CPEN 400D: Deep Learning

Lecture 10: Deep Reinforcement Learning

Renjie Liao

University of British Columbia

Winter, Term 2, 2022

Outline

- Reinforcement Learning
  - **Overview & Applications**
  - Key Concepts
  - Markov Decision Process (MDP) and its Extensions
  - Bellman Equation and its Optimality
  - RL Taxonomy
- Deep Reinforcement Learning
  - Q-learning & Deep Q-learning
  - Policy Gradient Methods

Reinforcement Learning (RL)

RL is about *learning to take actions that can maximize total future reward*!

Reinforcement Learning (RL)

RL is about *learning to take actions that can maximize total future reward*!

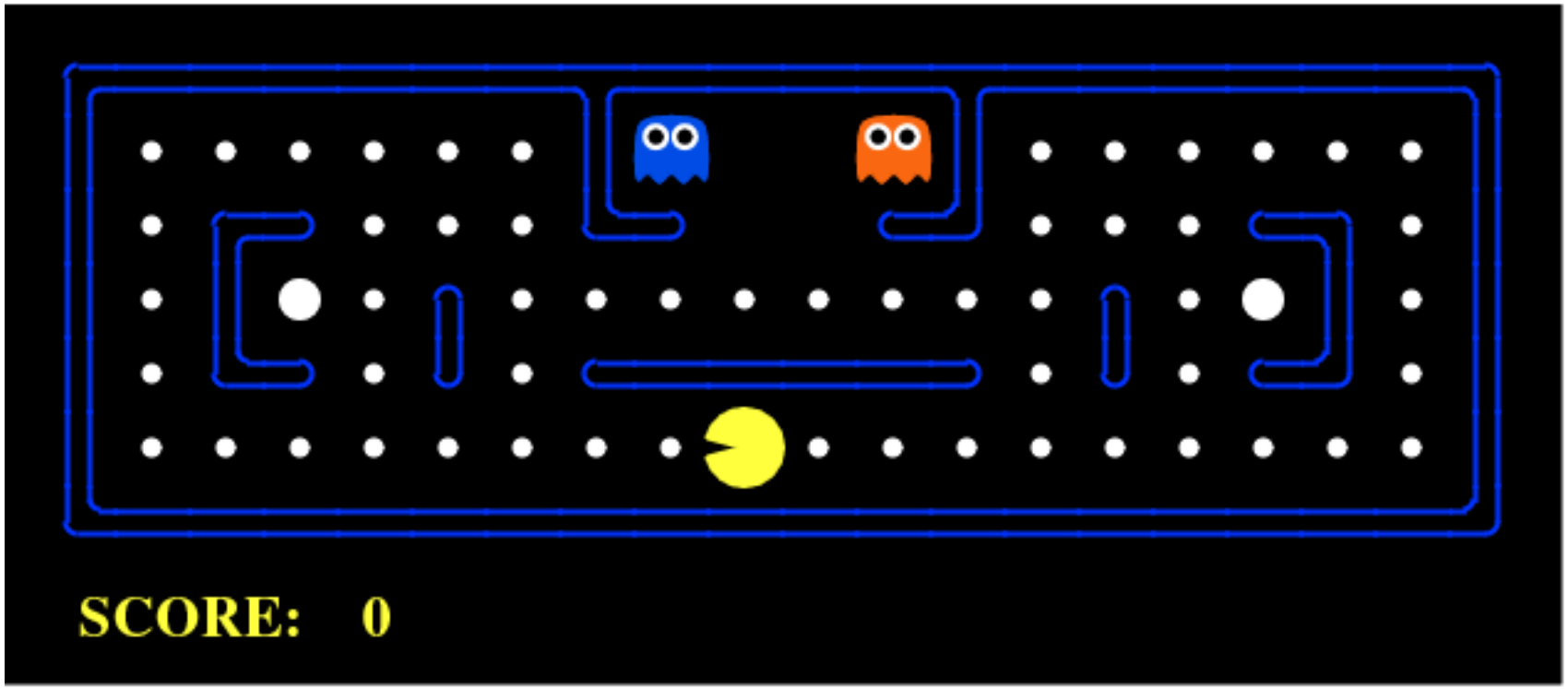


Image Credit: [1]

Reinforcement Learning (RL)

RL is about *learning to take actions that can maximize total future reward*!

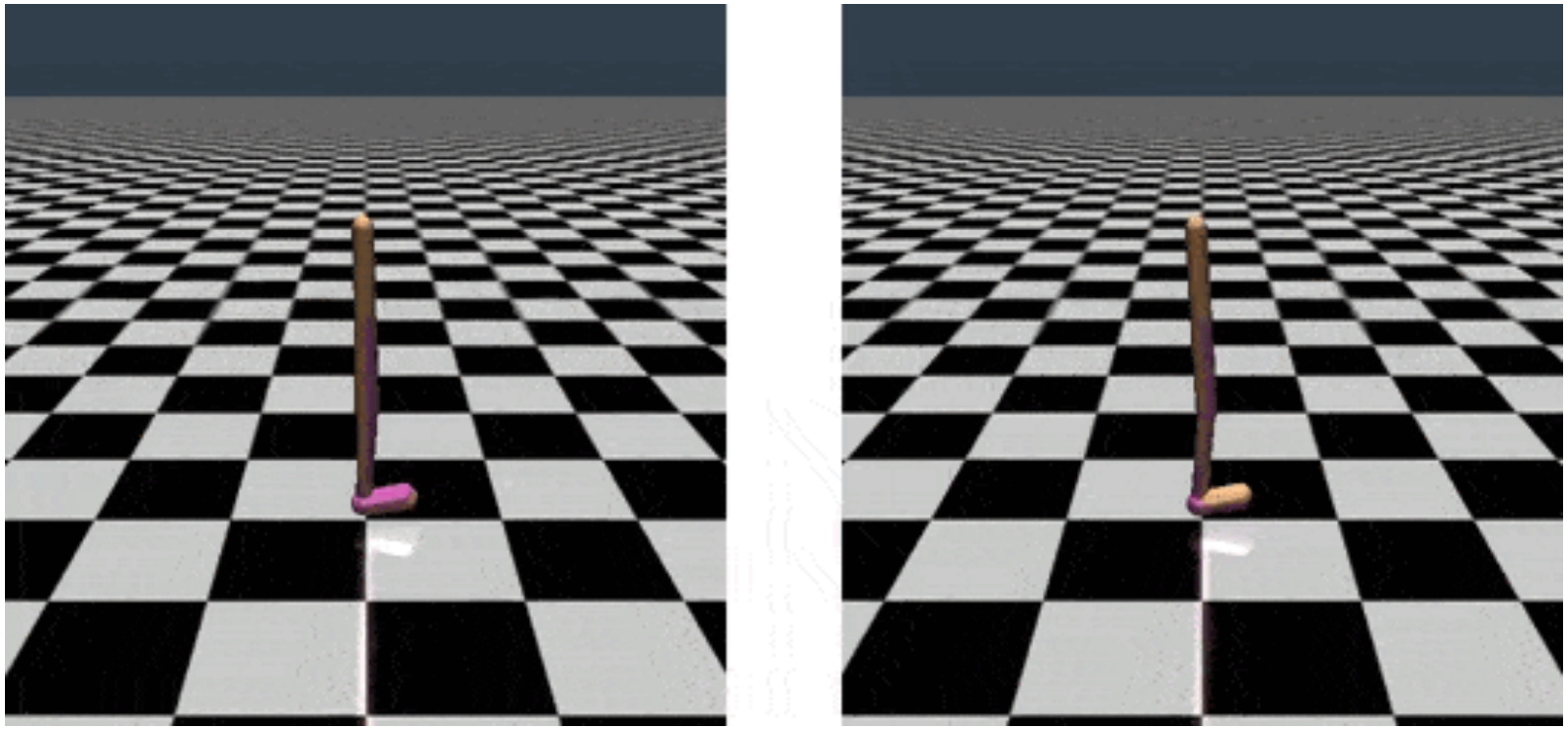


Image Credit: [2]

Reinforcement Learning (RL)

RL is about *learning to take actions that can maximize total future reward*!

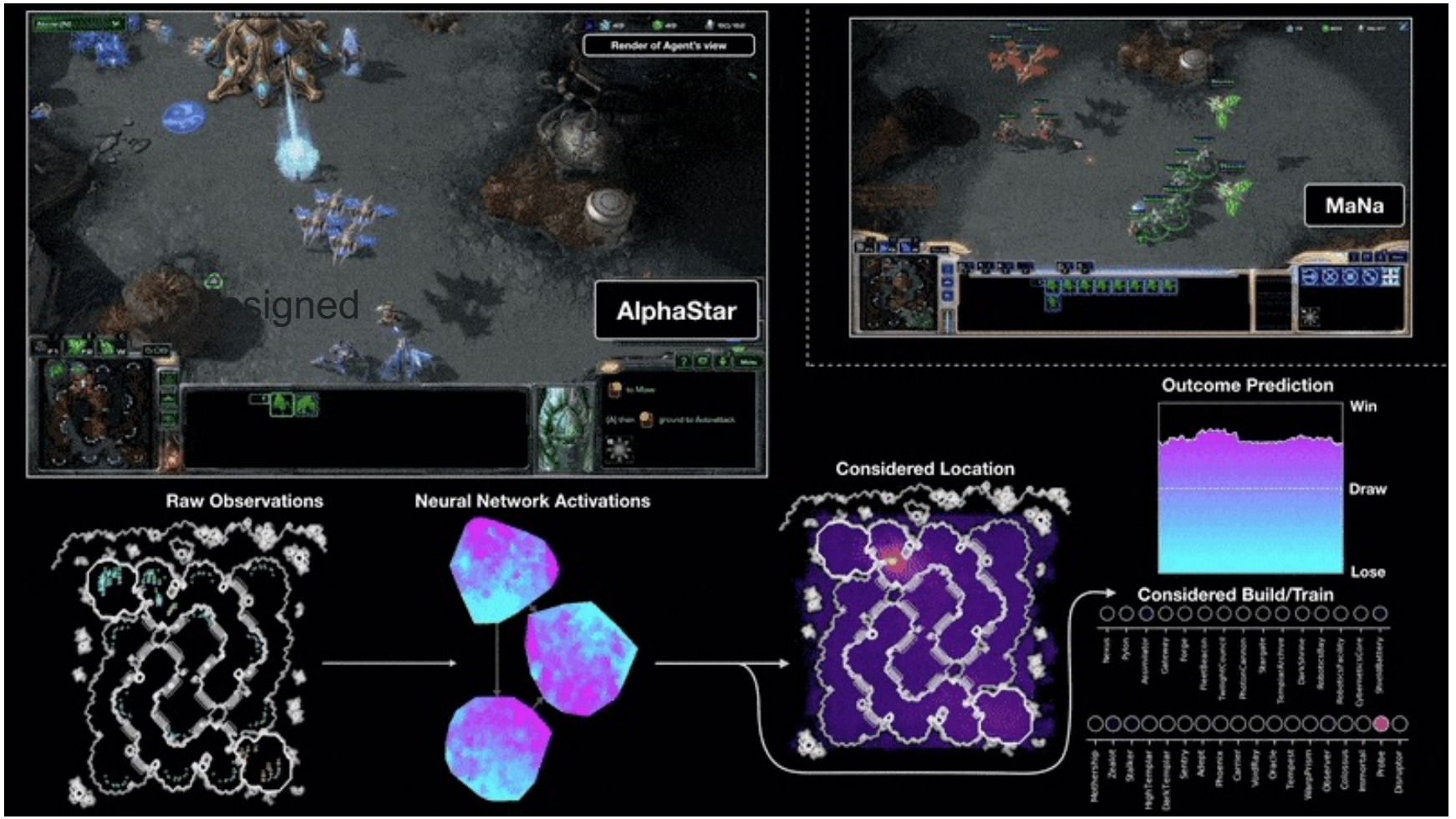
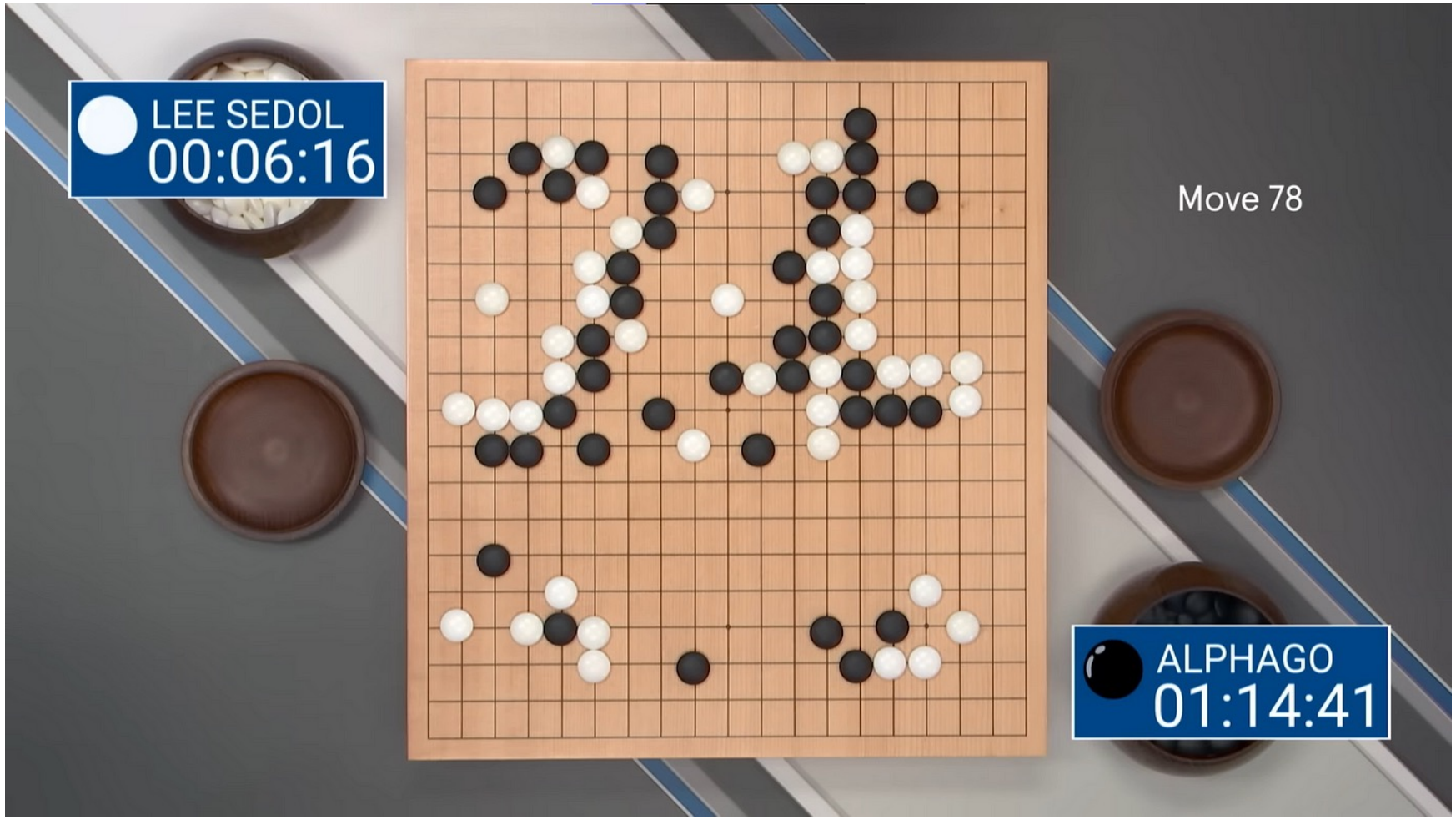


Image Credit: [3]

Reinforcement Learning (RL)

RL is about *learning to take actions that can maximize total future reward*!



God's move: AlphaGo thought this move happens with 0.007% probability in human players!

This may be the last time a human go player beats AI!

Image Credit: [4]

Reinforcement Learning (RL)

RL is about *learning to take actions that can maximize total future reward*!

Image Credit: [5]

Reinforcement Learning (RL)

RL is about *learning to take actions that can maximize total future reward*!

Image Credit: [6]

Reinforcement Learning (RL)

RL is about *learning to take actions that can maximize total future reward*!

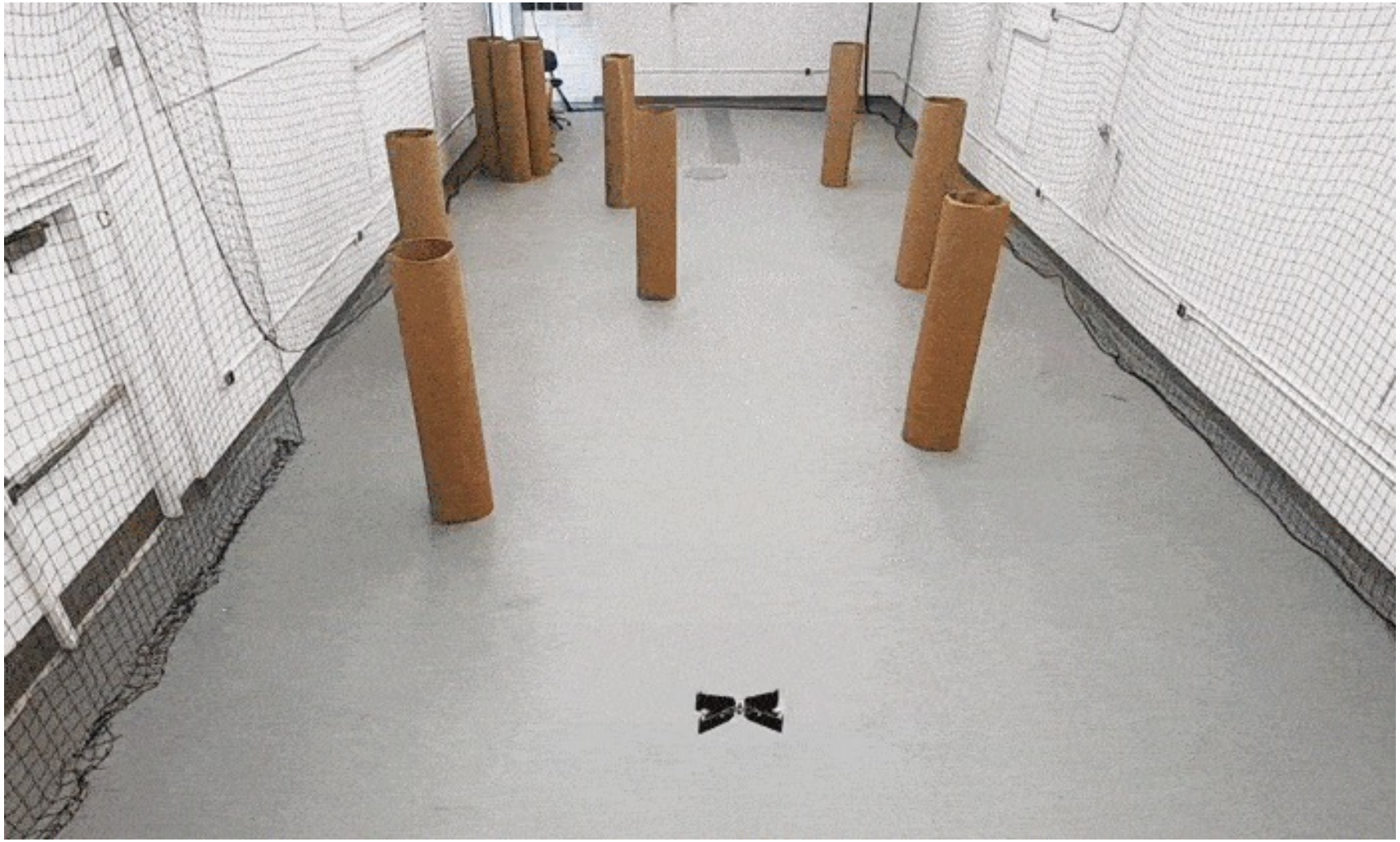


Image Credit: [7]

Outline

- Reinforcement Learning
  - Overview & Applications
  - **Key Concepts**
  - Markov Decision Process (MDP) and its Extensions
  - Bellman Equation and its Optimality
  - RL Taxonomy
- Deep Reinforcement Learning
  - Q-learning & Deep Q-learning
  - Policy Gradient Methods

Reinforcement Learning (RL)

Let us look at the Super Mario example to grab the key concepts:

Agent: an intelligent program or a real robot

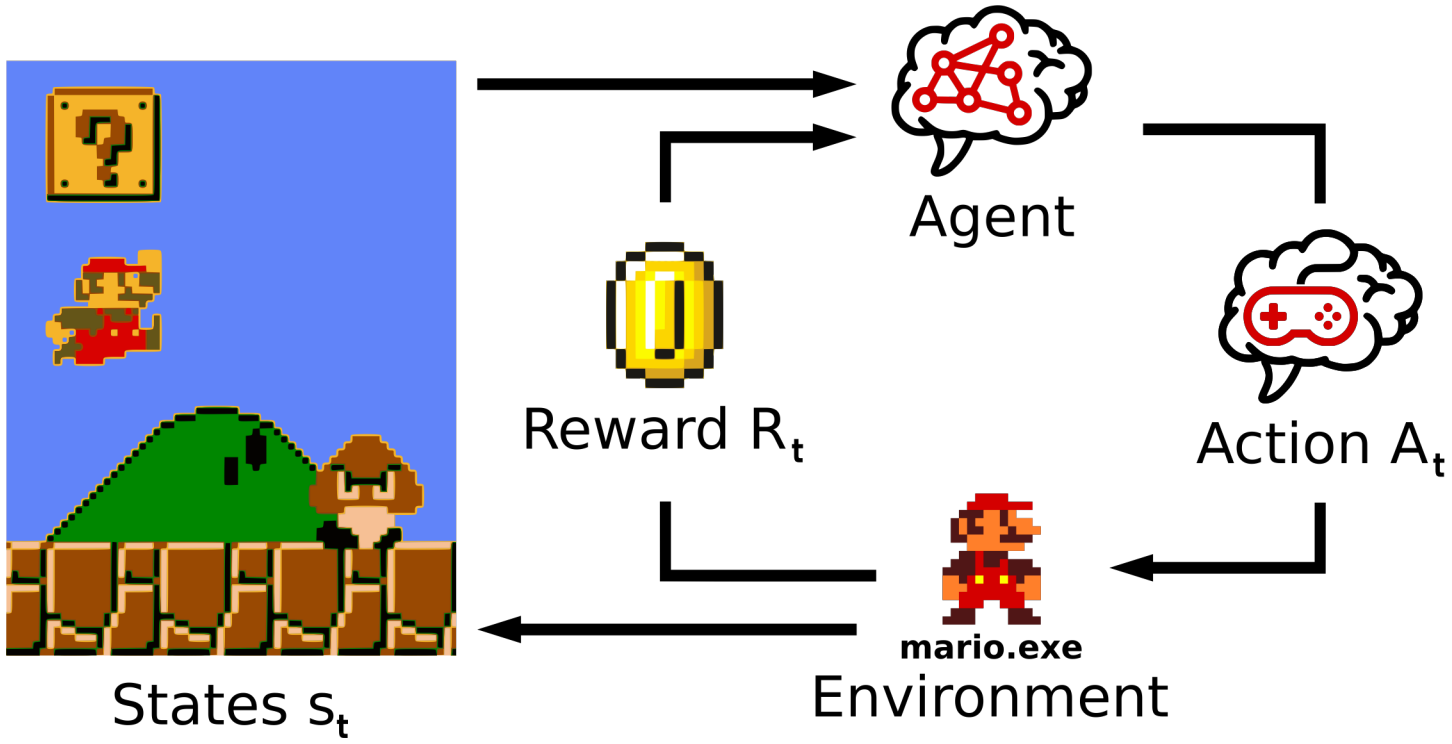
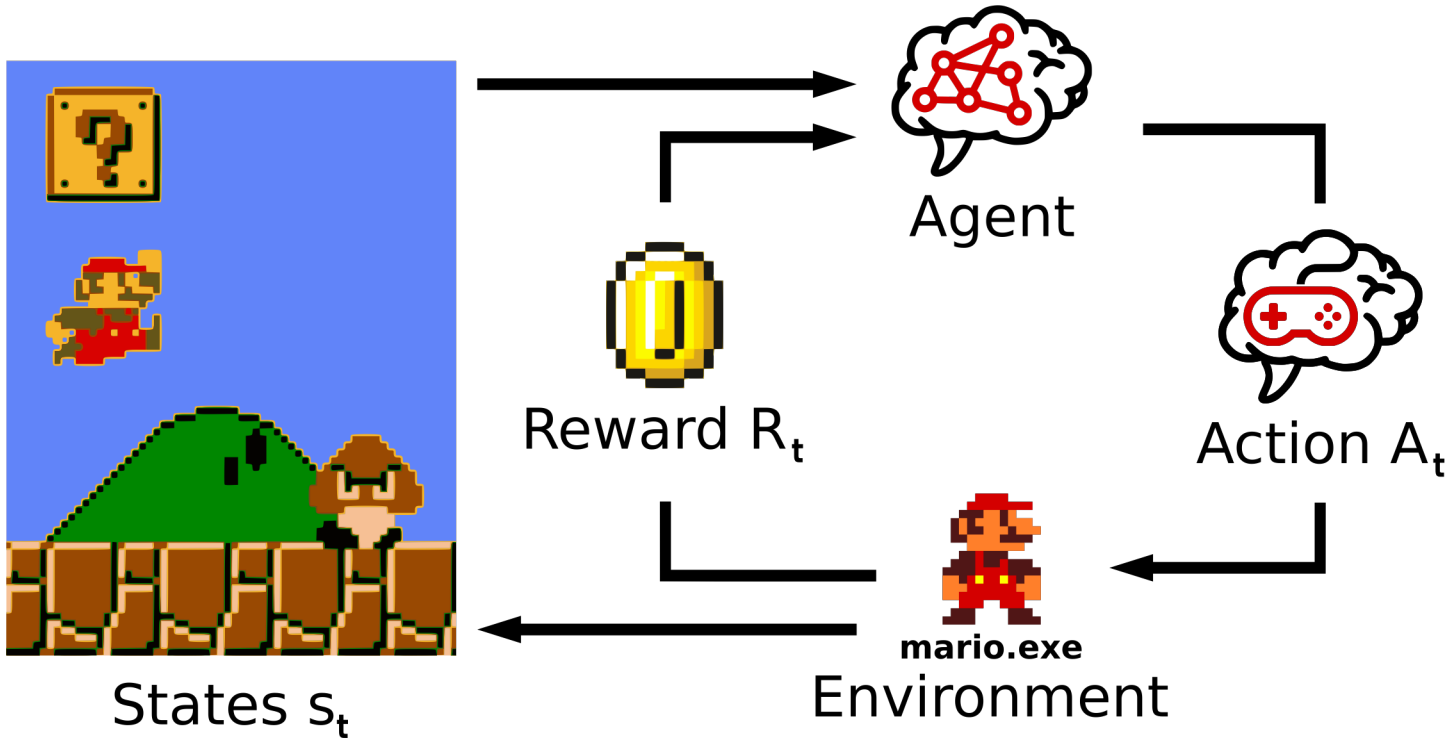


Image Credit: [8]

Reinforcement Learning (RL)

Let us look at the Super Mario example to grab the key concepts:



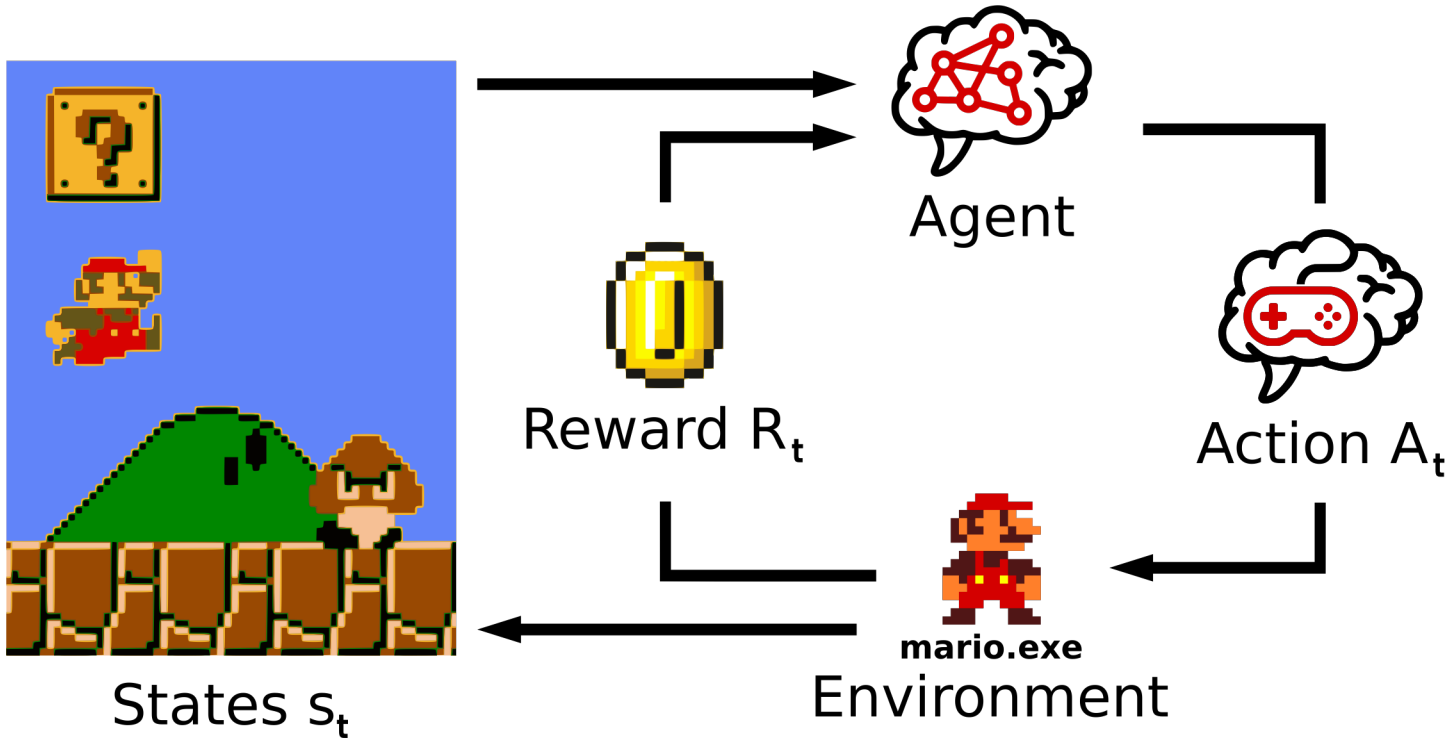
Agent: an intelligent program or a real robot

Environment: the (simulated/real) "world" where the agent interacts

Image Credit: [8]

Reinforcement Learning (RL)

Let us look at the Super Mario example to grab the key concepts:



Agent: an intelligent program or a real robot

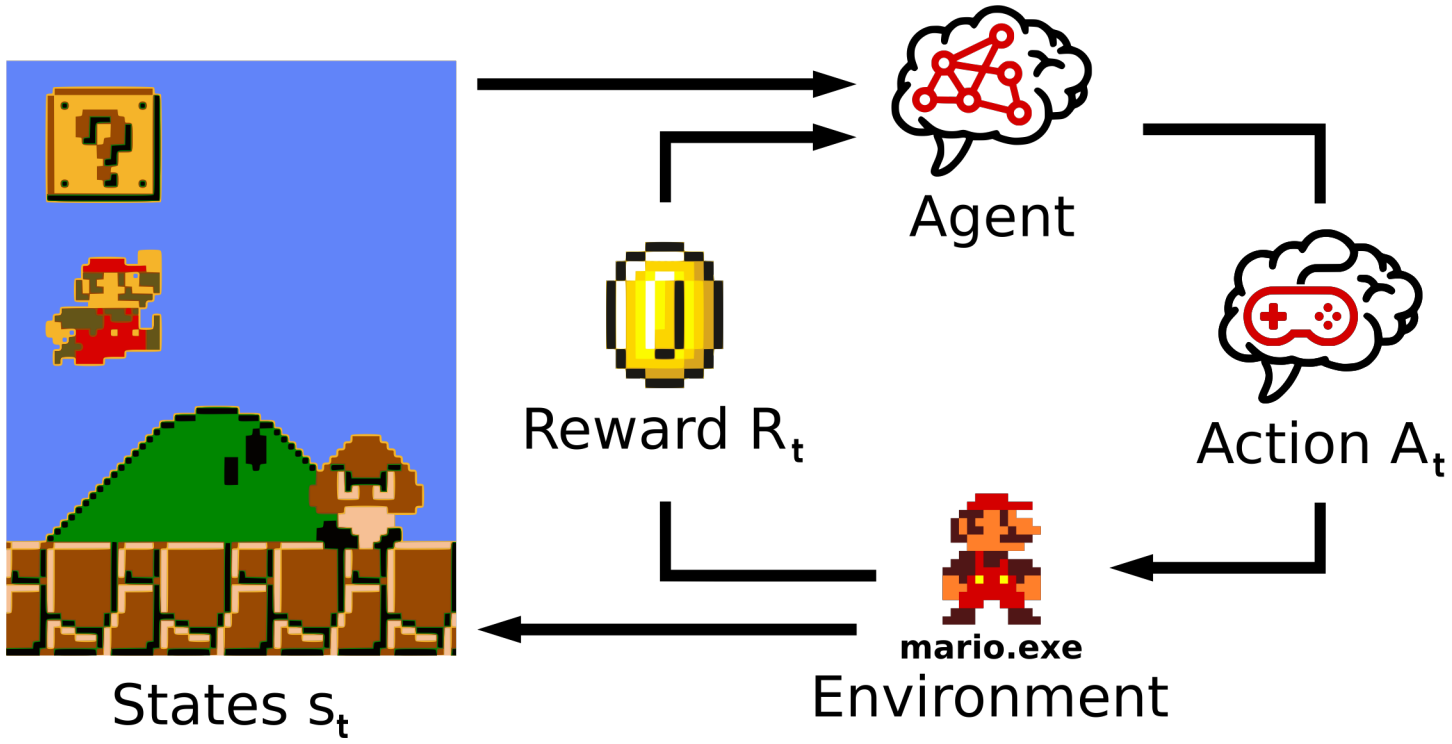
Environment: the (simulated/real) "world" where the agent interacts

State: a function of the past sequence of observations, actions, and rewards.

Image Credit: [8]

Reinforcement Learning (RL)

Let us look at the Super Mario example to grab the key concepts:



Agent: an intelligent program or a real robot

Environment: the (simulated/real) "world" where the agent interacts

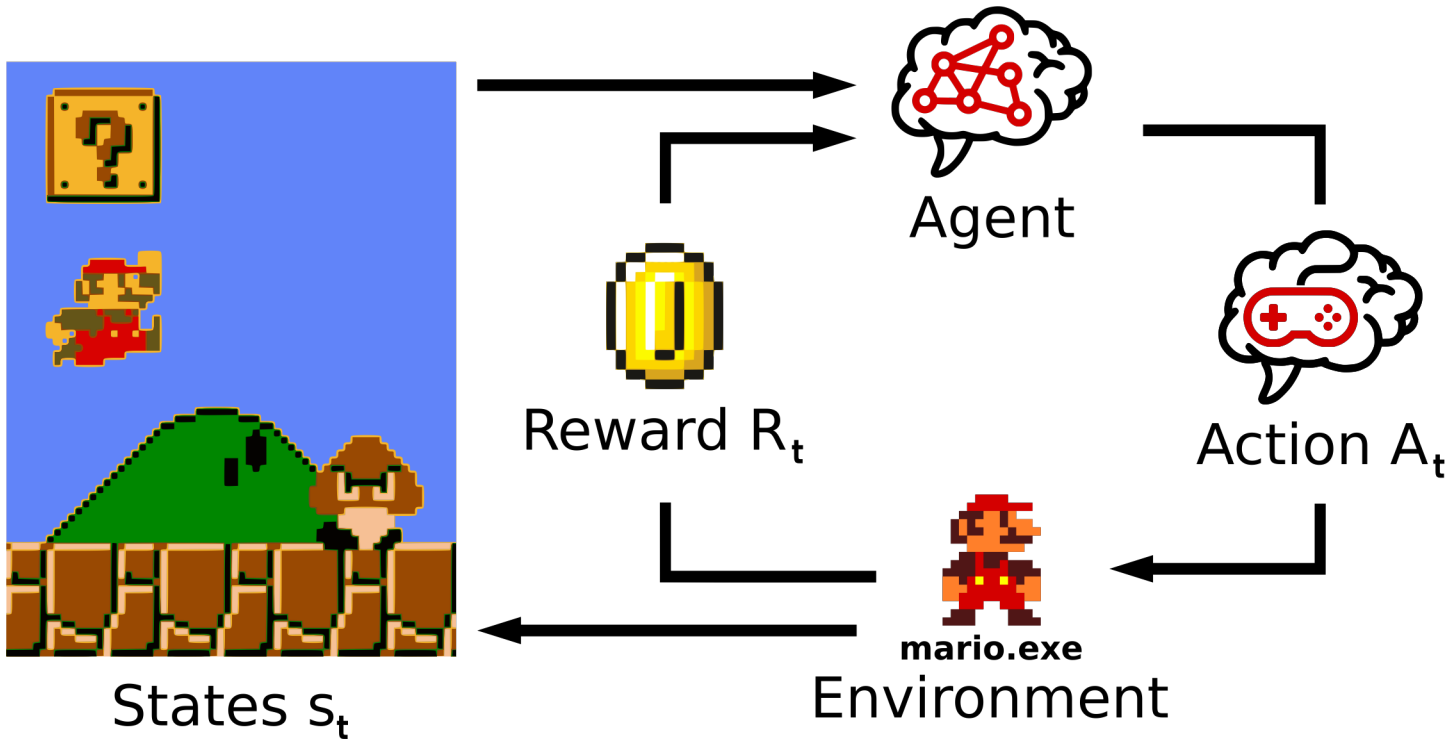
State: a function of the past sequence of observations, actions, and rewards.

Policy: a probability distribution over actions the agent can take given a state

Image Credit: [8]

Reinforcement Learning (RL)

Let us look at the Super Mario example to grab the key concepts:



Agent: an intelligent program or a real robot

Environment: the (simulated/real) "world" where the agent interacts

State: a function of the past sequence of observations, actions, and rewards.

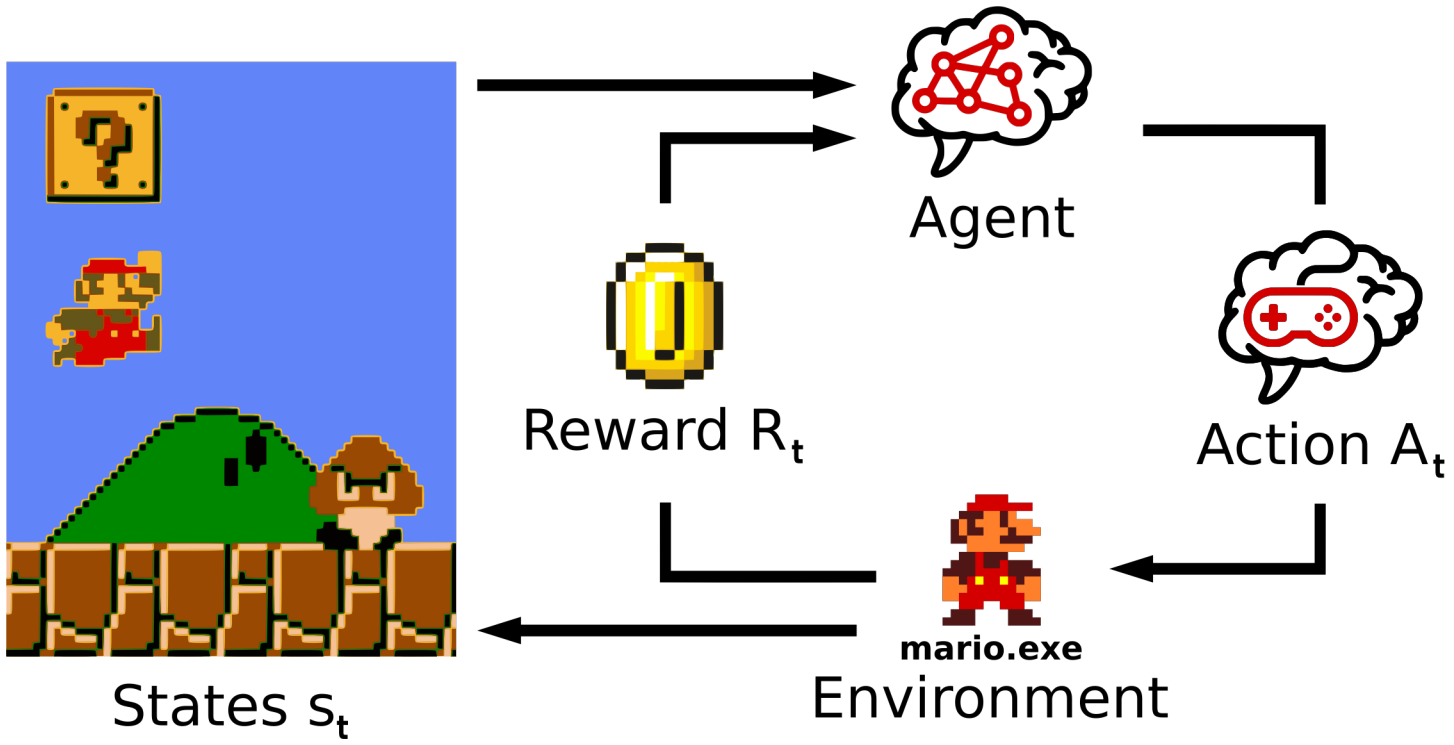
Policy: a probability distribution over actions the agent can take given a state

Action: a transition step the agent takes to move within the environment

Image Credit: [8]

Reinforcement Learning (RL)

Let us look at the Super Mario example to grab the key concepts:



Agent: an intelligent program or a real robot

Environment: the (simulated/real) "world" where the agent interacts

State: a function of the past sequence of observations, actions, and rewards.

Policy: a probability distribution over actions the agent can take given a state

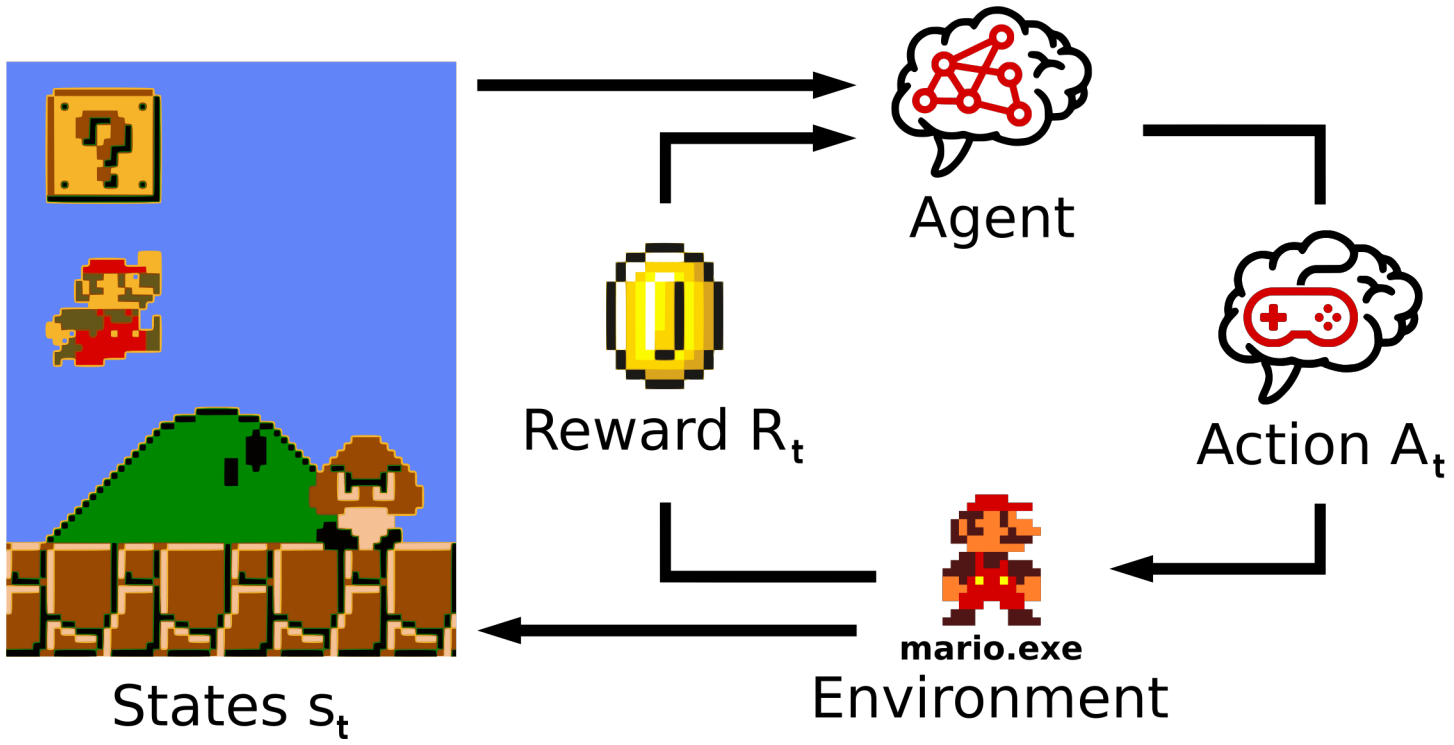
Action: a transition step the agent takes to move within the environment

Reward: the value responded by the environment to the agent's action

Image Credit: [8]

Reinforcement Learning (RL)

Let us look at the Super Mario example to grab the key concepts:



Agent: an intelligent program or a real robot

Environment: the (simulated/real) "world" where the agent interacts

State: a function of the past sequence of observations, actions, and rewards.

Policy: a probability distribution over actions the agent can take given a state

Action: a transition step the agent takes to move within the environment

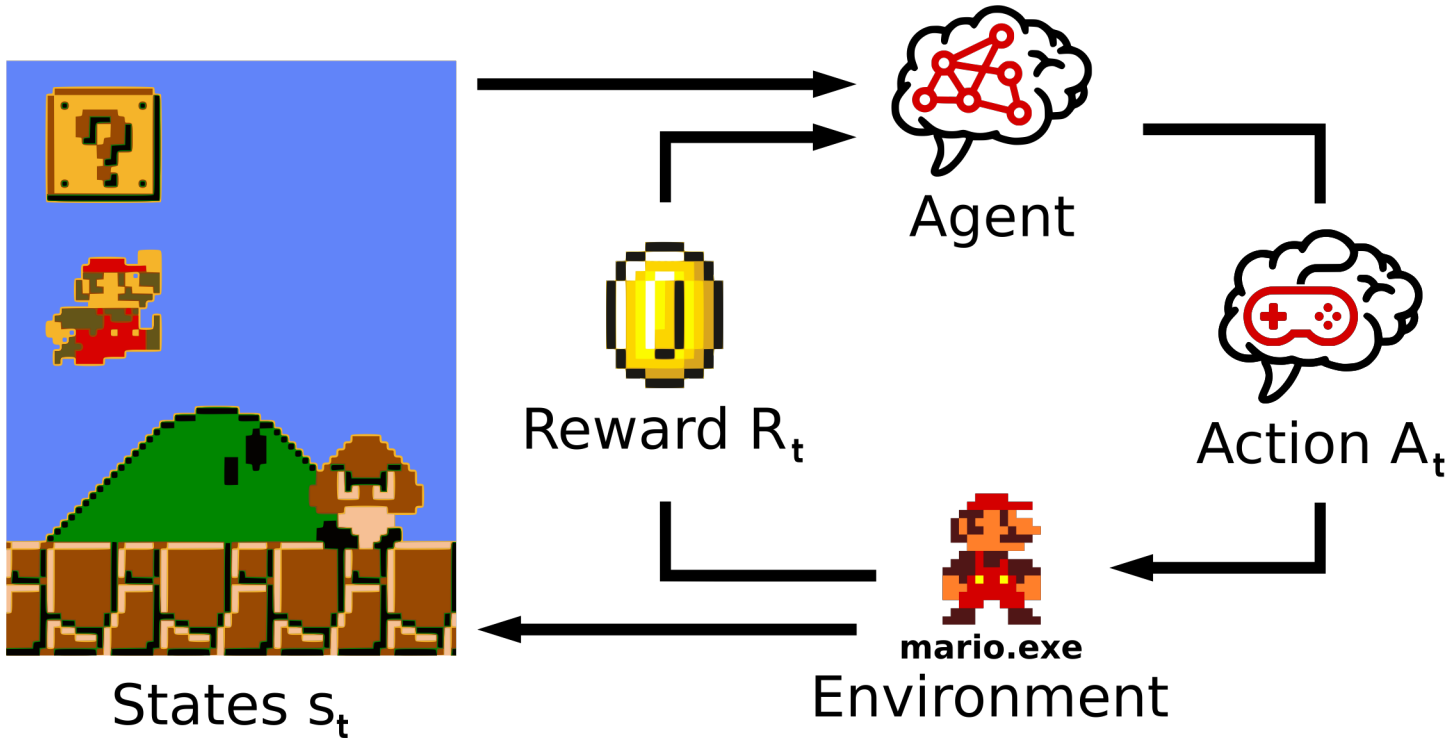
Reward: the value responded by the environment to the agent's action

The interaction within an *episode* leads to a *trajectory* $\boldsymbol{\tau} = (s_0, a_0, r_0, s_1, a_1, r_1, \cdots, s_T)$

Image Credit: [8]

Reinforcement Learning (RL)

As a learning paradigm, RL is different from supervised/unsupervised learning:



- Supervision is scarce, e.g., *reward* is often a scalar
- Supervision is often delayed, e.g., an agent gets the reward after a sequence of actions
- Sequential data is often non-iid, e.g., an agent's current decision would affect the future data distribution

Image Credit: [8]

Outline

- Reinforcement Learning
  - Overview & Applications
  - Key Concepts
  - **Markov Decision Process (MDP) and its Extensions**
  - Bellman Equation and its Optimality
  - RL Taxonomy
- Deep Reinforcement Learning
  - Q-learning & Deep Q-learning
  - Policy Gradient Methods

Markov Decision Process

Almost all RL problems can be formalized as Markov decision processes (MDPs).

Markov Decision Process

Almost all RL problems can be formalized as Markov decision processes (MDPs).

Markov Property: The future is independent of the past given the present!

Markov Decision Process

Almost all RL problems can be formalized as Markov decision processes (MDPs).

A Markov decision process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

Markov Property: The future is independent of the past given the present!

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix $\mathcal{P}^a_{ss'} = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$
- $\mathcal{R}$ is a reward function $\mathcal{R}^a_s = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right]$
- $\gamma \in [0, 1]$ is a discount factor

Markov Decision Process

Almost all RL problems can be formalized as Markov decision processes (MDPs).

A Markov decision process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

Markov Property: The future is independent of the past given the present!

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix
- $\mathcal{R}$ is a reward function
- $\gamma \in [0, 1]$ is a discount factor

$$\mathcal{P}^a_{ss'} = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$$

$$\mathcal{R}^a_s = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right]$$

MDP describes an environment where all states are Markov and can be extended to:

- countably infinite states and or action spaces
- continuous state and or action spaces
- continuous time (requires partial differentiable equations)
- partially observable (POMDPs)

Markov Decision Process

Return: the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Markov Decision Process

Return: the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Why discount? Mathematically convenient, avoid infinite returns, uncertainty about the future……

Markov Decision Process

Return: the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Policy: the distribution over actions given states

$$\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$$

Why discount? Mathematically convenient, avoid infinite returns, uncertainty about the future……

Markov Decision Process

Return: the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Why discount? Mathematically convenient, avoid infinite returns, uncertainty about the future……

Policy: the distribution over actions given states

$$\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$$

We assume *stationary* policies

Markov Decision Process

Return: the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Why discount? Mathematically convenient, avoid infinite returns, uncertainty about the future……

Policy: the distribution over actions given states

$$\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$$

We assume *stationary* policies

Value (a.k.a., State-Value) function: the expected return starting from state s and then following policy π

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t|S_t = s\right]$$

Markov Decision Process

Return: the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Why discount? Mathematically convenient, avoid infinite returns, uncertainty about the future……

Policy: the distribution over actions given states

$$\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$$

We assume *stationary* policies

Value (a.k.a., State-Value) function: the expected return starting from state s and then following policy π

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t|S_t = s\right]$$

Optimal value function

$$V_*(s) = \max_\pi \mathbb{E}_\pi \left[G_t|S_t = s\right]$$

Markov Decision Process

Return: the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Why discount? Mathematically convenient, avoid infinite returns, uncertainty about the future……

Policy: the distribution over actions given states

$$\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$$

We assume *stationary* policies

Value (a.k.a., State-Value) function: the expected return starting from state s and then following policy π

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t|S_t = s\right]$$

Optimal value function

$$V_*(s) = \max_\pi \mathbb{E}_\pi \left[G_t|S_t = s\right]$$

Q (a.k.a. Action-Value) function: the expected return starting from state s, taking action a, and then following policy π

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[G_t|S_t = s, A_t = a\right]$$

Markov Decision Process

Return: the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Why discount? Mathematically convenient, avoid infinite returns, uncertainty about the future……

Policy: the distribution over actions given states

$$\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$$

We assume *stationary* policies

Value (a.k.a., State-Value) function: the expected return starting from state s and then following policy π

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t|S_t = s\right]$$

Optimal value function

$$V_*(s) = \max_\pi \mathbb{E}_\pi \left[G_t|S_t = s\right]$$

Q (a.k.a. Action-Value) function: the expected return starting from state s, taking action a, and then following policy π

$$Q_\pi(s,a) = \mathbb{E}_\pi \left[G_t|S_t = s, A_t = a\right]$$

Optimal Q function

$$Q_*(s,a) = \max_\pi \mathbb{E}_\pi \left[G_t|S_t = s, A_t = a\right]$$

Outline

- Reinforcement Learning
  - Overview & Applications
  - Key Concepts
  - Markov Decision Process (MDP) and its Extensions
  - **Bellman Equation and its Optimality**
  - RL Taxonomy
- Deep Reinforcement Learning
  - Q-learning & Deep Q-learning
  - Policy Gradient Methods

Bellman Equation

Most RL algorithms are based on *Bellman Equation*, which is a recursive formula and has many variations. In particular, for Q-function, we have:

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right]$$

Bellman Equation

Most RL algorithms are based on *Bellman Equation*, which is a recursive formula and has many variations. In particular, for Q-function, we have:

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right]$$

$$= \mathbb{E}_\pi\left[R_{t+1} + \sum_{k=1}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right]$$

Bellman Equation

Most RL algorithms are based on *Bellman Equation*, which is a recursive formula and has many variations. In particular, for Q-function, we have:

$$
\begin{aligned}
Q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \sum_{k=1}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} | S_t = s, A_t = a\right] + \gamma \mathbb{E}_\pi\left[\sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k+1} | S_t = s, A_t = a\right]
\end{aligned}
$$

Bellman Equation

Most RL algorithms are based on *Bellman Equation*, which is a recursive formula and has many variations. In particular, for Q-function, we have:

$$
\begin{aligned}
Q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t|S_t=s,A_t=a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}+\sum_{k=1}^{\infty}\gamma^k R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}|S_t=s,A_t=a\right] + \gamma\mathbb{E}_\pi\left[\sum_{k=1}^{\infty}\gamma^{k-1} R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}\left[R_{t+1}|S_t=s,A_t=a\right] + \\
&\quad \gamma \sum_{S_{t+1},A_{t+1},R_{t+1},\cdots}\left(\sum_{k=1}^{\infty}\gamma^{k-1}R_{t+k+1}\right)\mathbb{P}(S_{t+1},A_{t+1},R_{t+1},\cdots|S_t=s,A_t=a)
\end{aligned}
$$

Bellman Equation

Most RL algorithms are based on *Bellman Equation*, which is a recursive formula and has many variations.
In particular, for Q-function, we have:

$$
\begin{aligned}
Q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t|S_t=s,A_t=a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}+\sum_{k=1}^{\infty}\gamma^k R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}|S_t=s,A_t=a\right] + \gamma\mathbb{E}_\pi\left[\sum_{k=1}^{\infty}\gamma^{k-1} R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}\left[R_{t+1}|S_t=s,A_t=a\right] + \\
&\quad \gamma \sum_{S_{t+1},A_{t+1},R_{t+1},\cdots}\left(\sum_{k=1}^{\infty}\gamma^{k-1}R_{t+k+1}\right)\mathbb{P}(S_{t+1},A_{t+1},R_{t+1},\cdots|S_t=s,A_t=a)
\end{aligned}
$$

$$
\mathcal{R}_s^a = \mathbb{E}\left[R_{t+1}|S_t=s,A_t=a\right]
$$

Bellman Equation

Most RL algorithms are based on *Bellman Equation*, which is a recursive formula and has many variations.
In particular, for Q-function, we have:

$$
\begin{aligned}
Q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t|S_t=s,A_t=a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}+\sum_{k=1}^{\infty}\gamma^k R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}|S_t=s,A_t=a\right]+\gamma\mathbb{E}_\pi\left[\sum_{k=1}^{\infty}\gamma^{k-1} R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}\left[R_{t+1}|S_t=s,A_t=a\right]+ \\
&\quad \gamma\sum_{S_{t+1},A_{t+1},R_{t+1},\cdots}\left(\sum_{k=1}^{\infty}\gamma^{k-1}R_{t+k+1}\right)\mathbb{P}(S_{t+1},A_{t+1},R_{t+1},\cdots|S_t=s,A_t=a) \\
&= \mathcal{R}_s^a+\gamma\sum_{S_{t+1},A_{t+1},R_{t+1},\cdots}\left(\sum_{k=1}^{\infty}\gamma^{k-1}R_{t+k+1}\right)\mathbb{P}(S_{t+2},A_{t+2},R_{t+2},\cdots|S_{t+1}=s',A_{t+1}=a') \\
&\quad \mathbb{P}(A_{t+1}=a'|S_{t+1}=s')\mathbb{P}(S_{t+1}=s'|S_t=s,A_t=a)
\end{aligned}
$$

$$\mathcal{R}_s^a = \mathbb{E}\left[R_{t+1}|S_t=s,A_t=a\right]$$

Bellman Equation

Most RL algorithms are based on *Bellman Equation*, which is a recursive formula and has many variations.
In particular, for Q-function, we have:

$$
\begin{aligned}
Q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t|S_t=s,A_t=a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}+\sum_{k=1}^{\infty}\gamma^k R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}|S_t=s,A_t=a\right]+\gamma\mathbb{E}_\pi\left[\sum_{k=1}^{\infty}\gamma^{k-1} R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}\left[R_{t+1}|S_t=s,A_t=a\right]+ \\
&\quad \gamma\sum_{S_{t+1},A_{t+1},R_{t+1},\cdots}\left(\sum_{k=1}^{\infty}\gamma^{k-1}R_{t+k+1}\right)\mathbb{P}(S_{t+1},A_{t+1},R_{t+1},\cdots|S_t=s,A_t=a) \\
&= \mathcal{R}_s^a+\gamma\sum_{S_{t+1},A_{t+1},R_{t+1},\cdots}\left(\sum_{k=1}^{\infty}\gamma^{k-1}R_{t+k+1}\right)\mathbb{P}(S_{t+2},A_{t+2},R_{t+2},\cdots|S_{t+1}=s',A_{t+1}=a') \\
&\quad \mathbb{P}(A_{t+1}=a'|S_{t+1}=s')\mathbb{P}(S_{t+1}=s'|S_t=s,A_t=a)
\end{aligned}
$$

$$\mathcal{R}_s^a = \mathbb{E}\left[R_{t+1}|S_t=s,A_t=a\right]$$

$$\pi(a|s) = \mathbb{P}(A_t=a|S_t=s)$$

$$\mathcal{P}_{ss'}^a = \mathbb{P}(S_{t+1}=s'|S_t=s,A_t=a)$$

Bellman Equation

Most RL algorithms are based on *Bellman Equation*, which is a recursive formula and has many variations.
In particular, for Q-function, we have:

$$
\begin{aligned}
Q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t|S_t=s,A_t=a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}+\sum_{k=1}^{\infty}\gamma^k R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}|S_t=s,A_t=a\right]+\gamma\mathbb{E}_\pi\left[\sum_{k=1}^{\infty}\gamma^{k-1} R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}\left[R_{t+1}|S_t=s,A_t=a\right]+ \\
&\quad \gamma\sum_{S_{t+1},A_{t+1},R_{t+1},\cdots}\left(\sum_{k=1}^{\infty}\gamma^{k-1}R_{t+k+1}\right)\mathbb{P}(S_{t+1},A_{t+1},R_{t+1},\cdots|S_t=s,A_t=a) \\
&= \mathcal{R}_s^a+\gamma\sum_{S_{t+1},A_{t+1},R_{t+1},\cdots}\left(\sum_{k=1}^{\infty}\gamma^{k-1}R_{t+k+1}\right)\mathbb{P}(S_{t+2},A_{t+2},R_{t+2},\cdots|S_{t+1}=s',A_{t+1}=a') \\
&\quad \mathbb{P}(A_{t+1}=a'|S_{t+1}=s')\mathbb{P}(S_{t+1}=s'|S_t=s,A_t=a) \\
&= \mathcal{R}_s^a+\gamma\sum_{s',a'}\pi(a'|s')\mathcal{P}_{ss'}^a\mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+2}|S_{t+1}=s',A_{t+1}=a'\right]
\end{aligned}
$$

$$\mathcal{R}_s^a = \mathbb{E}\left[R_{t+1}|S_t=s,A_t=a\right]$$

$$\pi(a|s) = \mathbb{P}(A_t=a|S_t=s)$$

$$\mathcal{P}_{ss'}^a = \mathbb{P}(S_{t+1}=s'|S_t=s,A_t=a)$$

Bellman Equation

Most RL algorithms are based on *Bellman Equation*, which is a recursive formula and has many variations.
In particular, for Q-function, we have:

$$
\begin{aligned}
Q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t|S_t=s,A_t=a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}+\sum_{k=1}^{\infty}\gamma^k R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}|S_t=s,A_t=a\right]+\gamma\mathbb{E}_\pi\left[\sum_{k=1}^{\infty}\gamma^{k-1} R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}\left[R_{t+1}|S_t=s,A_t=a\right]+ \\
&\quad \gamma\sum_{S_{t+1},A_{t+1},R_{t+1},\cdots}\left(\sum_{k=1}^{\infty}\gamma^{k-1}R_{t+k+1}\right)\mathbb{P}(S_{t+1},A_{t+1},R_{t+1},\cdots|S_t=s,A_t=a) \\
&= \mathcal{R}_s^a+\gamma\sum_{S_{t+1},A_{t+1},R_{t+1},\cdots}\left(\sum_{k=1}^{\infty}\gamma^{k-1}R_{t+k+1}\right)\mathbb{P}(S_{t+2},A_{t+2},R_{t+2},\cdots|S_{t+1}=s',A_{t+1}=a') \\
&\quad \mathbb{P}(A_{t+1}=a'|S_{t+1}=s')\mathbb{P}(S_{t+1}=s'|S_t=s,A_t=a) \\
&= \mathcal{R}_s^a+\gamma\sum_{s',a'}\pi(a'|s')\mathcal{P}_{ss'}^a\mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+2}|S_{t+1}=s',A_{t+1}=a'\right] \\
&= \mathcal{R}_s^a+\gamma\sum_{s',a'}\pi(a'|s')\mathcal{P}_{ss'}^a Q_\pi(s',a')
\end{aligned}
$$

$$\mathcal{R}_s^a = \mathbb{E}\left[R_{t+1}|S_t=s,A_t=a\right]$$

$$\pi(a|s) = \mathbb{P}(A_t=a|S_t=s)$$

$$\mathcal{P}_{ss'}^a = \mathbb{P}(S_{t+1}=s'|S_t=s,A_t=a)$$

Bellman Equation

Most RL algorithms are based on *Bellman Equation*, which is a recursive formula and has many variations.
In particular, for Q-function, we have:

$$
\begin{aligned}
Q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t|S_t=s,A_t=a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}+\sum_{k=1}^{\infty}\gamma^k R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}|S_t=s,A_t=a\right]+\gamma\mathbb{E}_\pi\left[\sum_{k=1}^{\infty}\gamma^{k-1} R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}\left[R_{t+1}|S_t=s,A_t=a\right]+ \\
&\quad \gamma\sum_{S_{t+1},A_{t+1},R_{t+1},\cdots}\left(\sum_{k=1}^{\infty}\gamma^{k-1}R_{t+k+1}\right)\mathbb{P}(S_{t+1},A_{t+1},R_{t+1},\cdots|S_t=s,A_t=a) \\
&= \mathcal{R}_s^a+\gamma\sum_{S_{t+1},A_{t+1},R_{t+1},\cdots}\left(\sum_{k=1}^{\infty}\gamma^{k-1}R_{t+k+1}\right)\mathbb{P}(S_{t+2},A_{t+2},R_{t+2},\cdots|S_{t+1}=s',A_{t+1}=a') \\
&\quad \mathbb{P}(A_{t+1}=a'|S_{t+1}=s')\mathbb{P}(S_{t+1}=s'|S_t=s,A_t=a) \\
&= \mathcal{R}_s^a+\gamma\sum_{s',a'}\pi(a'|s')\mathcal{P}_{ss'}^a\mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+2}|S_{t+1}=s',A_{t+1}=a'\right] \\
&= \mathcal{R}_s^a+\gamma\sum_{s',a'}\pi(a'|s')\mathcal{P}_{ss'}^a Q_\pi(s',a')
\end{aligned}
$$

$$\mathcal{R}_s^a = \mathbb{E}\left[R_{t+1}|S_t=s,A_t=a\right]$$

$$\pi(a|s) = \mathbb{P}(A_t=a|S_t=s)$$

$$\mathcal{P}_{ss'}^a = \mathbb{P}(S_{t+1}=s'|S_t=s,A_t=a)$$

Proof by induction using *stationary* policies and *homogeneous* Markov chains!

Optimal Bellman Equation

Recall the *optimal Q function is* $Q_*(s,a) = \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right]$

Optimal Bellman Equation

Recall the *optimal Q function is* $Q_*(s,a) = \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a\right]$

The *optimal policy* π_* is thus the one that maximizes the expected return, which can be found as

$$\pi_*(a|s) = \begin{cases} 1 & a = a_*(s) = \underset{a}{\operatorname{argmax}} \;\; Q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

Optimal Bellman Equation

Recall the *optimal Q function is* $Q_*(s,a) = \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a\right]$

The *optimal policy* π_* is thus the one that maximizes the expected return, which can be found as

$$\pi_*(a|s) = \begin{cases} 1 & a = a_*(s) = \operatorname*{argmax}_a \ Q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

The optimal Bellman equation gives a recursive formula for the optimal Q function:

$$\begin{aligned} Q_*(s,a) &= \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a\right] \\ &= \max_\pi \mathcal{R}_s^a + \gamma \sum_{s',a'} \pi(a'|s') \mathcal{P}_{ss'}^a Q_\pi(s',a') \\ &= \mathcal{R}_s^a + \gamma \max_\pi \sum_{s',a'} \pi(a'|s') \mathcal{P}_{ss'}^a Q_\pi(s',a') \\ &= \mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a Q_*(s', a_*(s')) \\ &= \mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \max_{a'} Q_*(s',a') \end{aligned}$$

Outline

- Reinforcement Learning
  - Overview & Applications
  - Key Concepts
  - Markov Decision Process (MDP) and its Extensions
  - Bellman Equation and its Optimality
  - **RL Taxonomy**
- Deep Reinforcement Learning
  - Q-learning & Deep Q-learning
  - Policy Gradient Methods

RL Taxonomy

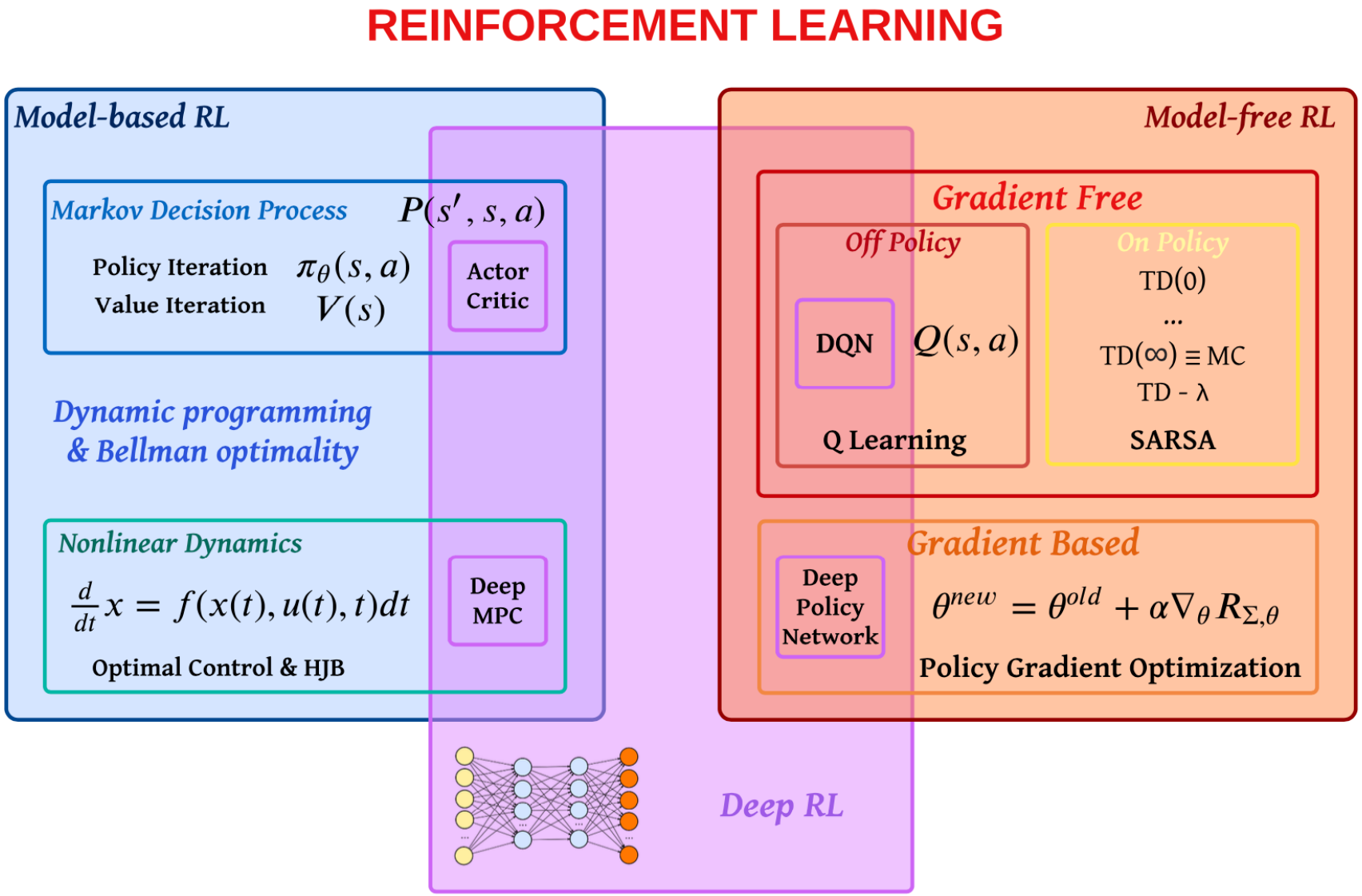


Image Credit: [9]

Outline

- Reinforcement Learning
  - Overview & Applications
  - Key Concepts
  - Markov Decision Process (MDP) and its Extensions
  - Bellman Equation and its Optimality
  - RL Taxonomy
- Deep Reinforcement Learning
  - **Q-learning and Deep Q-learning**
  - Policy Gradient Methods

Q Learning

Recall the *Optimal Bellman Equation*:

$$Q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \max_{a'} Q_*(s',a')$$

Q Learning

Recall the *Optimal Bellman Equation*: $$Q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \max_{a'} Q_*(s',a')$$

Given sampled trajectories, we can define the *Bellman Error* (of one time step) as:

$$\mathcal{R}_{s_t}^{a_t} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$$

Q Learning

Recall the *Optimal Bellman Equation*: $$Q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \max_{a'} Q_*(s',a')$$

Given sampled trajectories, we can define the *Bellman Error* (of one time step) as:

$$\mathcal{R}_{s_t}^{a_t} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$$

The idea of *Q Learning* [10] is to learn a Q function that minimizes the Bellman Error. In particular, we can use the fix point iteration to update the Q function iteratively:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta \left[\mathcal{R}_{s_t}^{a_t} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

If this update converges, the Bellman error should reach 0!

Q Learning

Recall the *Optimal Bellman Equation*: $$Q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \max_{a'} Q_*(s',a')$$

Given sampled trajectories, we can define the *Bellman Error* (of one time step) as:

$$\mathcal{R}_{s_t}^{a_t} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)$$

The idea of *Q Learning* [10] is to learn a Q function that minimizes the Bellman Error. In particular, we can use the fix point iteration to update the Q function iteratively:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \eta \left[\mathcal{R}_{s_t}^{a_t} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)\right]$$

If this update converges, the Bellman error should reach 0!

Exploration-exploitation tradeoff: Q learning only learns from the state-action pairs it visits. One often needs some strategy to improve the exploration, e.g., ϵ-greedy policy [11] (choose optimal action based on Q with probability ϵ and choose a random action with probability 1- ϵ).

Q Learning

Recall the *Optimal Bellman Equation*: $$Q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \max_{a'} Q_*(s',a')$$

Given sampled trajectories, we can define the *Bellman Error* (of one time step) as:

$$\mathcal{R}_{s_t}^{a_t} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)$$

The idea of *Q Learning* [10] is to learn a Q function that minimizes the Bellman Error. In particular, we can use the fix point iteration to update the Q function iteratively:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \eta \left[\mathcal{R}_{s_t}^{a_t} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)\right]$$

If this update converges, the Bellman error should reach 0!

Exploration-exploitation tradeoff: Q learning only learns from the state-action pairs it visits. One often needs some strategy to improve the exploration, e.g., ϵ-greedy policy [11] (choose optimal action based on Q with probability ϵ and choose a random action with probability 1- ϵ).

For small state and action spaces, we can represent Q function as a table and learn it. However, for large or infinite spaces, we need to represent it as a parametric function, e.g., a deep neural network!

Deep Q Learning

Approximating Q function with a neural net is a decades-old idea, but DeepMind got it to work really well on Atari games in 2013 (“deep Q-learning”) [12].

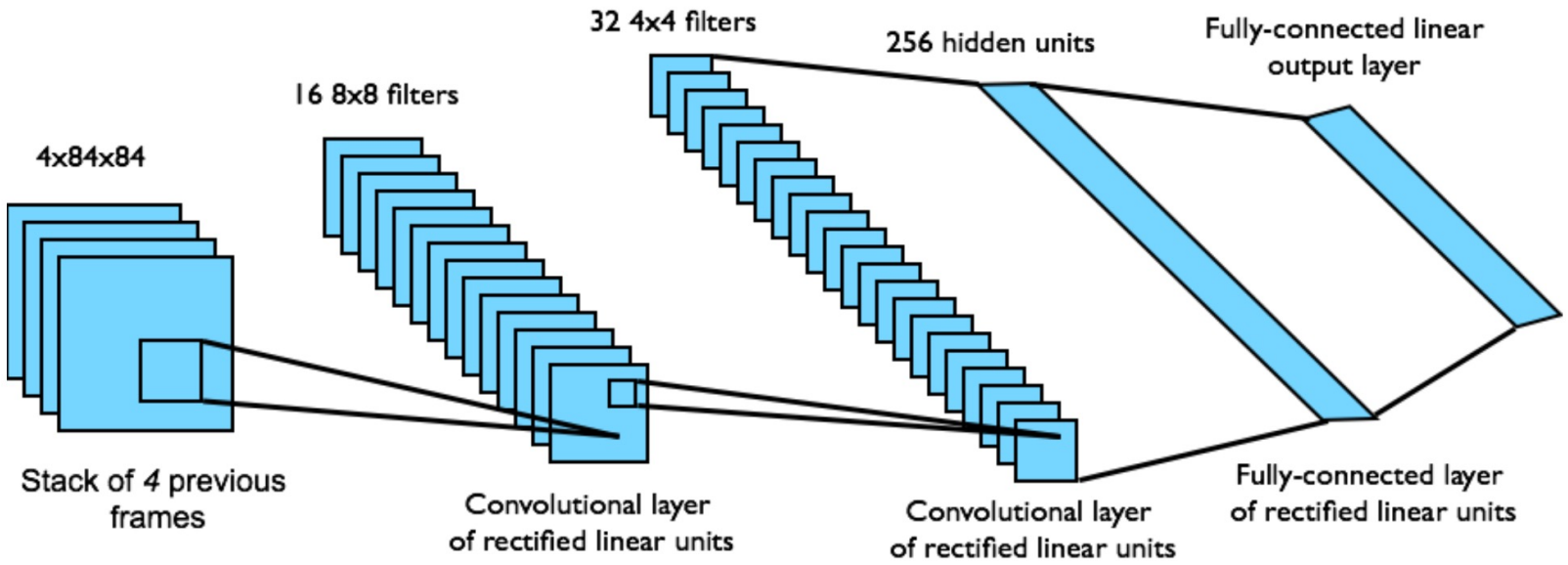


Image Credit: [13]

Deep Q Learning

Approximating Q function with a neural net is a decades-old idea, but DeepMind got it to work really well on Atari games in 2013 (“deep Q-learning”) [12].

- Take actions following ϵ-greedy policy

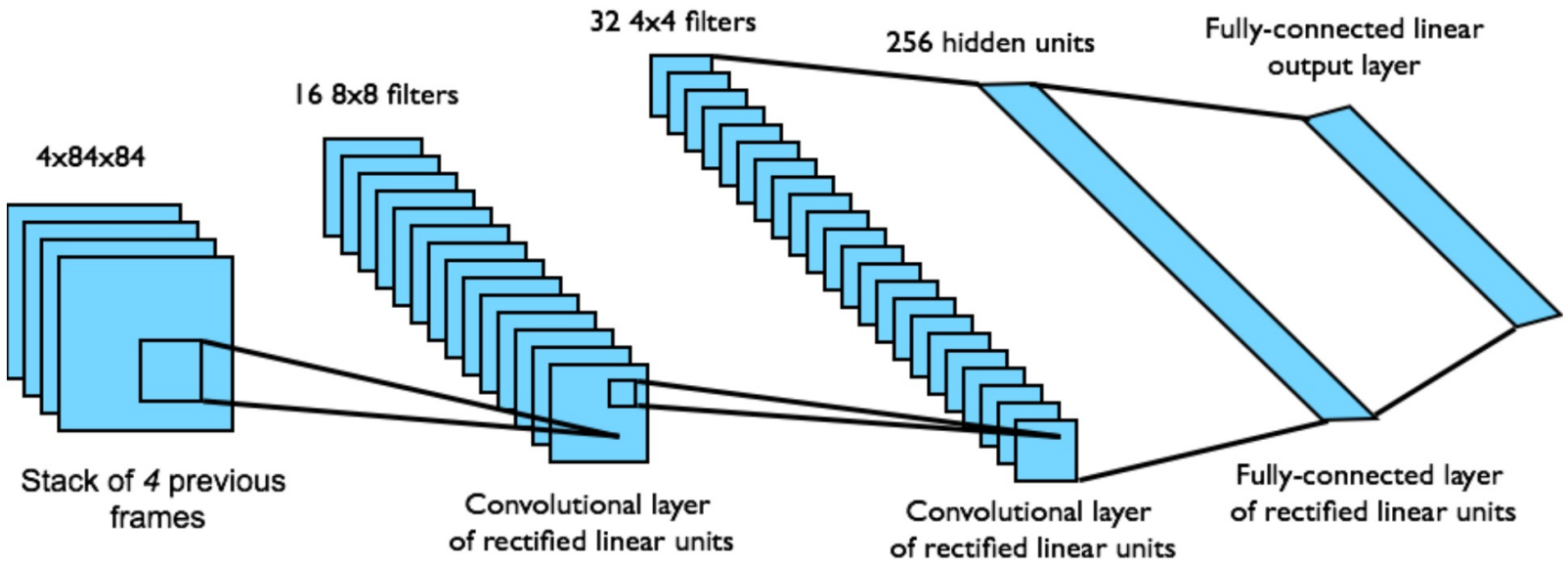


Image Credit: [13]

Deep Q Learning

Approximating Q function with a neural net is a decades-old idea, but DeepMind got it to work really well on Atari games in 2013 ("deep Q-learning") [12].

- Take actions following ϵ-greedy policy
- Store $(s_t, a_t, r_{t+1}, s_{t+1})$ in *replay buffer* and sample random mini-batch of tuples from the buffer

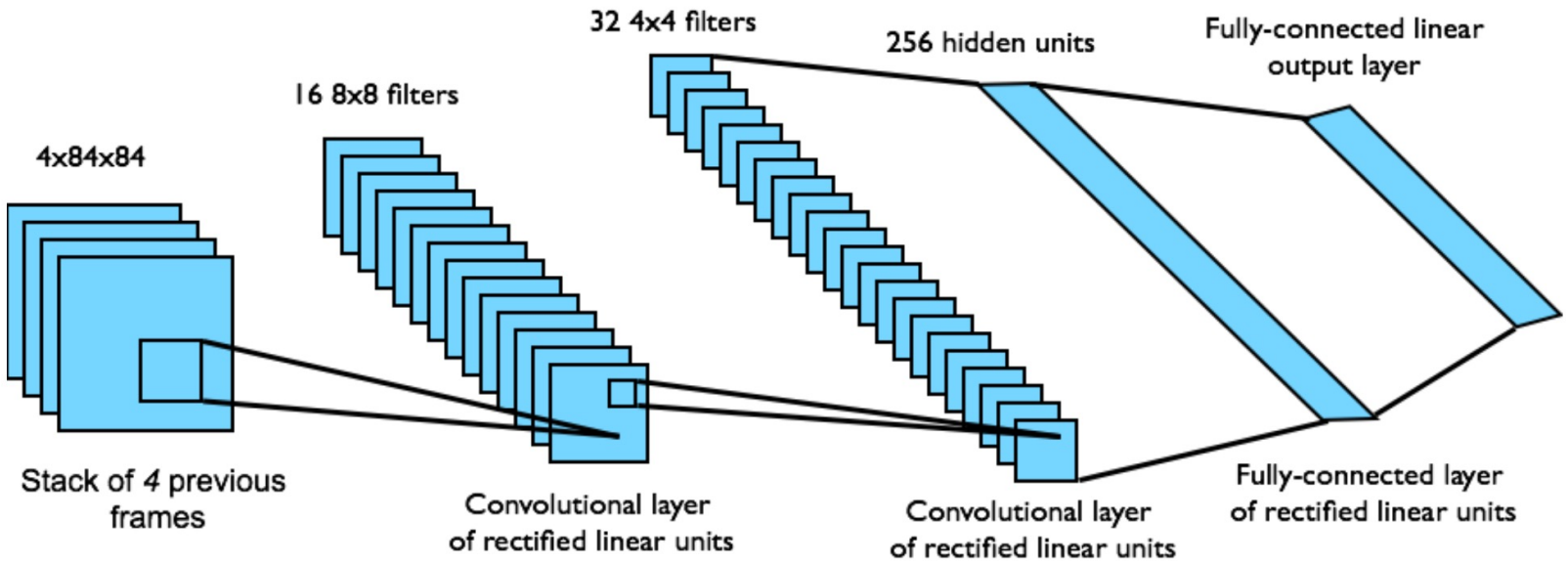


Image Credit: [13]

Deep Q Learning

Approximating Q function with a neural net is a decades-old idea, but DeepMind got it to work really well on Atari games in 2013 ("deep Q-learning") [12].

- Take actions following ϵ-greedy policy
- Store $(s_t, a_t, r_{t+1}, s_{t+1})$ in *replay buffer* and sample random mini-batch of tuples from the buffer
- Compute Q-targets w.r.t. old and fixed parameters $\bar{\theta}$

$$\theta_{t+1} \leftarrow \theta_t + \eta \mathbb{E}_{s_t, a_t, s_{t+1}} \left[\left(\mathcal{R}_{s_t}^{a_t} + \gamma \max_a Q_{\bar{\theta}}(s_{t+1}, a) - Q_{\theta_t}(s_t, a_t) \right) \frac{\partial Q_{\theta_t}(s_t, a_t)}{\partial \theta_t} \right]$$

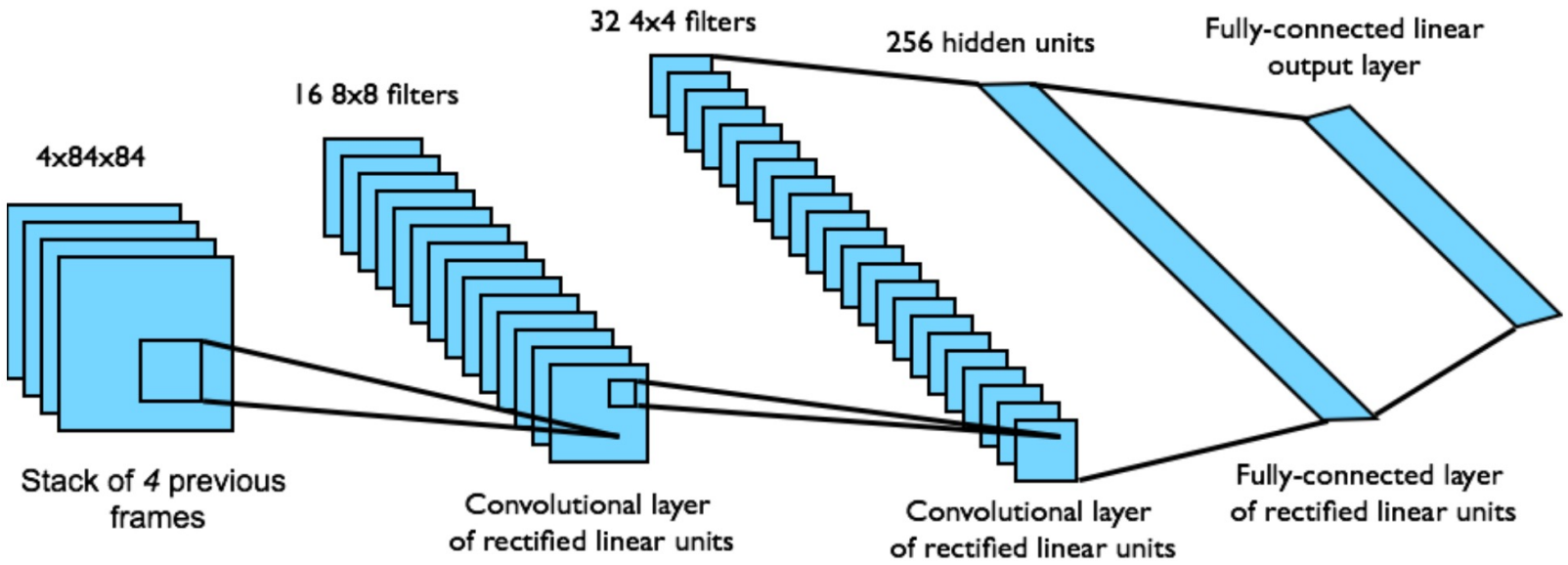


Image Credit: [13]

Outline

- Reinforcement Learning
  - Overview & Applications
  - Key Concepts
  - Markov Decision Process (MDP) and its Extensions
  - Bellman Equation and its Optimality
  - RL Taxonomy
- Deep Reinforcement Learning
  - Q-learning and Deep Q-learning
  - **Policy Gradient Methods**

Policy Gradient Methods

In deep Q learning, we parameterize the Q function as a neural network and learn it to minimize Bellman error. A policy is then obtained from Q function, e.g., via ϵ-greedy strategy.

Policy Gradient Methods

In deep Q learning, we parameterize the Q function as a neural network and learn it to minimize Bellman error. A policy is then obtained from Q function, e.g., via ϵ-greedy strategy.

- Value Based
  - Learnt Value Function
  - Implicit policy (e.g. ϵ-greedy)
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic
  - Learnt Value Function
  - Learnt Policy

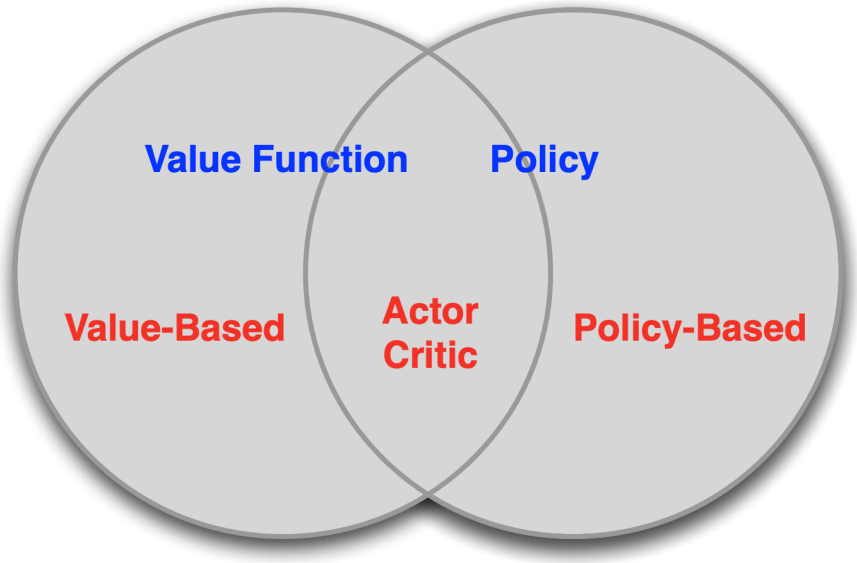


Image Credit: [14]

Policy Gradient Methods

In policy based methods, we directly parameterize the policy and learn it to maximize some reward.

Policy Gradient Methods

In policy based methods, we directly parameterize the policy and learn it to maximize some reward.

Advantages:

- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

Policy Gradient Methods

In policy based methods, we directly parameterize the policy and learn it to maximize some reward.

Advantages:

- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

Disadvantages:

- Often converge to local rather than global optimum
- Evaluating a policy is typically inefficient and high variance

Policy Gradient Methods

In policy based methods, we directly parameterize the policy and learn it to maximize some reward.

Given a trajectory $\boldsymbol{\tau}$, let us consider the simple expected reward:

$$J(\theta) = \mathbb{E}_{\boldsymbol{\tau}}\left[R(\boldsymbol{\tau})\right] = \int \mathbb{P}_{\theta}(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$$

Policy Gradient Methods

In policy based methods, we directly parameterize the policy and learn it to maximize some reward.

Given a trajectory $\boldsymbol{\tau}$, let us consider the simple expected reward:

$$J(\theta) = \mathbb{E}_{\boldsymbol{\tau}}\left[R(\boldsymbol{\tau})\right] = \int \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$$

$$= \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{t=1}^{T} R_t\right] = \mathbb{E}_{\mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)}\left[\sum_{t=1}^{T} R_t(A_t, S_t)\right]$$

Policy Gradient Methods

In policy based methods, we directly parameterize the policy and learn it to maximize some reward.

Given a trajectory $\boldsymbol{\tau}$, let us consider the simple expected reward:

$$J(\theta) = \mathbb{E}_{\boldsymbol{\tau}}\left[R(\boldsymbol{\tau})\right] = \int \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$$

$$= \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{t=1}^{T} R_t\right] = \mathbb{E}_{\mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)}\left[\sum_{t=1}^{T} R_t(A_t, S_t)\right]$$

Log derivative trick:

$$\nabla_\theta J(\theta) = \nabla_\theta \int \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$$

$$= \int \nabla_\theta \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$$

$$= \int \mathbb{P}_\theta(\boldsymbol{\tau}) \nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$$

$$= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau})\right]$$

Policy Gradient Methods

Let us substitute $\mathbb{P}_\theta(\boldsymbol{\tau}) = \mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)$ $R(\boldsymbol{\tau}) = \sum_{t=1}^{T} R_t(A_t,S_t)$

We have

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau})\right] \\ &= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log\left(\mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right]\end{aligned}$$

Policy Gradient Methods

Let us substitute $\mathbb{P}_\theta(\boldsymbol{\tau}) = \mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)$ $\quad R(\boldsymbol{\tau}) = \sum_{t=1}^{T} R_t(A_t,S_t)$

We have

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau})\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log\left(\mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta\left(\log \mathbb{P}_0(S) + \sum_{t=1}^{T}\log\pi_\theta(A_t|S_t) + \sum_{t=1}^{T}\log\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(A_t|S_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right]
\end{aligned}$$

No dependence on starting and transition probability of the environment, thus being model-free!

Policy Gradient Methods

Let us substitute $\mathbb{P}_\theta(\boldsymbol{\tau}) = \mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)$ $\quad R(\boldsymbol{\tau}) = \sum_{t=1}^{T} R_t(A_t,S_t)$

We have

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau})\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log\left(\mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta\left(\log \mathbb{P}_0(S) + \sum_{t=1}^{T}\log\pi_\theta(A_t|S_t) + \sum_{t=1}^{T}\log\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(A_t|S_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(a_t^{(i)}|s_t^{(i)})\right)\left(\sum_{t=1}^{T} r_t^{(i)}\right)
\end{aligned}
$$

No dependence on starting and transition probability of the environment, thus being model-free!

Monte Carlo Approximation!
REINFORCE algorithm [15]

Policy Gradient Methods

Let us substitute $\mathbb{P}_\theta(\boldsymbol{\tau}) = \mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)$ $\quad R(\boldsymbol{\tau}) = \sum_{t=1}^{T} R_t(A_t,S_t)$

We have

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau})\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log\left(\mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta\left(\log \mathbb{P}_0(S) + \sum_{t=1}^{T}\log\pi_\theta(A_t|S_t) + \sum_{t=1}^{T}\log\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(A_t|S_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(a_t^{(i)}|s_t^{(i)})\right)\left(\sum_{t=1}^{T} r_t^{(i)}\right)
\end{aligned}
$$

No dependence on starting and transition probability of the environment, thus being model-free!

Monte Carlo Approximation!
REINFORCE algorithm [15]

Policy Gradient Methods

Let us substitute $\mathbb{P}_\theta(\boldsymbol{\tau}) = \mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)$ $\quad R(\boldsymbol{\tau}) = \sum_{t=1}^{T} R_t(A_t,S_t)$

We have

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau})\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log\left(\mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta\left(\log \mathbb{P}_0(S) + \sum_{t=1}^{T}\log\pi_\theta(A_t|S_t) + \sum_{t=1}^{T}\log\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(A_t|S_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(a_t^{(i)}|s_t^{(i)})\right)\left(\sum_{t=1}^{T} r_t^{(i)}\right)
\end{aligned}$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(a_t^{(i)}|s_t^{(i)})\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)}\right)\right)$$

No dependence on starting and transition probability of the environment, thus being model-free!

Monte Carlo Approximation! REINFORCE algorithm [15]

Reward-to-go version: *my action today can not change rewards in the past*!

Policy Gradient Methods

Let us substitute $\mathbb{P}_\theta(\boldsymbol{\tau}) = \mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)$ $\quad R(\boldsymbol{\tau}) = \sum_{t=1}^{T} R_t(A_t,S_t)$

We have

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau})\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log\left(\mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta\left(\log \mathbb{P}_0(S) + \sum_{t=1}^{T}\log\pi_\theta(A_t|S_t) + \sum_{t=1}^{T}\log\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(A_t|S_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(a_t^{(i)}|s_t^{(i)})\right)\left(\sum_{t=1}^{T} r_t^{(i)}\right)
\end{aligned}$$

No dependence on starting and transition probability of the environment, thus being model-free!

Monte Carlo Approximation! REINFORCE algorithm [15]

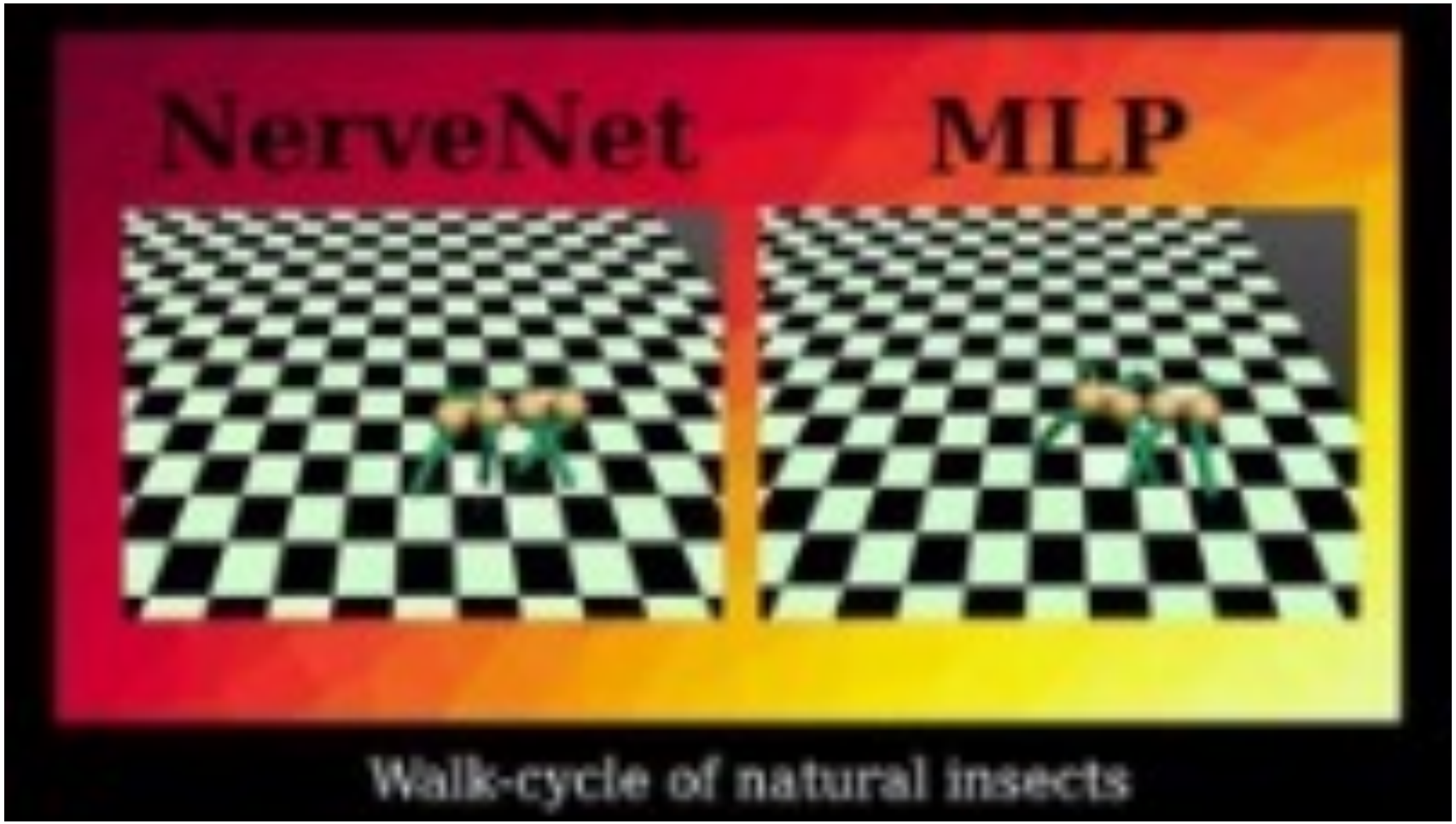
$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(a_t^{(i)}|s_t^{(i)})\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)}\right)\right)$$

Reward-to-go version: *my action today can not change rewards in the past*!

One often uses control variate method to reduce the high variance of policy gradients.

Demo

Simulated Continuous Control in Mujoco using PPO [16] (an advanced policy gradient method) and graph neural networks [17]:



References

[1] https://coolinventor.com/wiki/index.php?title=Beginner%27s_Guide_to_Deep_Reinforcement_Learning

[2] https://gym.openai.com/envs/Walker2d-v1/

[3] https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii

[4] https://medium.com/zerone-magazine/the-single-instance-where-man-triumphed-over-ai-the-google-deepmind-challenge-match-1d6af01005a

[5] https://ai.googleblog.com/2021/04/multi-task-robotic-reinforcement.html

[6] https://ai.googleblog.com/2021/01/google-research-looking-back-at-2020.html

[7] https://engineering.princeton.edu/news/2020/11/17/machine-learning-guarantees-robots-performance-unknown-territory

[8] https://siegel.work/blog/RLModelBased/

[9] Murphy, K.P., 2023. Probabilistic machine learning: Advanced topics. MIT Press.

[10] Watkins, C.J. and Dayan, P., 1992. Q-learning. Machine learning, 8, pp.279-292.

[11] Sutton, R.S. and Barto, A.G., 1998. Reinforcement Learning: An Introduction.

[12] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[13] https://www.davidsilver.uk/wp-content/uploads/2020/03/FA.pdf

References

[14] https://www.davidsilver.uk/wp-content/uploads/2020/03/pg.pdf

[15] Williams, R.J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Reinforcement learning, pp.5-32.

[16] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

[17] Wang, T., Liao, R., Ba, J. and Fidler, S., 2018. Nervenet: Learning structured policy with graph neural networks. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada (Vol. 30).

Questions?