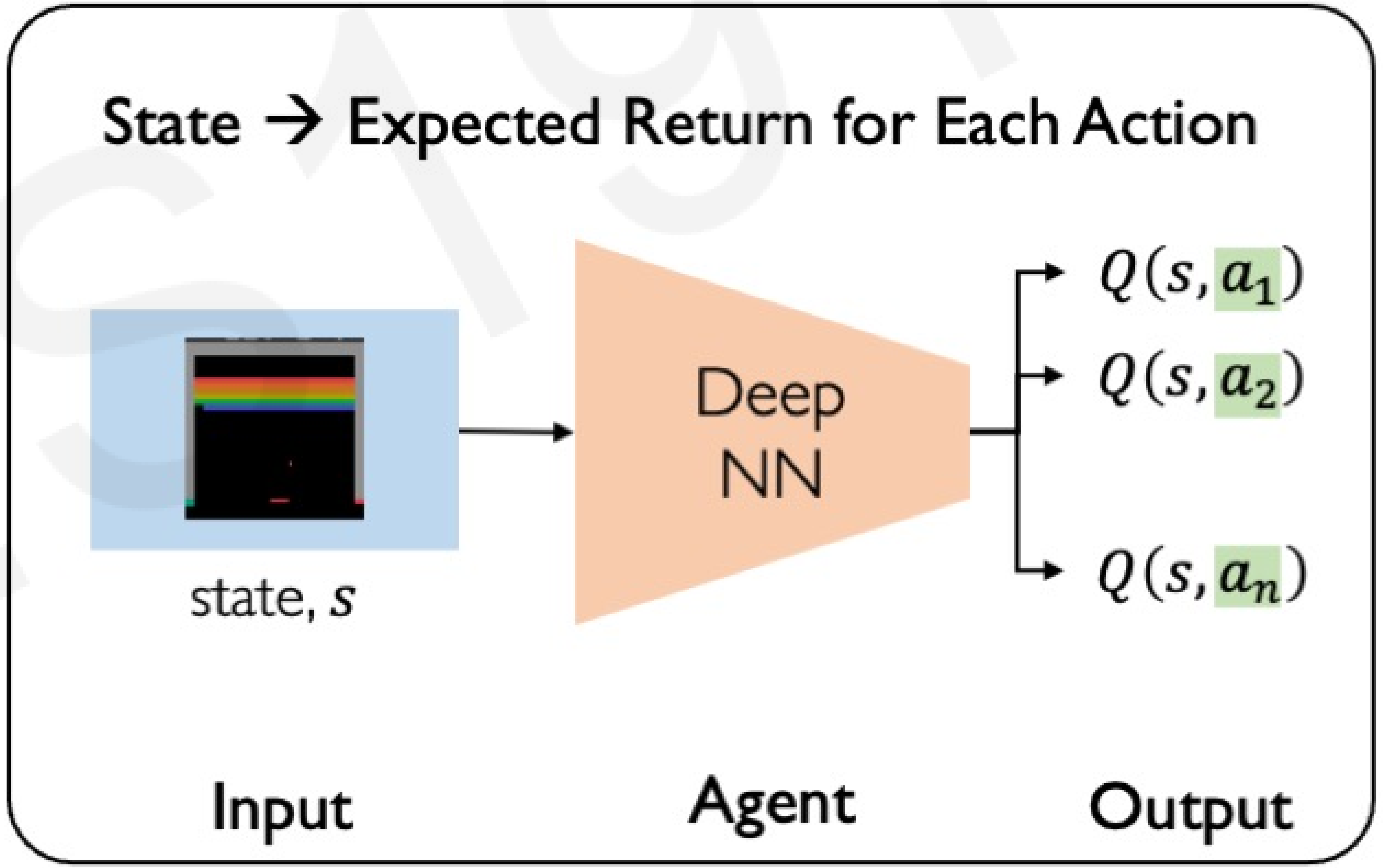
$$\overbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}^{\text{target}}$$

Take all the best actions → target return

Deep Q Networks (DQN): Training

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



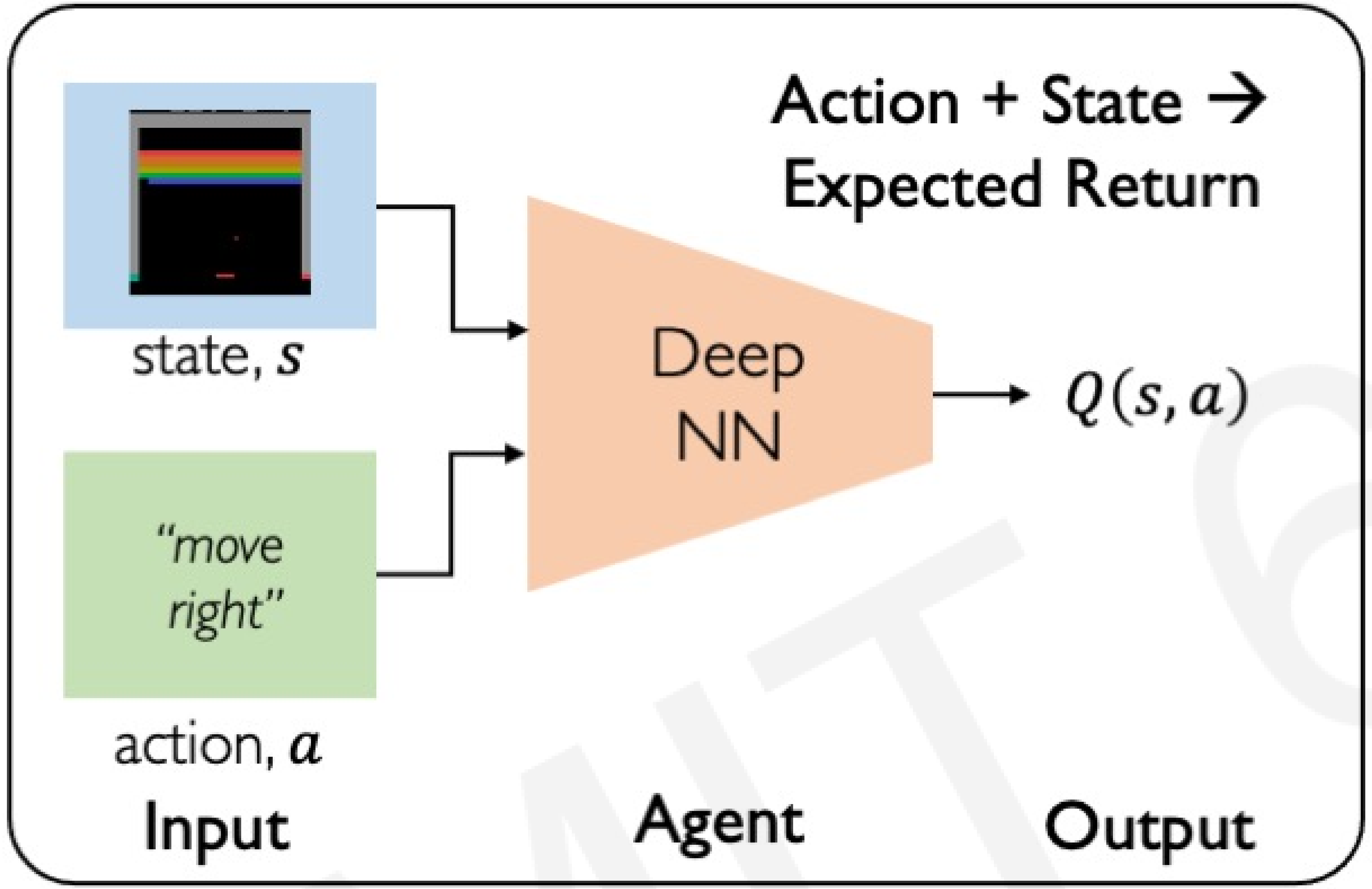


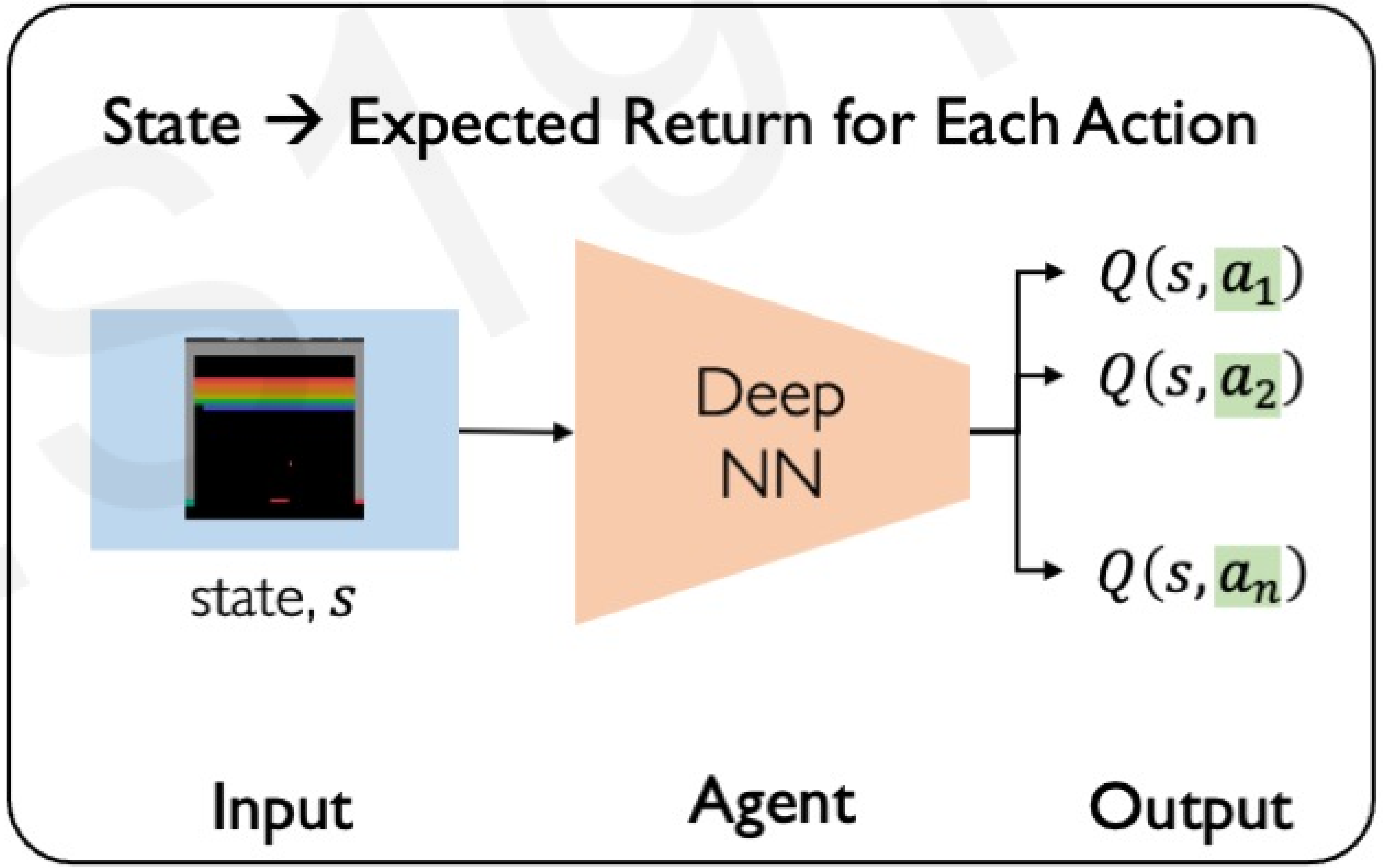
$$\underbrace{\left(r + \gamma \max_{a'} Q(s',a')\right)}_{\text{target}} \quad \underbrace{Q(s,a)}_{\text{predicted}}$$

Network prediction

Deep Q Networks (DQN): Training

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



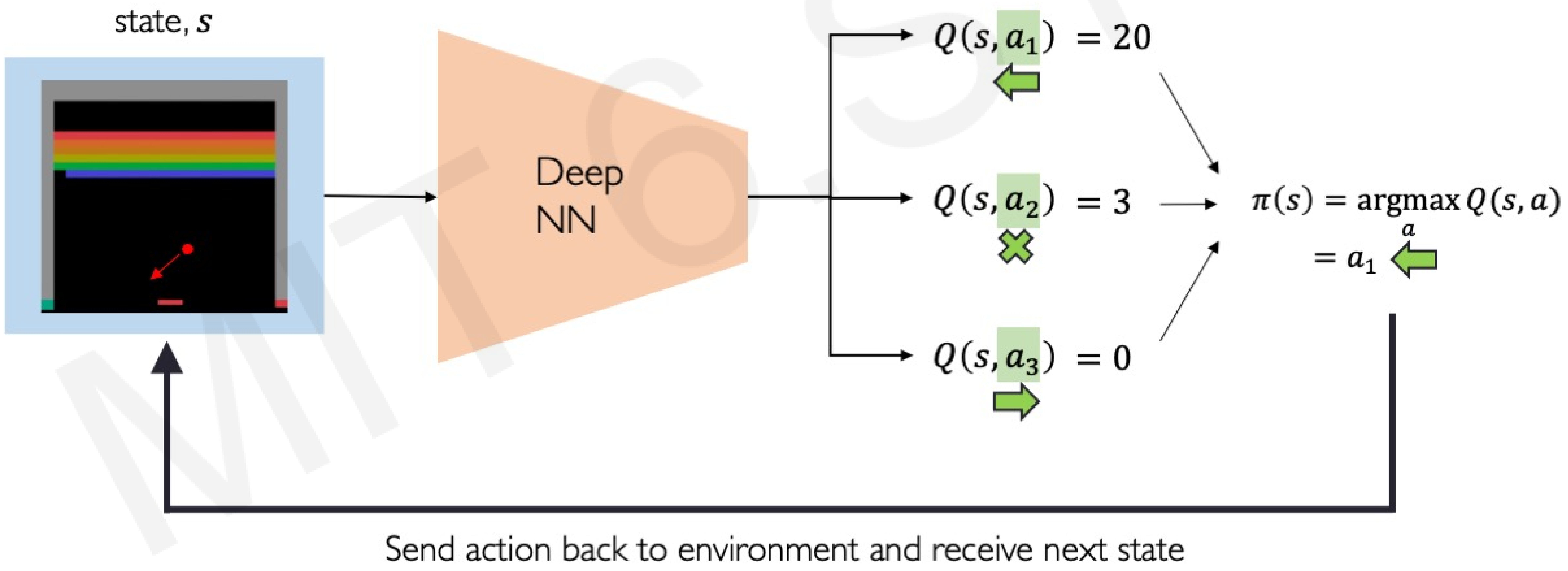


$$\mathcal{L} = \mathbb{E}\left[\left\| \overbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}^{\text{target}} - \overbrace{Q(s, a)}^{\text{predicted}} \right\|^2\right]$$

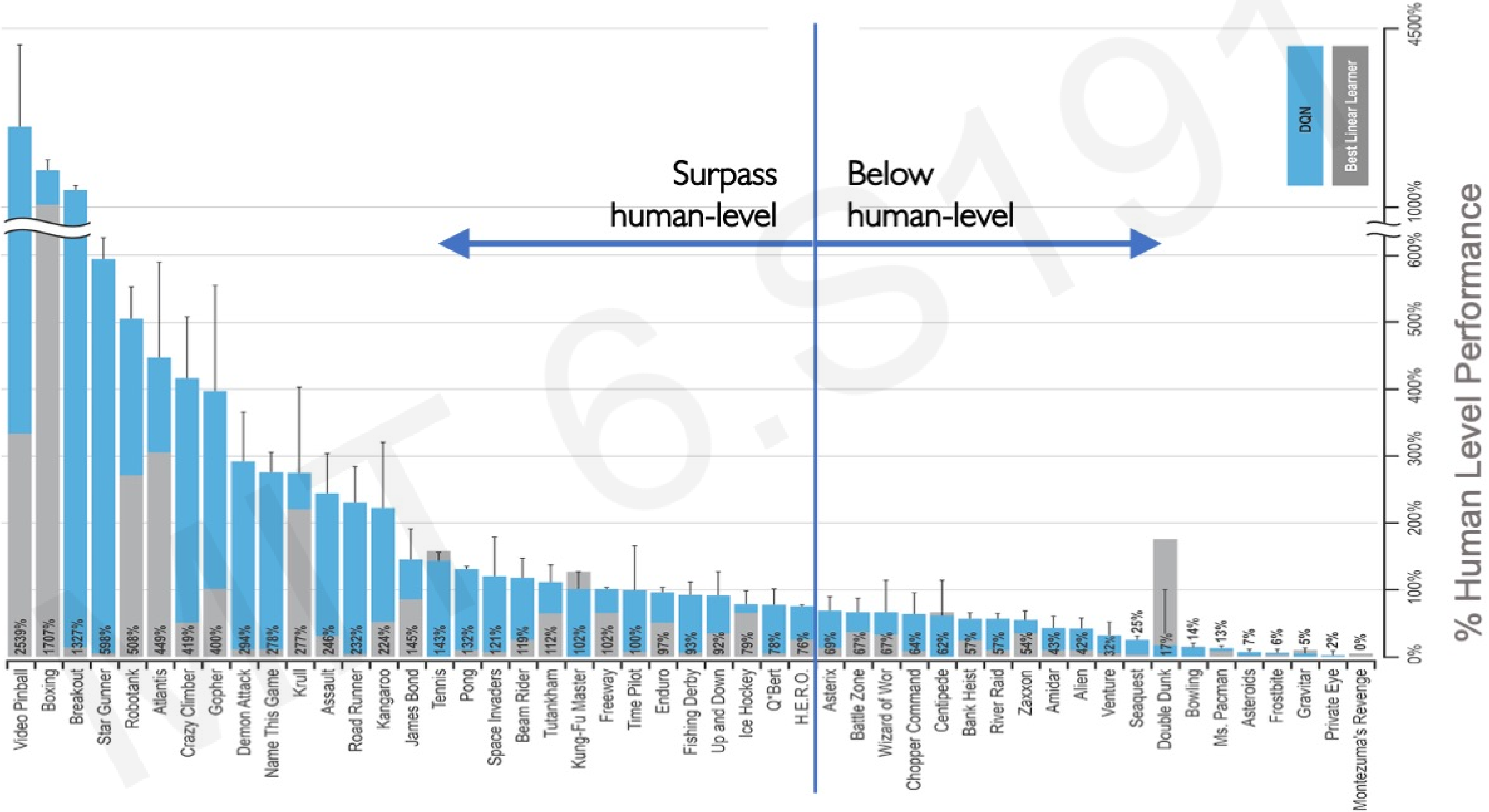
Q-Loss

Deep Q Network Summary

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



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

Value Learning

Find $Q(s,a)$

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

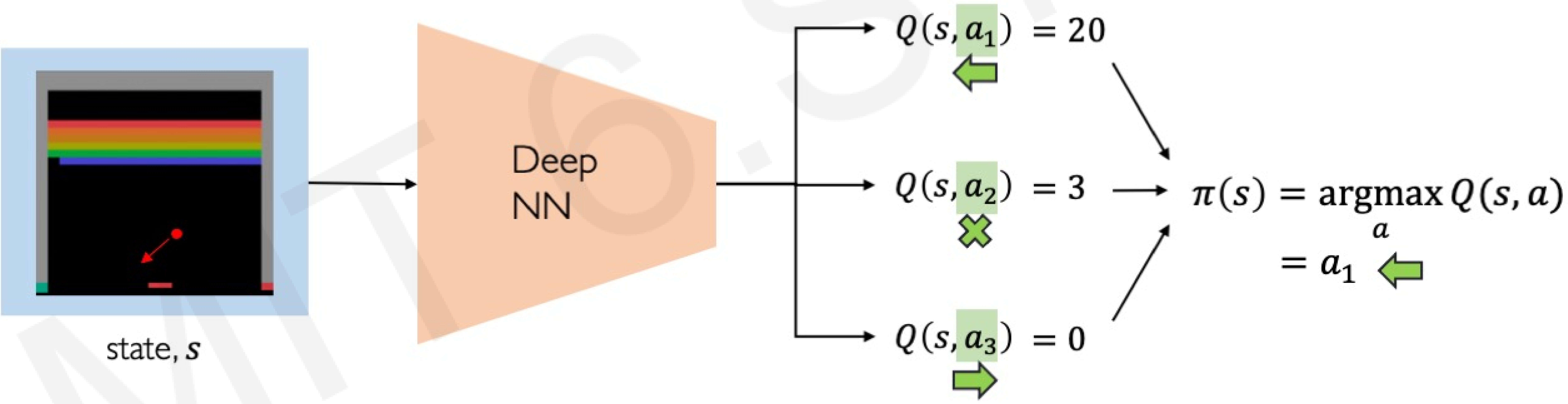
Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

Deep Q Networks (DQN)

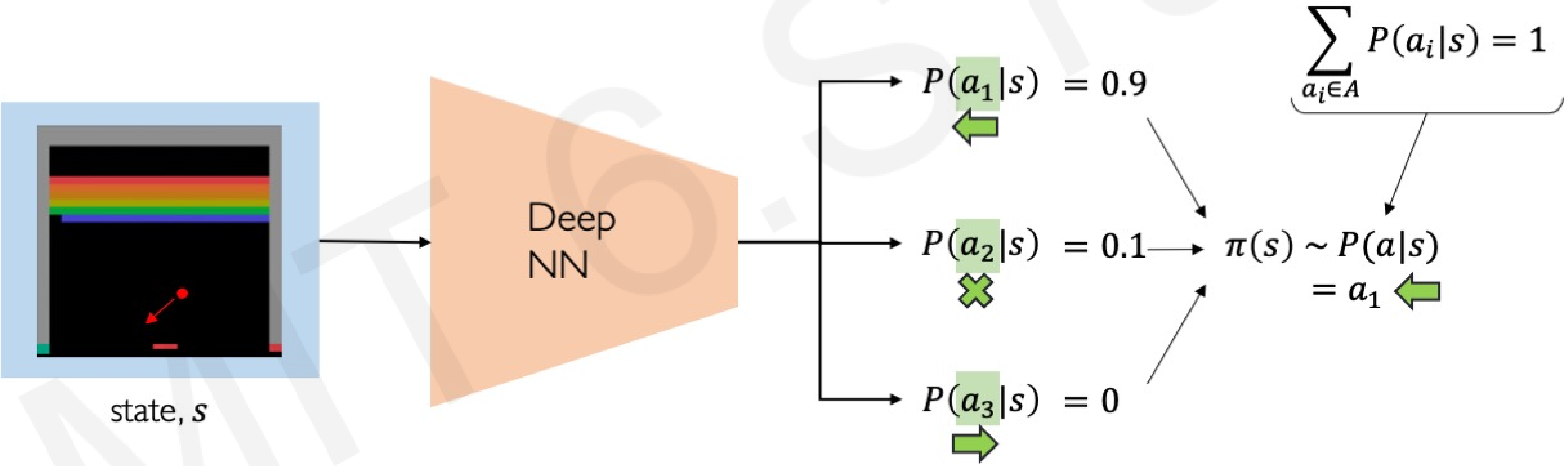
DQN: Approximate Q-function and use to infer the optimal policy, $\pi(s)$



Policy Gradient (PG): Key Idea

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

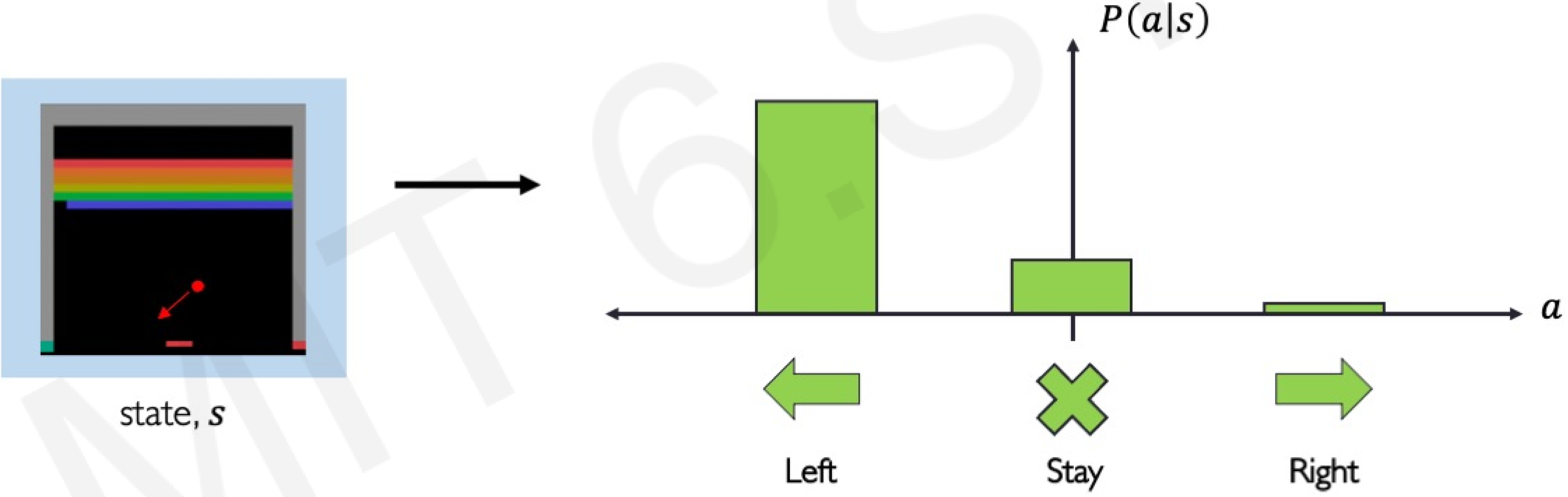
Policy Gradient: Directly optimize the policy $\pi(s)$



🤔 What are some advantages of this formulation?

Discrete vs Continuous Action Spaces

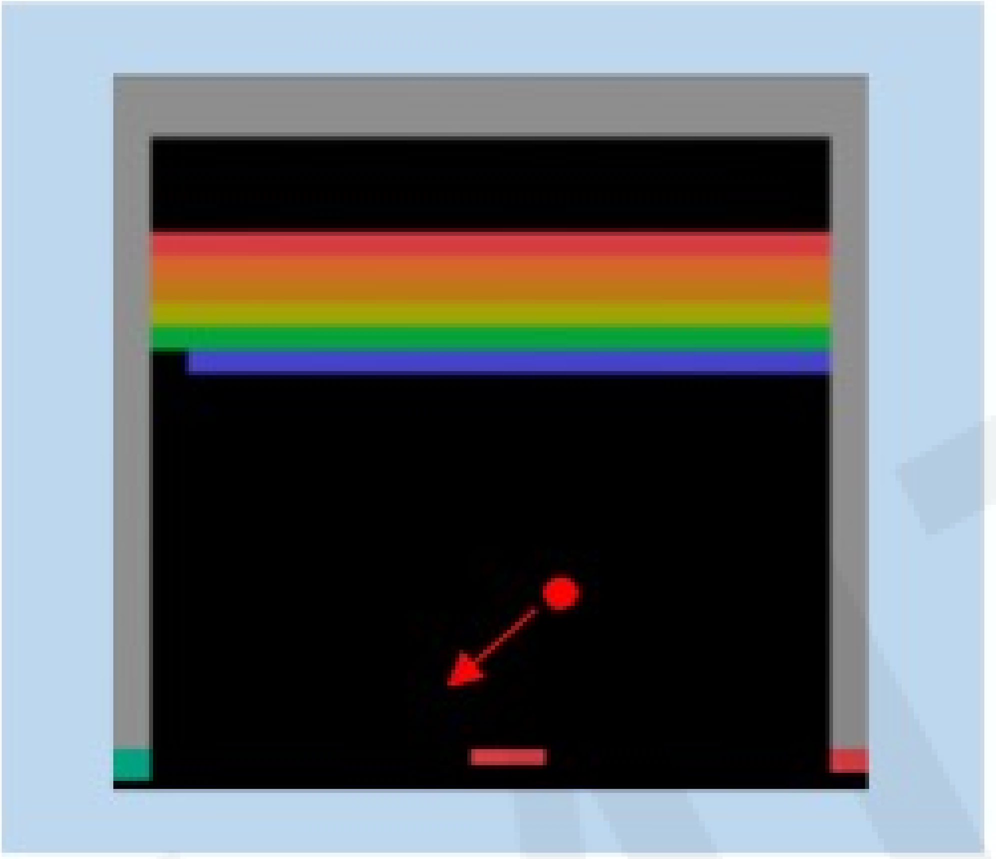
Discrete action space: which direction should I move? 



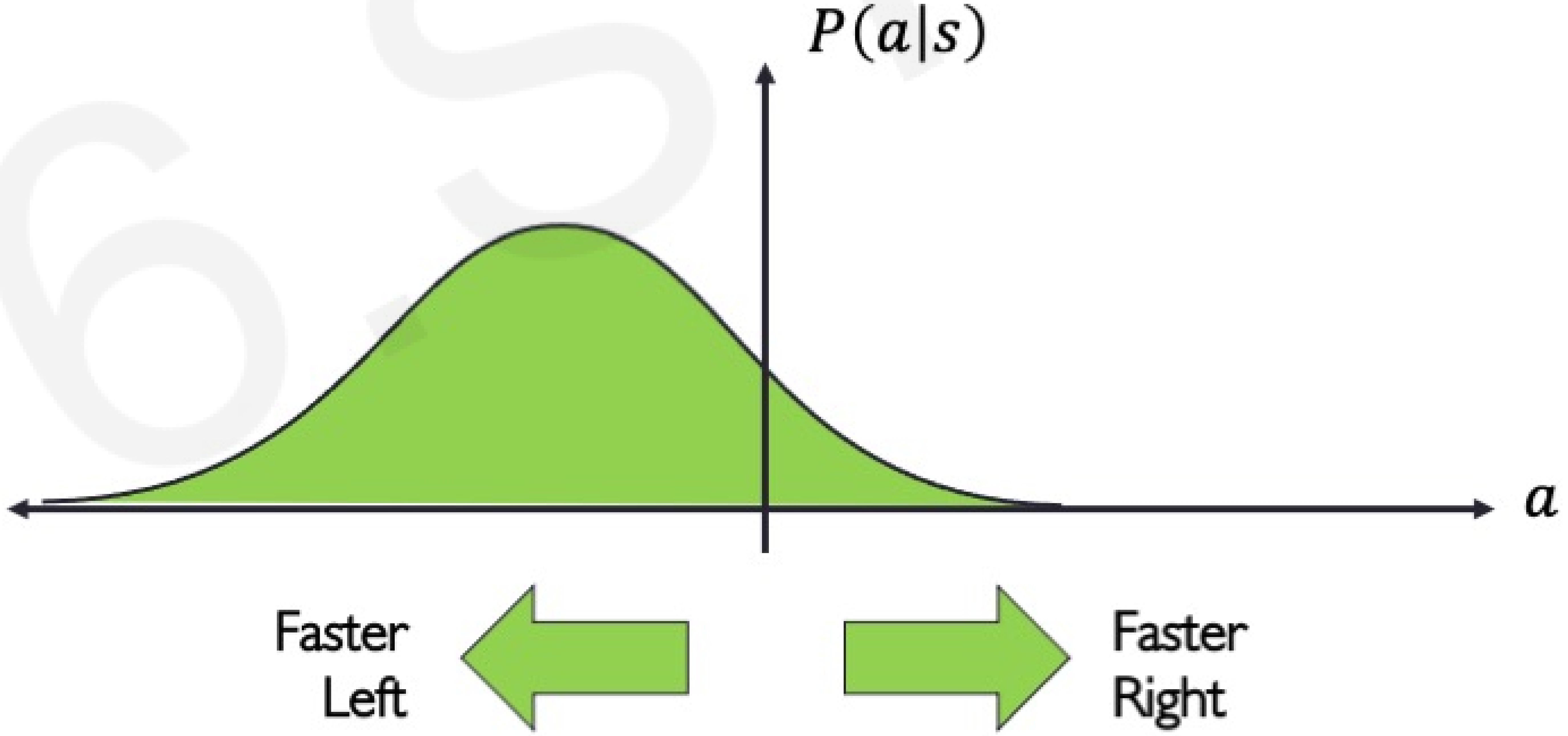
Discrete vs Continuous Action Spaces

Discrete action space: which direction should I move?

Continuous action space: how fast should I move?

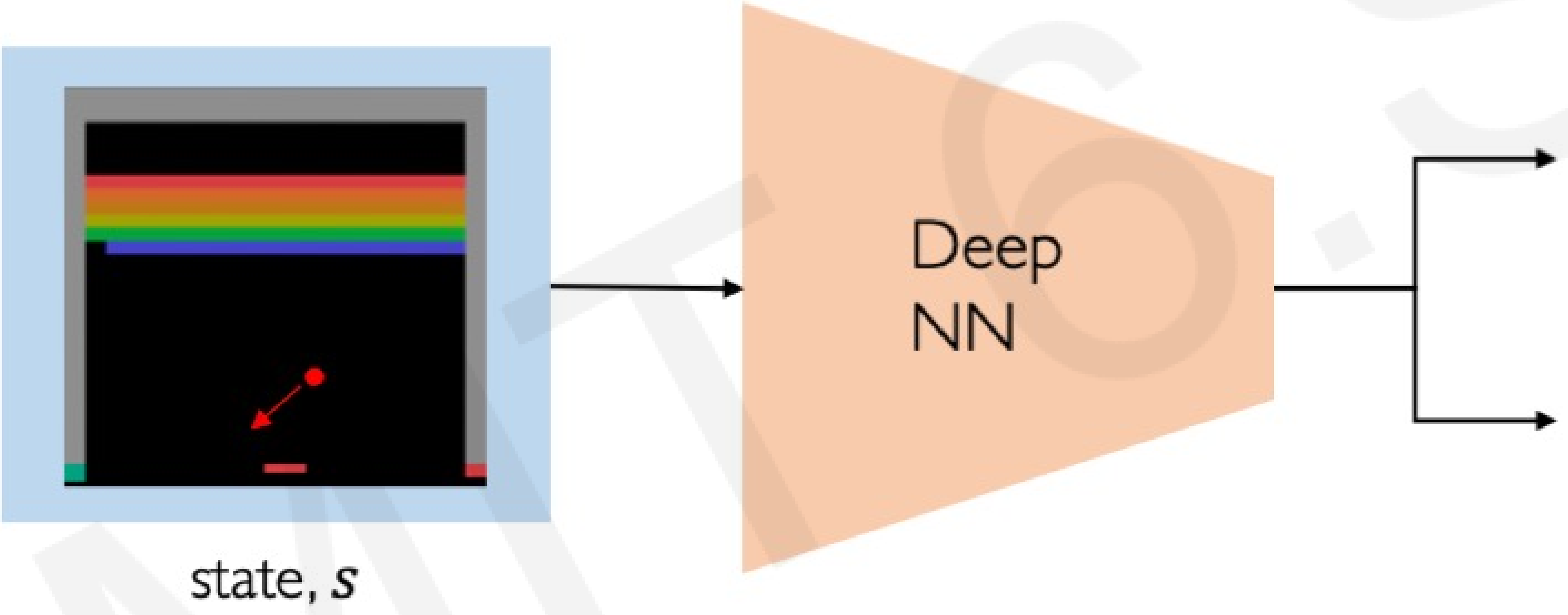


state, s



Policy Gradient (PG): Key Idea

Policy Gradient: Enables modeling of continuous action space



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

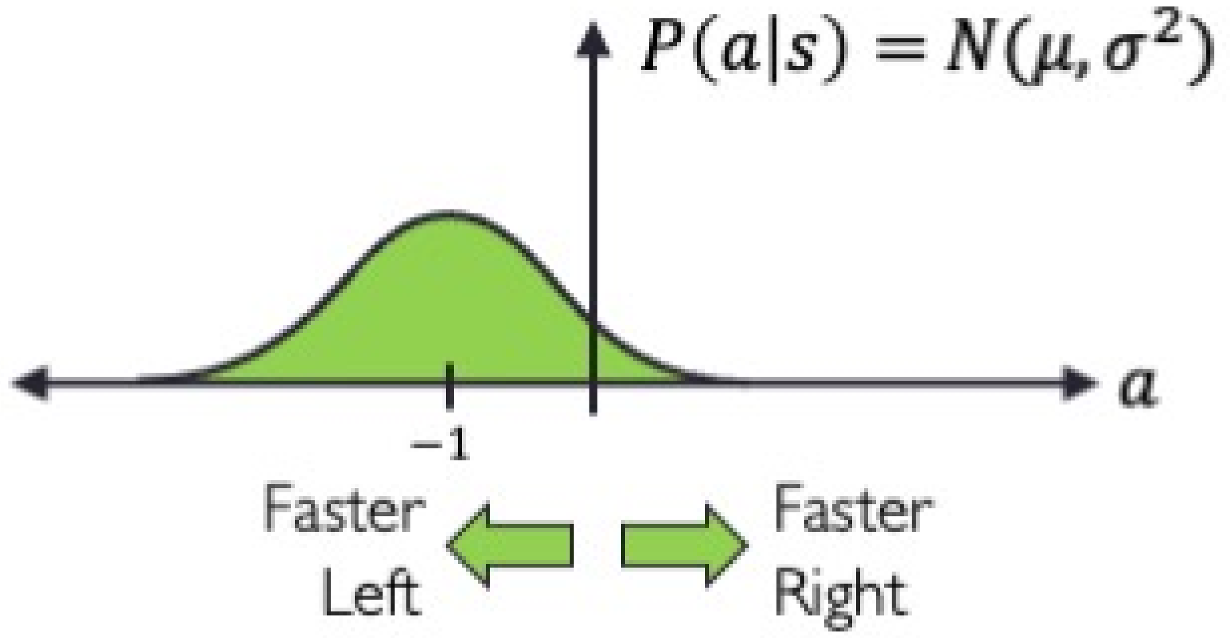
Mean, μ $= -1$

Variance, σ^2 $= 0.5$

$$P(a|s) = N(\mu, \sigma^2)$$

$$\pi(s) \sim P(a|s)$$

$$= -0.8 \text{ [m/s]}$$



Deep Reinforcement Learning: 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