CPSC 340:
Machine Learning and Data Mining

Reinforcement Learning -- Bonus Lecture

Ben Norman
(slides adapted from Daniele Reda, and Helen Zhang)
Spring 2022 (2021W2)

Talk Content

- Part 1: What, Why, When
  - What is Reinforcement Learning, RL?
  - Why RL? Why not RL? When to do RL?

- Part 2: How Reinforcement Learning Works

- Part 3: Applications of RL!

Part 1

- What is RL?
- Why RL?
- When RL?

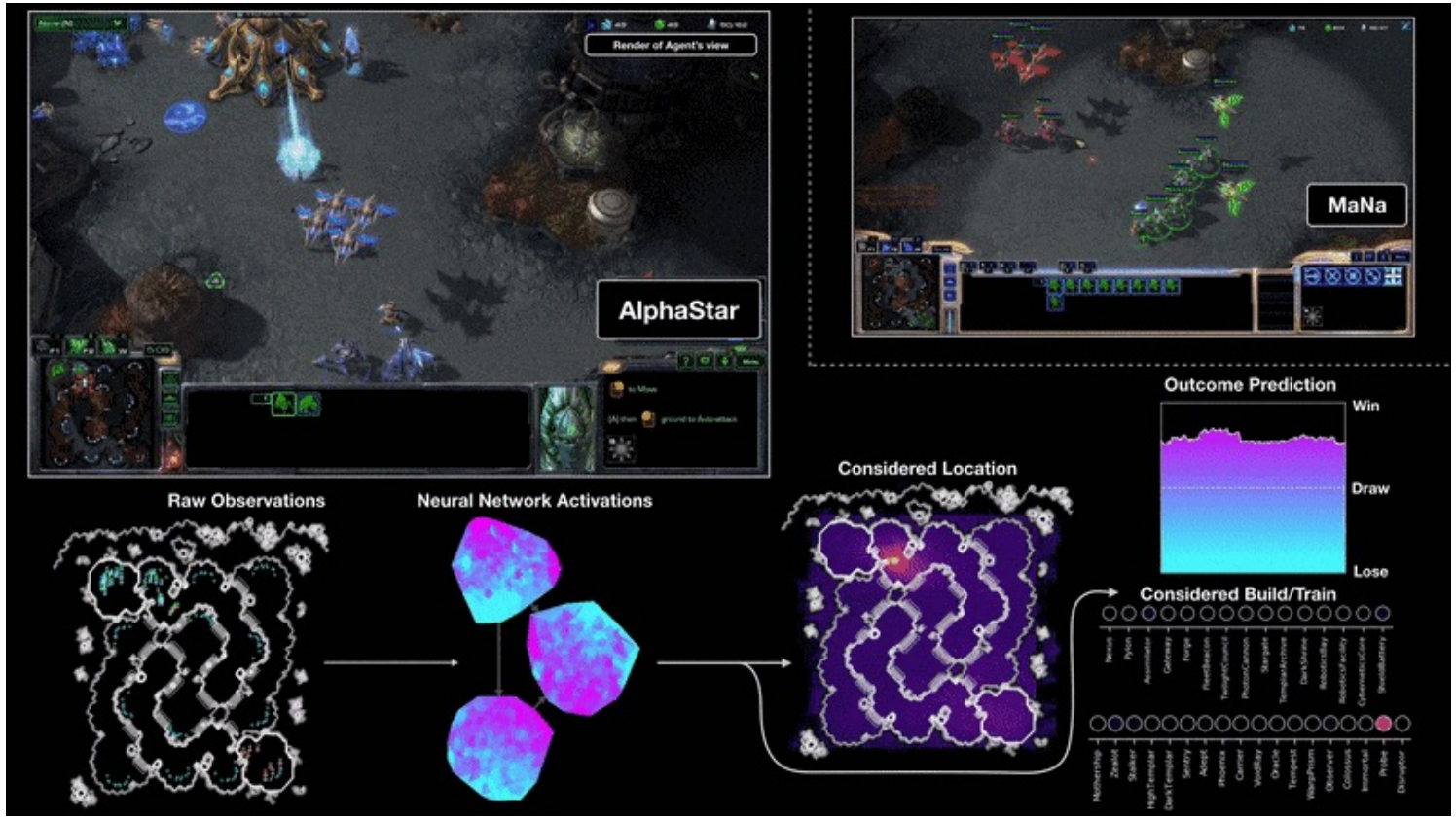
What are the problems we want solve?

Having a Robot Walk

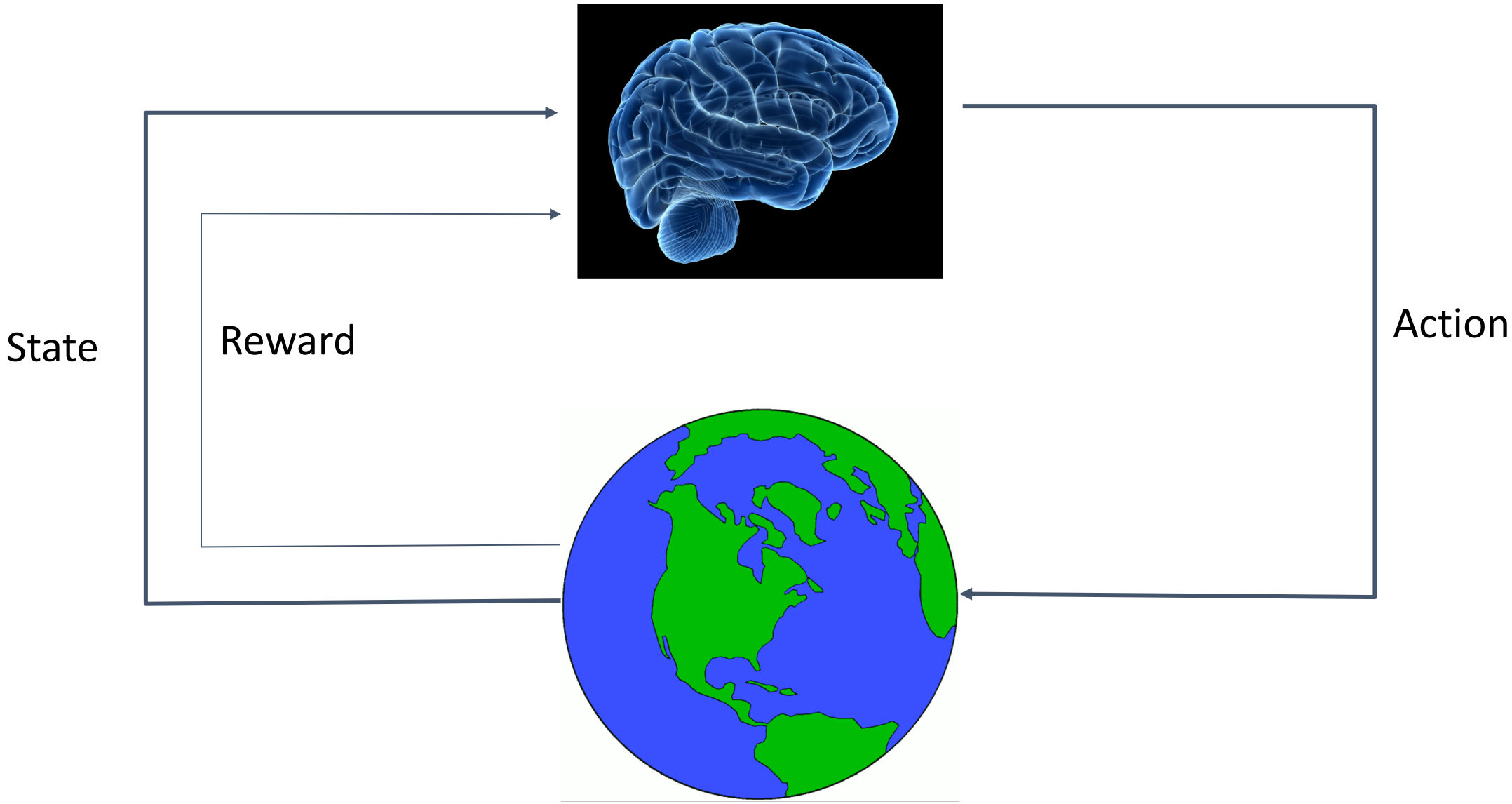
Investing

Chess

Starcraft



Problem Setting



What is the nature of these problems?

The agent gets information, and must take actions

The info:

State: Your limb positions, Your stock portfolio, The chess board, The Starcraft Game State

Output:

The next **action**: Muscle Activations, Buying and Selling Stock, A single chess move, Mouse clicks and Key presses

Note: we can have Complex Time and Action Dependencies

How can we solve these problems?

Two main approaches:

- We can do a form of **Supervised Learning** called **Imitation Learning** and **imitate** and **distill** expert play. (State is your **x**, Action is your **y**)

Effective and Popular

- We can do **Reinforcement Learning** and **explore** and **discover** play

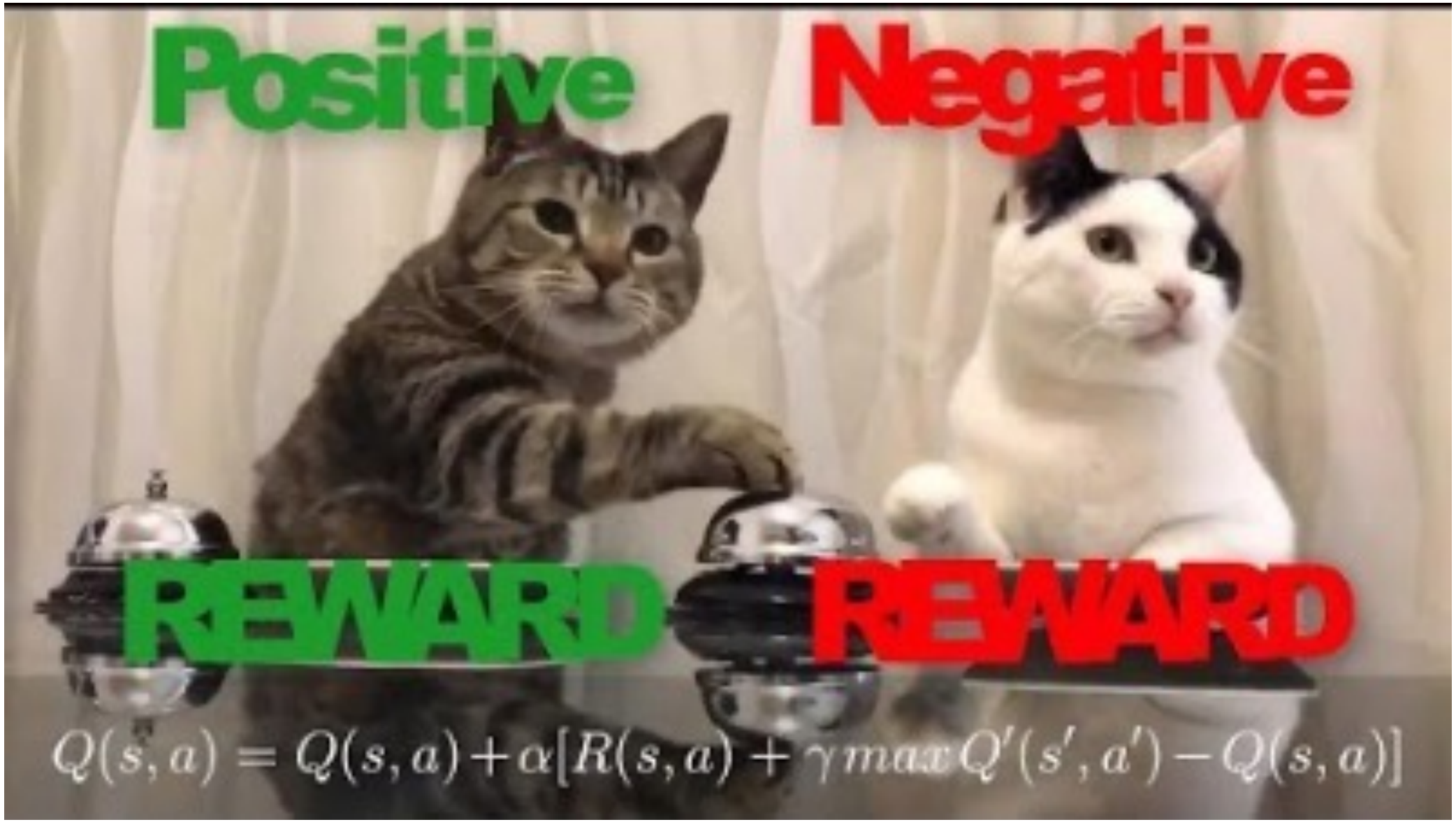
Effective and Popular

Types of Feedback

Imagine you take a mock test for your mid terms:

- **Model Answers Feedback**, “question 1 part a) you should make the point that …” : **Supervised Learning**
- **Score Feedback**, “you got 200”: **Reinforcement Learning**
- **No Feedback,** “ ” : **Unsupervised**, or Self-Supervised Learning

Learning from scores is far harder than learning from model answers.


Positive
Negative
REWARD
REWARD
$Q(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma max Q'(s',a') - Q(s,a)]$

Summary

- How is Reinforcement Learning, RL, different from Supervised Learning?

In **Supervised** Learning you have **Labels**

In **Reinforcement** Learning you have **Score, or Reward**

Imitation Learning **copies** and **distils** the expert plays

Reinforcement Learning **explores**, and **discovers** strategies

Why Reinforcement Learning?

- No need for Labels

Score is **cheap**, and can often be **generated** by algorithm

- Beyond distilling, **discovery**. The potential for **superhuman** performance!

- Can be more **robust**

Note: imitation learning *can* be better than human too

(imagine 5 experts each amazing at one specialty, the best imitation is amazing at all 5 specialties)

Why **not** Reinforcement Learning?

- **Scoring** can be **difficult,** sometimes having labels is easier,

e.g. 

- RL can have **low** sample **efficiency**, often run in simulations, but simulating can be hard

RL can give **super**-human performance

AlphaGo, and its successors, AlphaGo Zero, AlphaZero, MuZero… really are superhuman, in a *meaningful* way

- discovered new corner sequences (Joseki)

Not aware of any meaningful super-human imitation learning

Part 2

- How does Reinforcement Learning work?

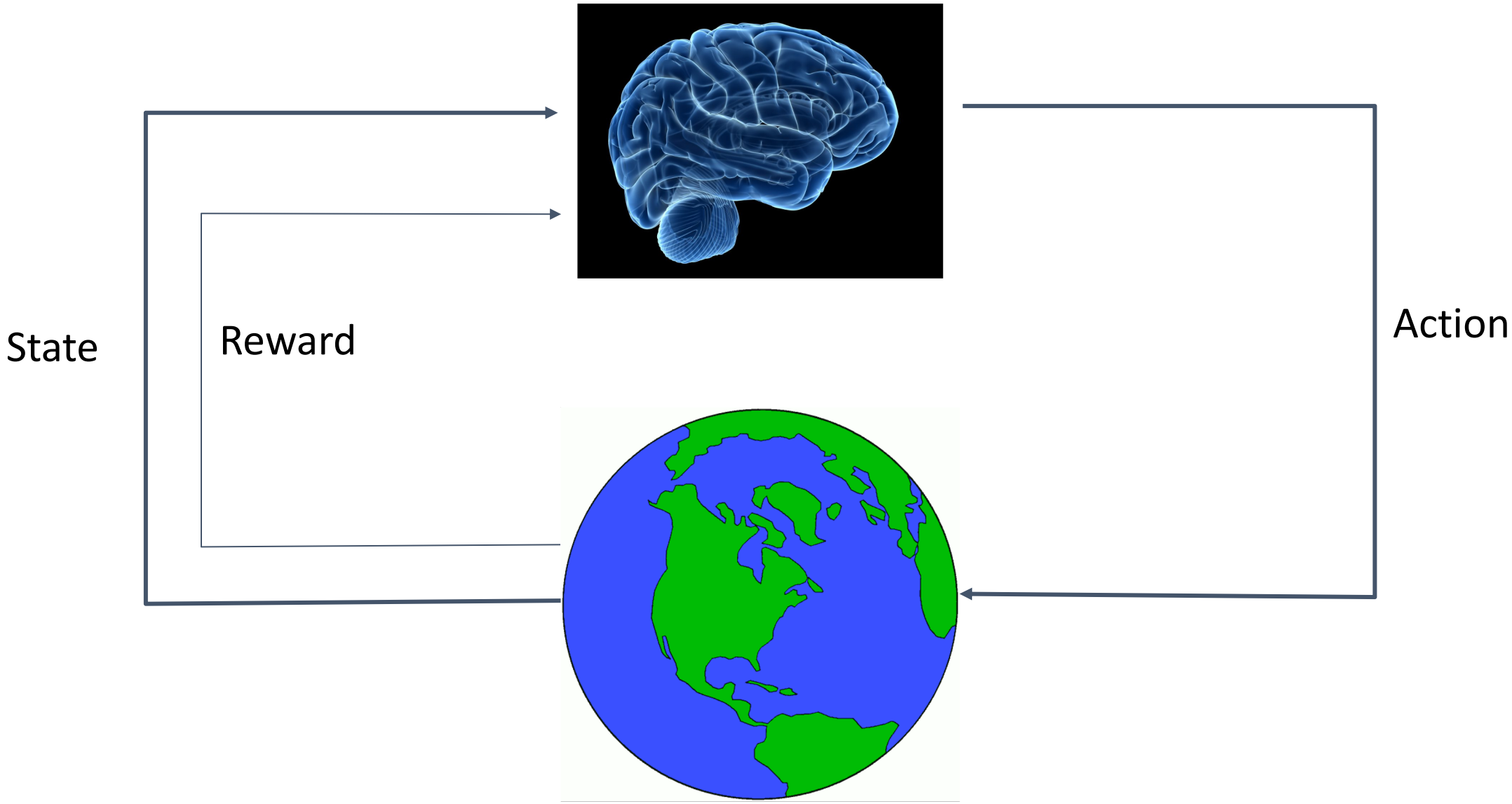
How does Reinforcement Learning Work?

Two basic approaches:

- Value Function Estimation : estimate the **long-term** value of states (and actions)

- Policy Gradient : policy in state **s** do action **a,** update our policy to increase likelihood of high scoring policies

Problem Setting



Goal

- maximize total reward

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T$$

- T to infinity
- Why discount factor?
  - Convergence, ‘it works mathematically’
  - Estimates of Future Reward are more uncertain.

What is the value of a state?

Given a policy, π, we can define the Value function of a state

$$V_\pi(s) = \mathrm{E}_\pi(R_t | \, S_t = s)$$

We want the value under the optimal policy, π_*

$$V_*(s) = \mathrm{E}_*(R_t | \, S_t = s)$$

Satisfies a ‘Bellman Equation’:

$$V^*(s) = \max_\alpha \left[R(s, \alpha) + \gamma \sum_{s'} p(s' | \, s, \alpha) V^*(s') \right]$$

How to choose action though?

Idea: **value state action pairs**

Definition:

$$Q_\pi(s,a) = \mathrm{E}_\pi(R_t | \, S_t = s, A_t = a)$$

Bellman Equation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} p(s' | \, s,a) \max_\alpha (Q^*(s',\alpha))$$

The Q-Function specifies a policy

	Actions						
States		0	1	2	3	4	5
	0	-1	-1	-1	-1	0	-1
	1	-1	-1	-1	0	1	100
	2	-1	-1	-1	0	-1	-1
	3	-1	0	0	-1	0	-1
	4	0	-1	-1	0	-1	100
	5	-1	0	-1	-1	0	100

How to learn it?

There is a theorem: Policy Improvement Theorem

Given any (tabular) Q-function, if we update our Q-function, through the Bellman Equations, the policy will improve

$$Q^{n+1}(s,a) = R(s,a) + \gamma \max_{\alpha} \left(Q^{n}(s',\alpha)\right)$$

Note, that in expectation is gives the Bellman Equation

Problem? How do we ensure exploration?

The simplest possible idea:

Epsilon greedy: take the best action $1 - \varepsilon$ of the time, random action ε of the time

Q-learning

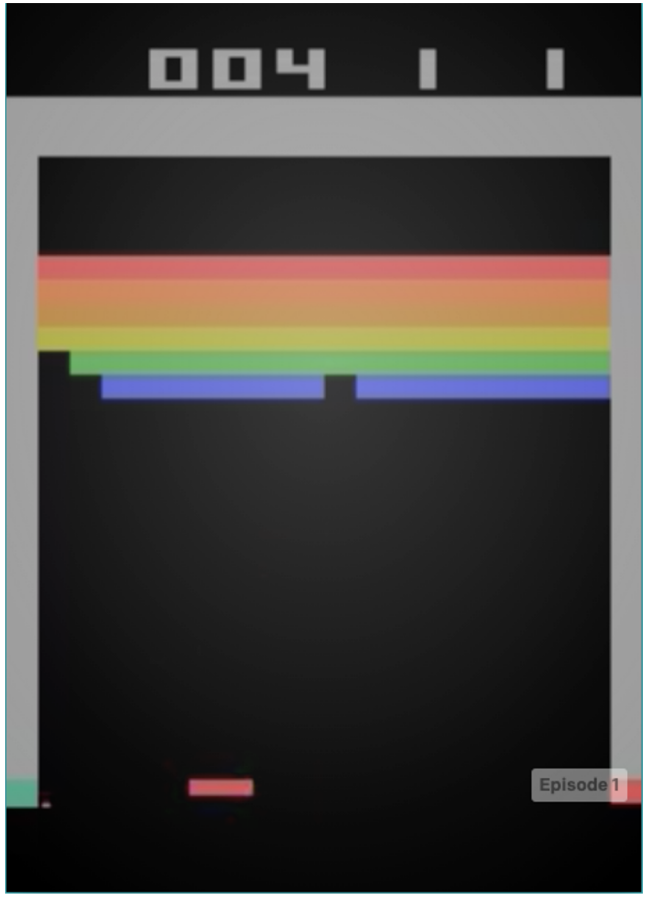
Initialize Q-table with random values.

1. Choose action a to perform in current state s. (ε-greedy)
2. Perform a and receive reward R(s,a).
3. Observe new state S(s,a).
4. Update Q-table.

$$Q'(s,a) \leftarrow \mathcal{R}(s,a) + \gamma \max_{\alpha} \left\{ Q'(\mathcal{S}(s,a), \alpha) \right\}$$

PROBLEM:
TABLE CAN EASILY EXPLODE IN DIMENSIONS

Let’s look at an example: ATARI



State (the actual image)
84x84x4 pixels (gray-scale)

2 Actions
Left-Right

Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning.

Could we use a q-table?

Atari Breakout example:

State = raw pixels of last 4 frames (84x84) with 256 different possible values.

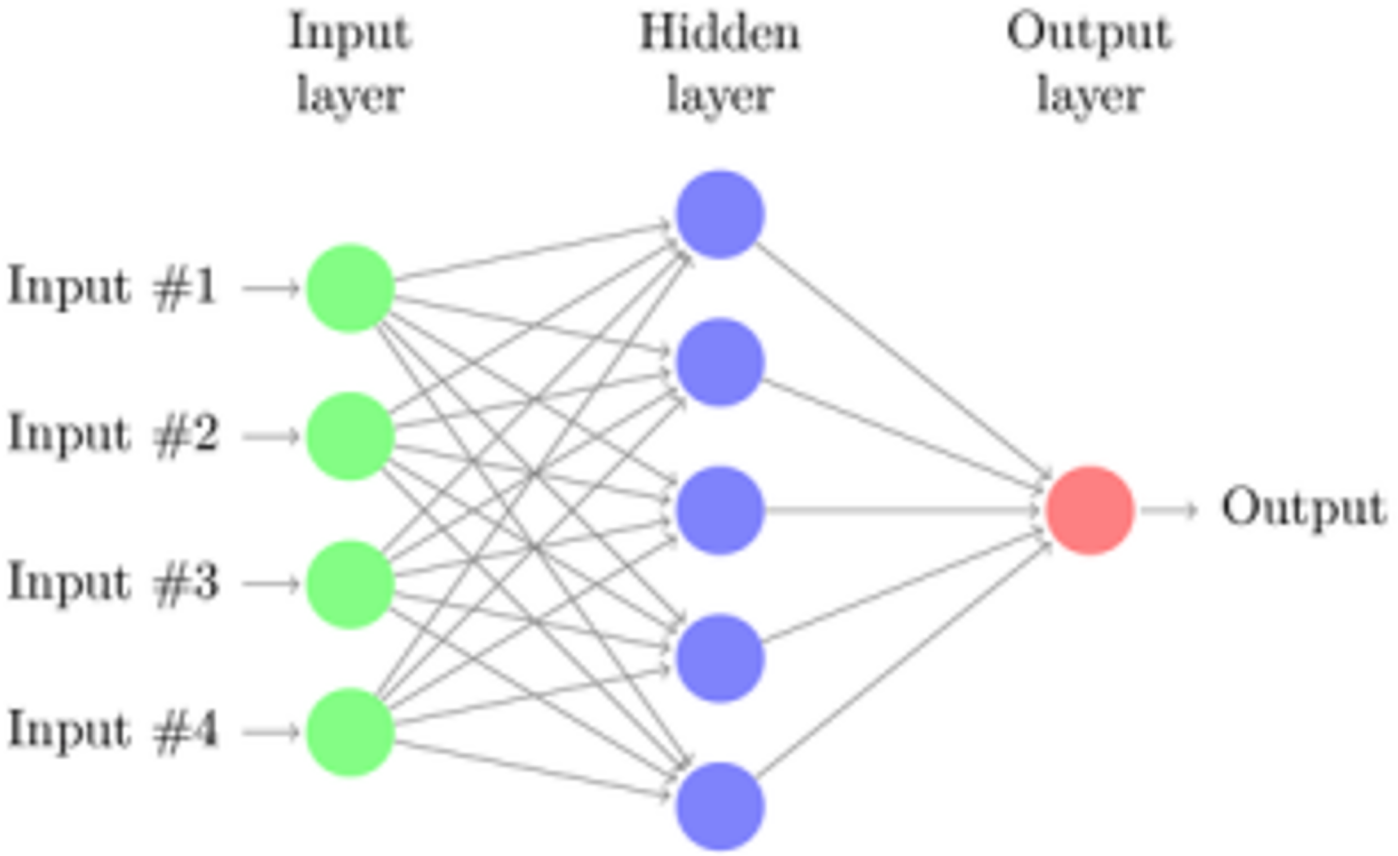
Actions = 2 actions available

(4x84x84x256)x2 = 14450688 different values

Solution : Functional Approximation

Neural networks!

Neural Networks



Neural Networks

Reward

Decay Rate

$$loss = \left(\underbrace{r + \gamma \max_{a`} \hat{Q}(s, a`)}_{\text{Target}} - \underbrace{Q(s, a)}_{\text{Prediction}} \right)^2$$

$$Q'(s,a) \leftarrow \mathcal{R}(s,a) + \gamma \max_{\alpha} \{Q'(\mathcal{S}(s,a), \alpha)\}$$

DQN Framework

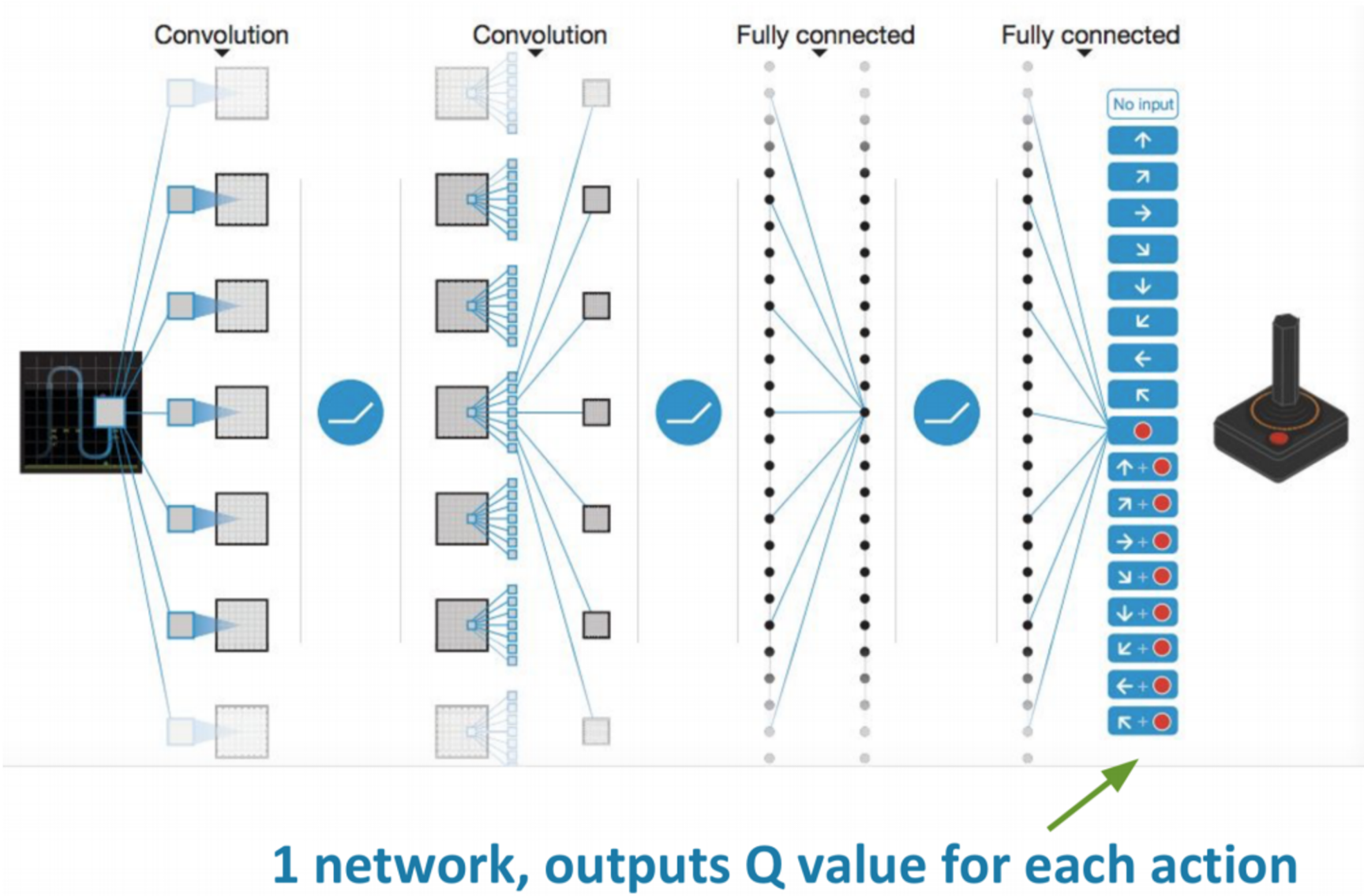


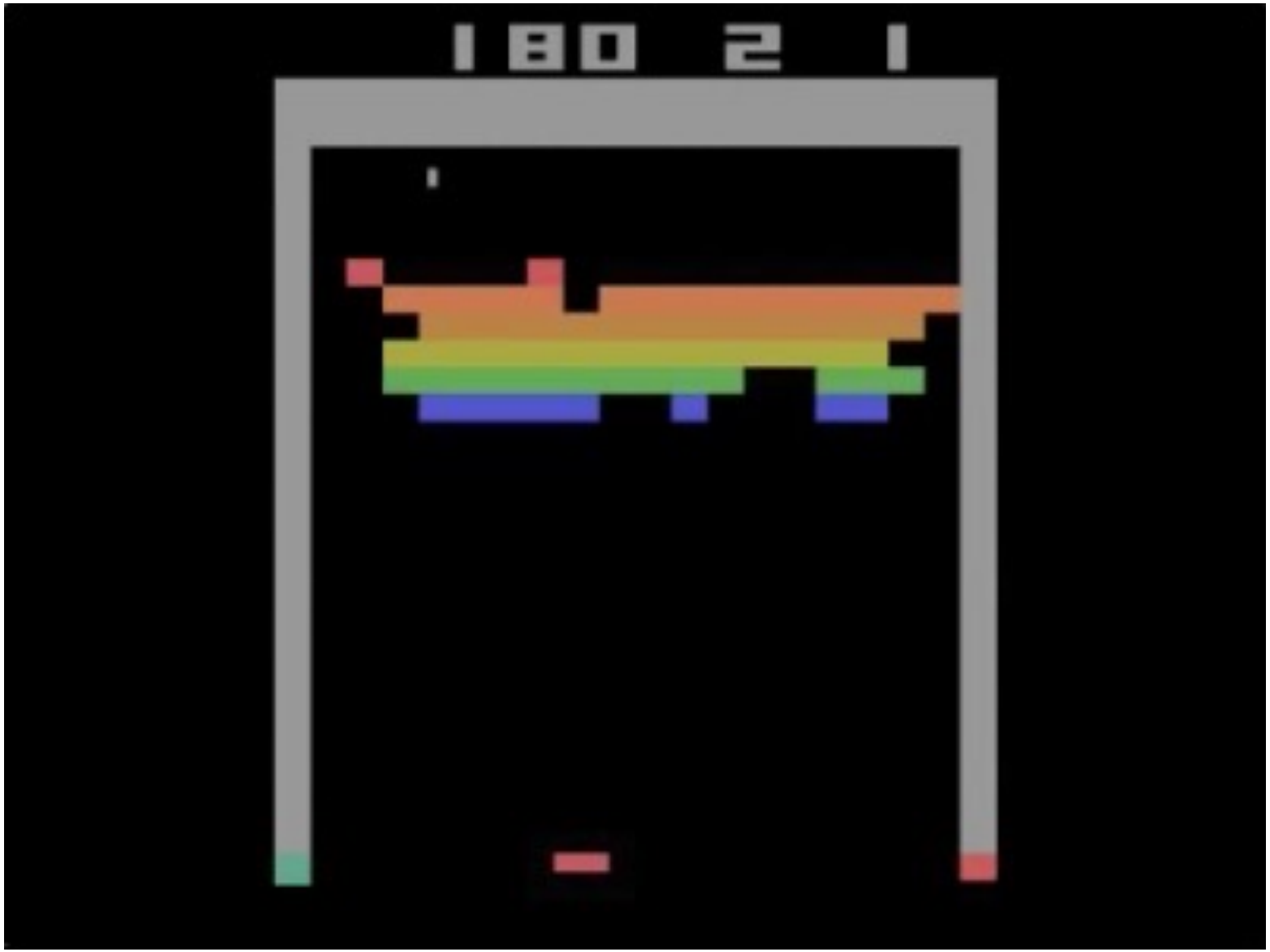
Image from Emma Brunskill's RL course CS234, Stanford

DQN Framework

Algorithm 1: DQN Pesudocode

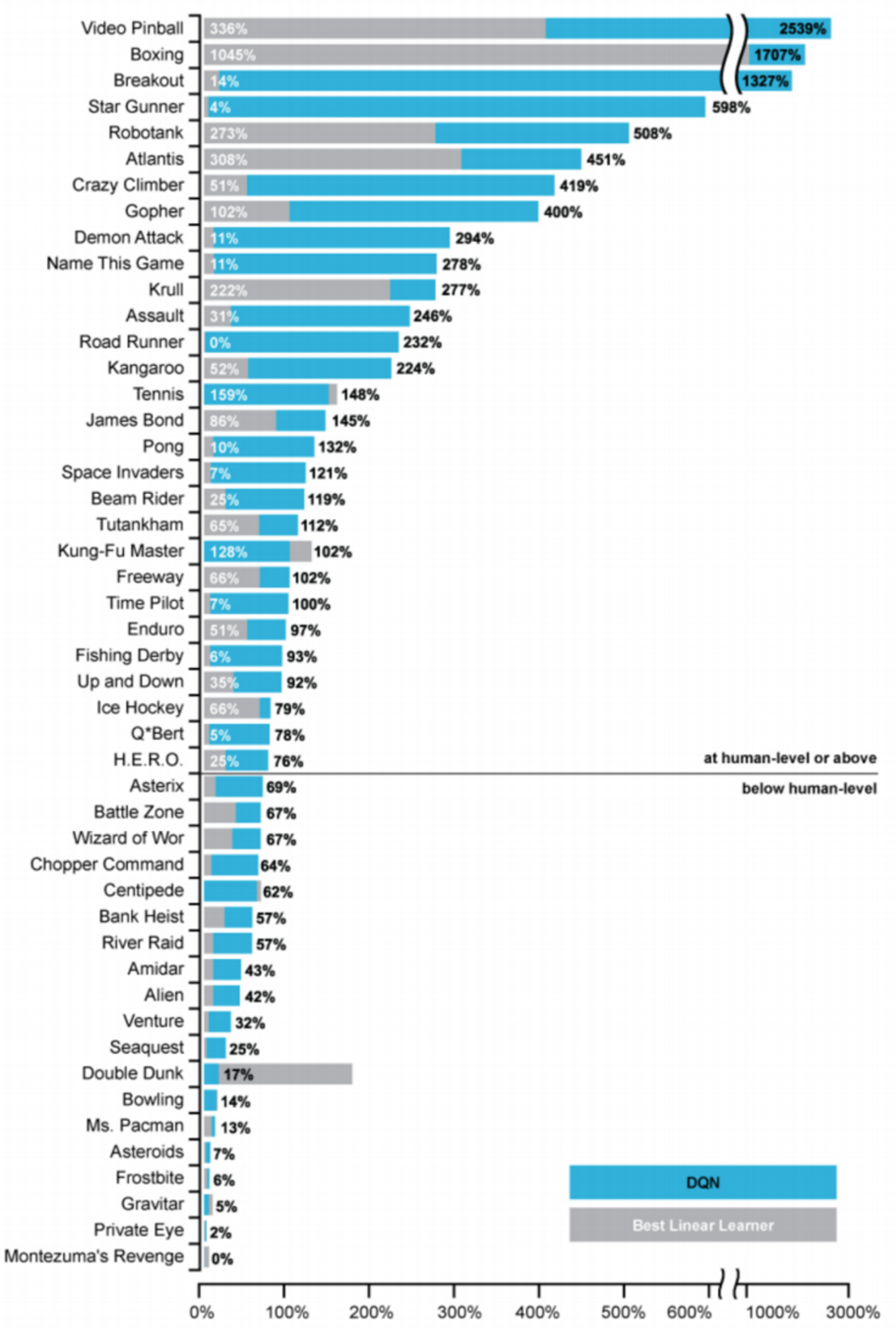
Randomly initialize neural network NN
Get initial state s_0
$t = 0$
while *1* **do**
    // ϵ-greedy strategy
    r = get random value from $(0, 1)$
    if $r > \epsilon$ **then**
        $a_t = NN(s_0)$
    else
        a_t = random action between the ones available
    end
    $s_{t+1}, r_t, done = environment(a_t)$
    if $done = True$ **then**
        $y = r_t$
    else
        $y = r + \gamma * \max_{a^i} NN(s_{t+1}$
    end
    Do gradient descent on $y - NN(s_t, a_t)$ to update weights of NN
    $t = t + 1$
end

Famous successes of RL: Atari Breakout



Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning.

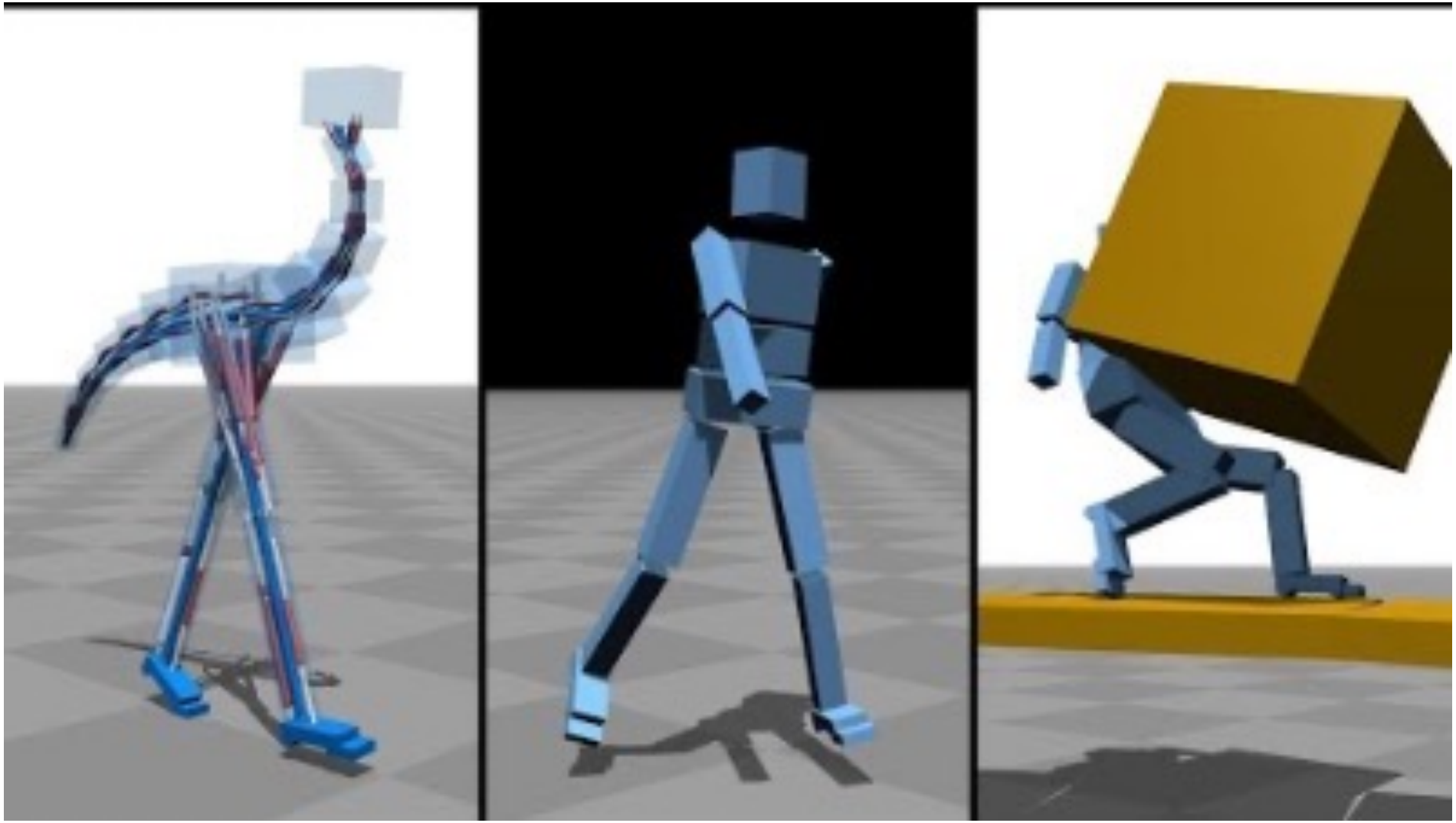
Image from Emma Brunskill's RL course CS234, Stanford



Part 3

- Applications of RL!

Flexible Muscle-Based Locomotion for Bipedal Creatures

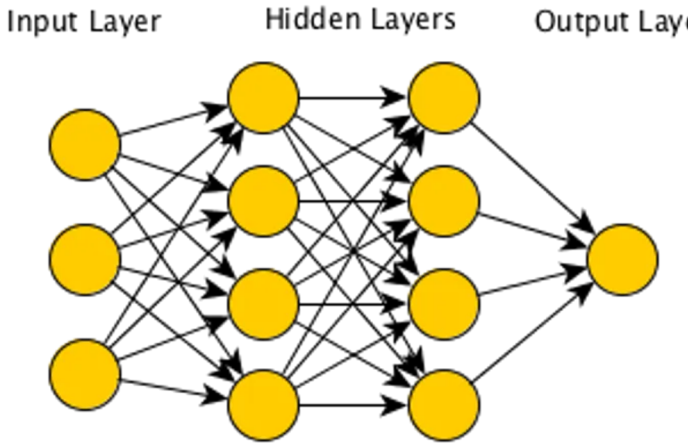


A real world example:
Learning to drive with Reinforcement Learning

Kendall, Alex, et al. "Learning to drive in a day."

Learning to drive with RL

Steering & Speed Measurement



Steering & Speed Command

Reward: forward distance

Terminate when it goes out of the lane

Kendall, Alex, et al. "Learning to drive in a day."

Kendall, Alex, et al. "Learning to drive in a day."

Where to go from here?

- openAI Spinning Up RL
- Sutton book
- David Silver UCL Course