Today: Outline

- **Reinforcement Learning**
- **Exam Guidelines**
- **Breakout Session**

- Reminder:
  - Exam Jun 22 in class
    (and ~12 hrs before for remote only students)
- Announcements:
  - Tomorrow: Project Guidelines / SCC Tutorial
  - Problem Set 1 Solutions are posted

Reinforcement Learning

Types of learning

Supervised

Unsupervised

Reinforcement

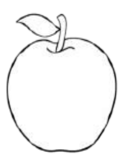
Classes of Learning Problems

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn function to map
$$x \to y$$

Apple example:



This thing is an apple.

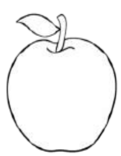
Classes of Learning Problems

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn function to map
$$x \rightarrow y$$

Apple example:



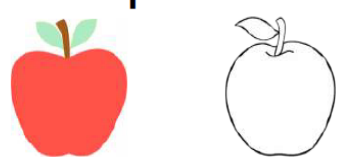
This thing is an apple.

Unsupervised Learning

Data: x
x is data, no labels!

Goal: Learn underlying structure

Apple example:



This thing is like the other thing.

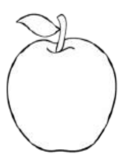
Classes of Learning Problems

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn function to map
$$x \rightarrow y$$

Apple example:



This thing is an apple.

Unsupervised Learning

Data: x
x is data, no labels!

Goal: Learn underlying structure

Apple example:

This thing is like the other thing.

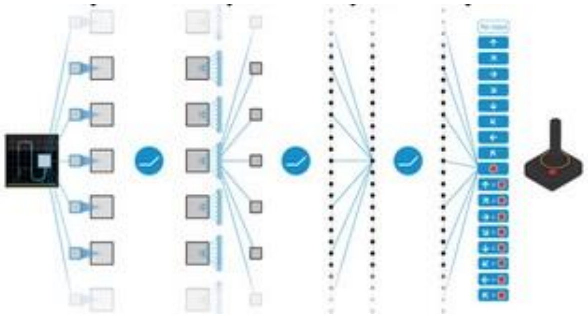
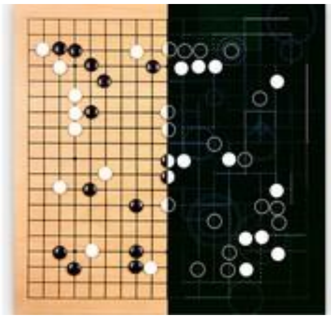
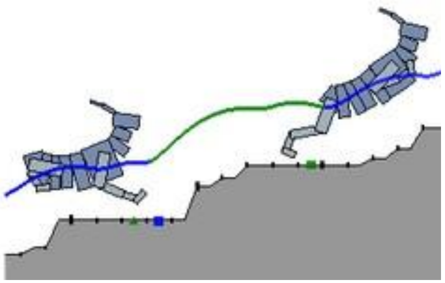
Reinforcement Learning

Data: state-action pairs

Goal: Maximize future rewards over many time steps

Apple example:

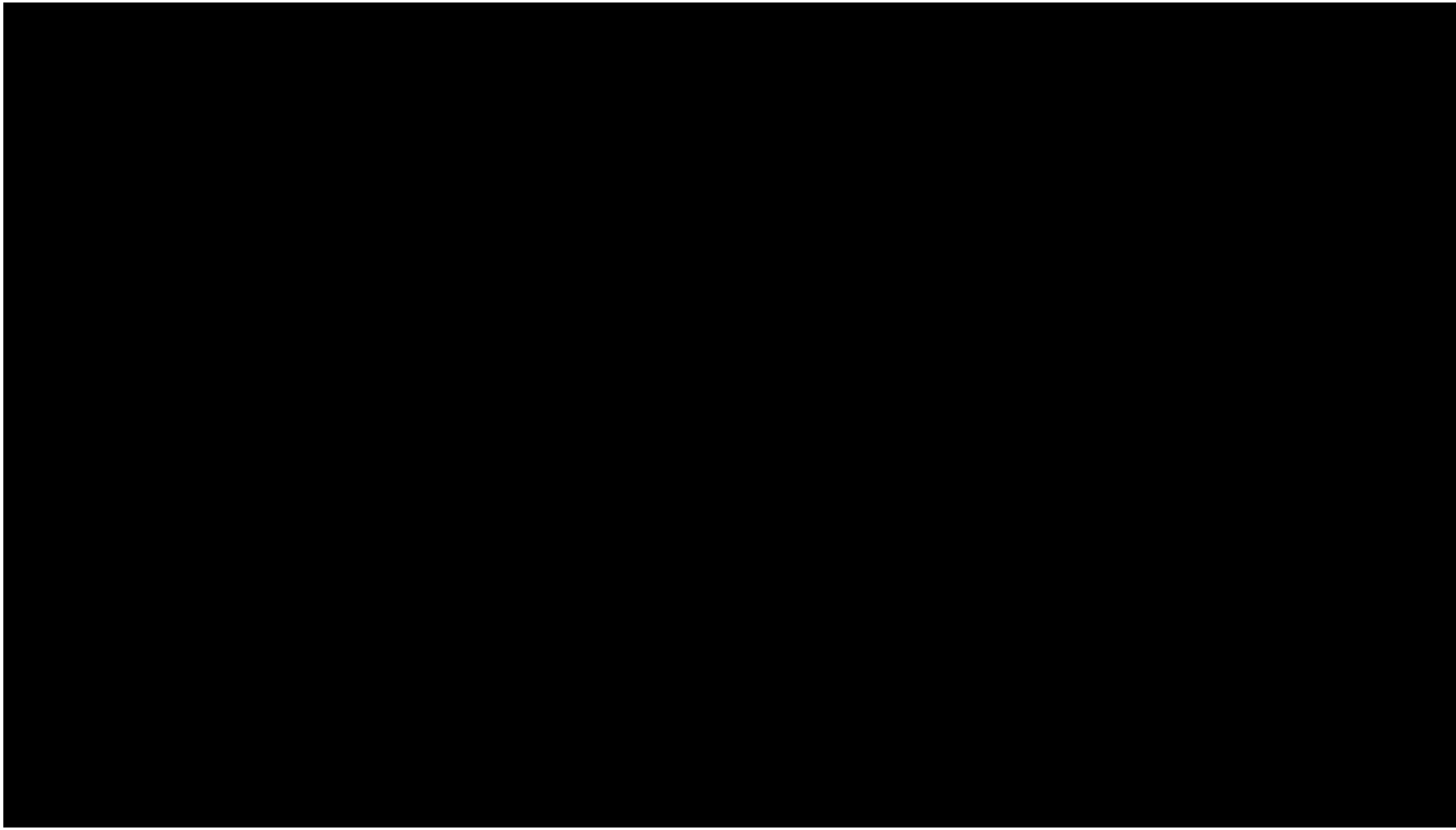
Eat this thing because it will keep you alive.



Reinforcement Learning

- Plays Atari video games
- Beats human champions at Poker and Go
- Robot learns to pick up, stack blocks
- Simulated quadruped learns to run

Deep Mind's bot playing Atari Breakout

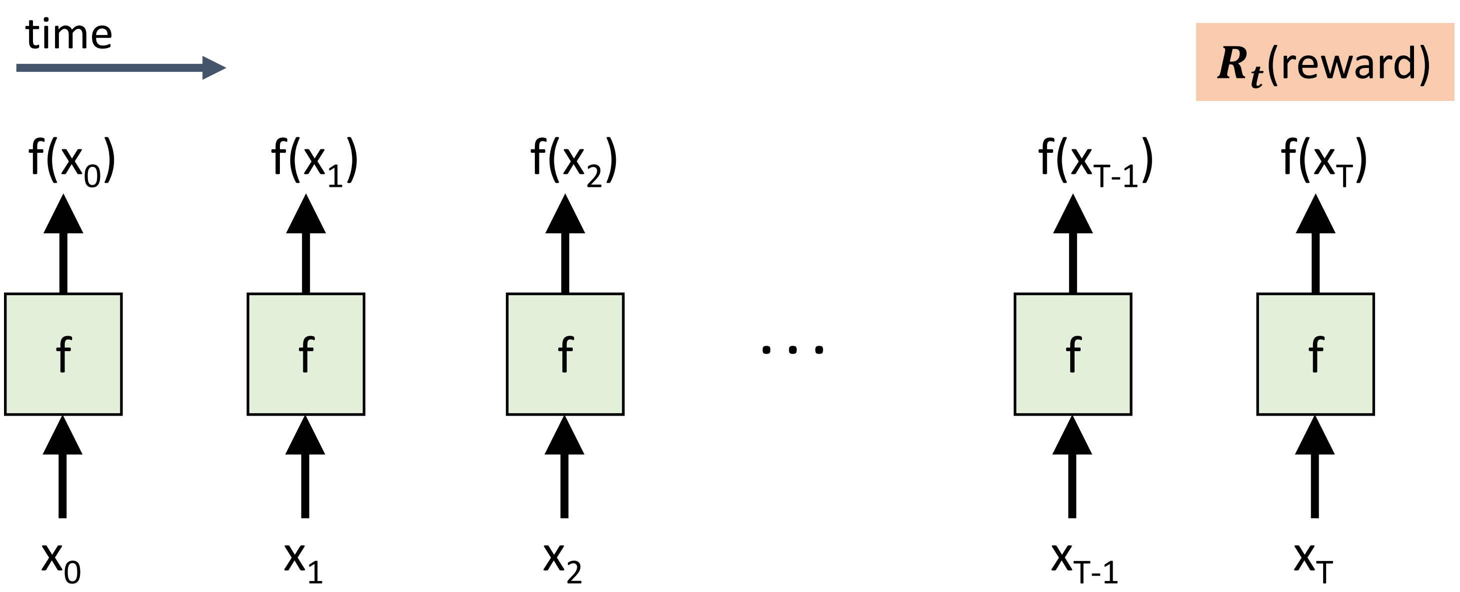


https://www.youtube.com/watch?v=TmPfTpjtdgg

Reinforcement learning

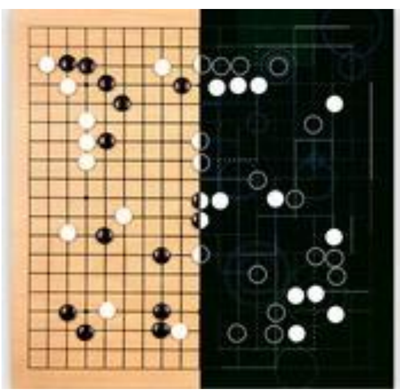
- agent receives input *x*, chooses action
- gets *R* (reward) after *T* time steps
- actions affect the next input (state)

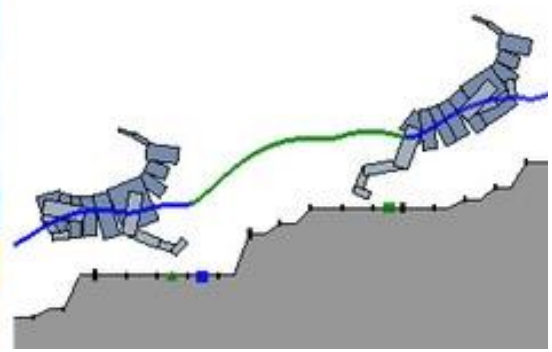
Reinforcement learning:



Input is the "world's" state

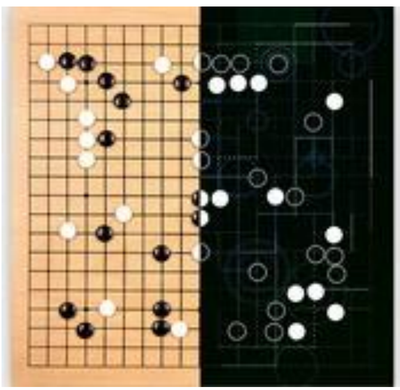
- Current game board layout
- Picture of table with blocks
- Quadruped position and orientation

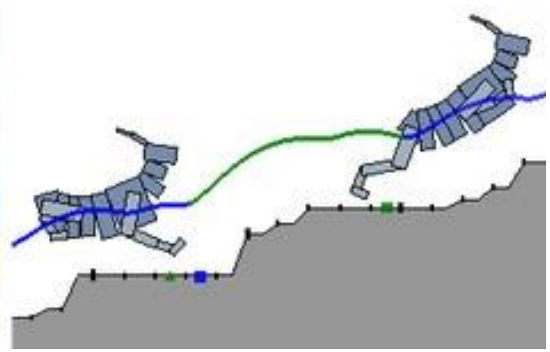




Output is an action

- Which game piece to move where (discrete)
- Orientation and position of robot arm (continuous)
- Joint angles of quadruped legs (continuous)





Actions affect state!

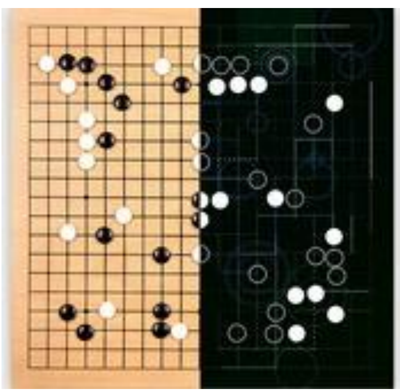
action→reward

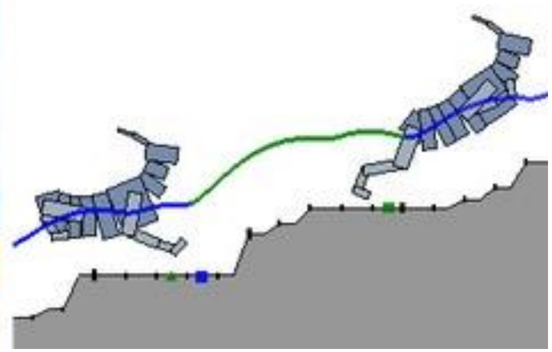
Only some actions lead to rewards

Some rewards are negative

Reward examples

- Wining the game (positive)
- Successfully picking up block (positive)
- Falling (negative)

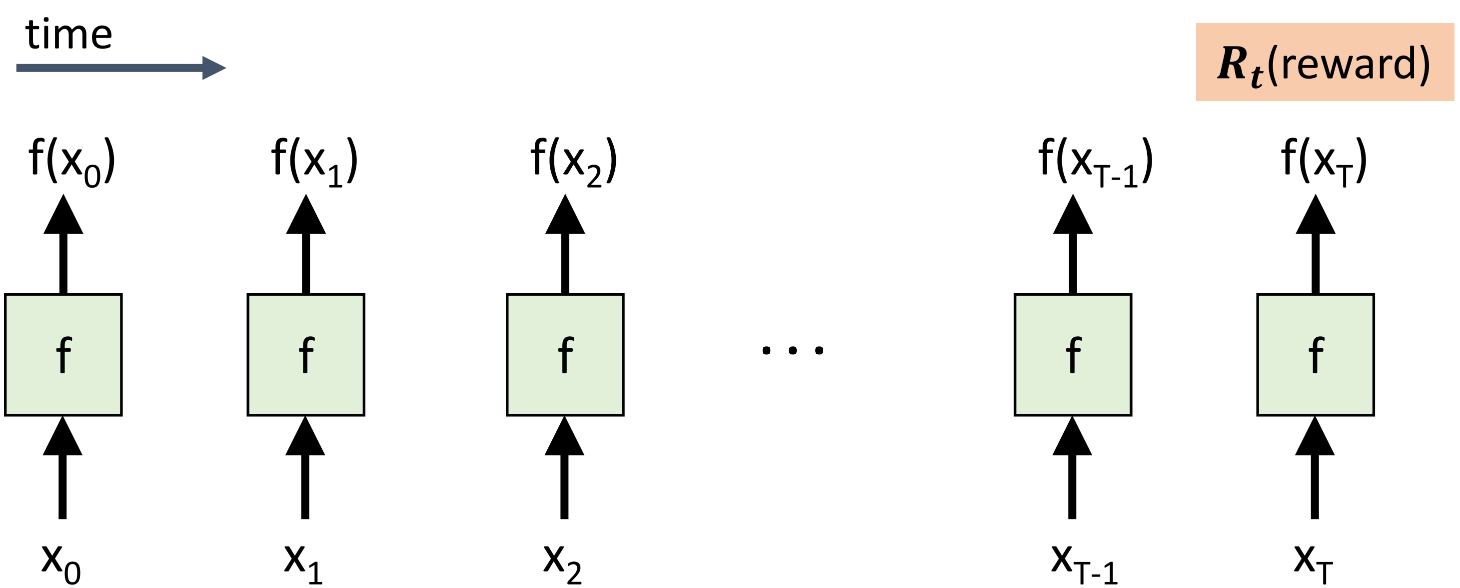




Goal of reinforcement learning

- Learn to predict actions that maximize future rewards
- Need a new mathematical framework

Reinforcement learning:



Reinforcement Learning (RL): Key Concepts

AGENT

Agent: takes actions.

Reinforcement Learning (RL): Key Concepts

AGENT

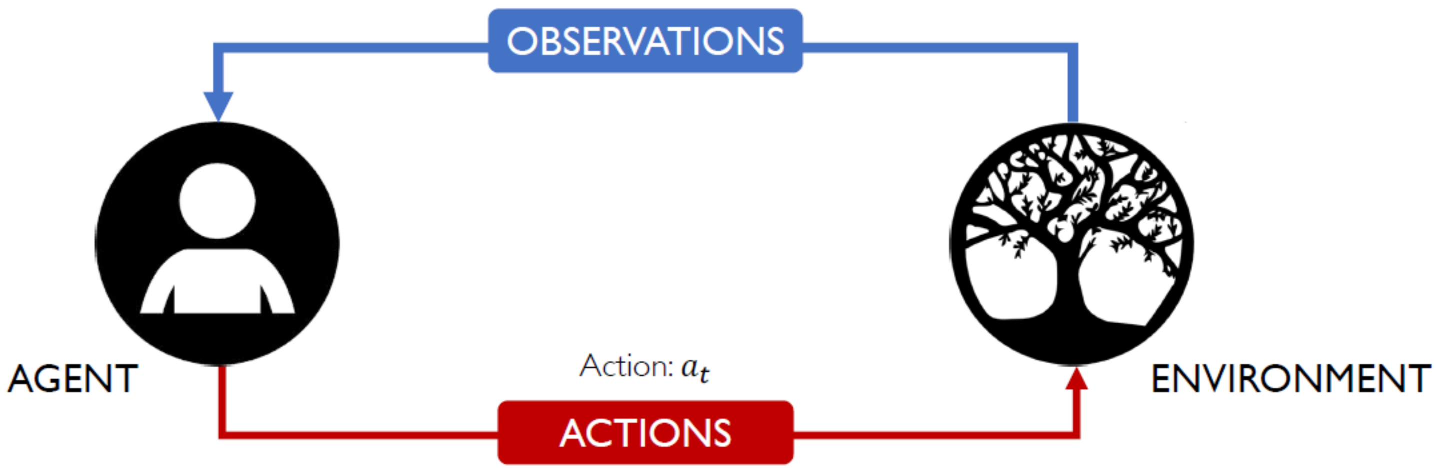
ENVIRONMENT

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

Reinforcement Learning (RL): Key Concepts

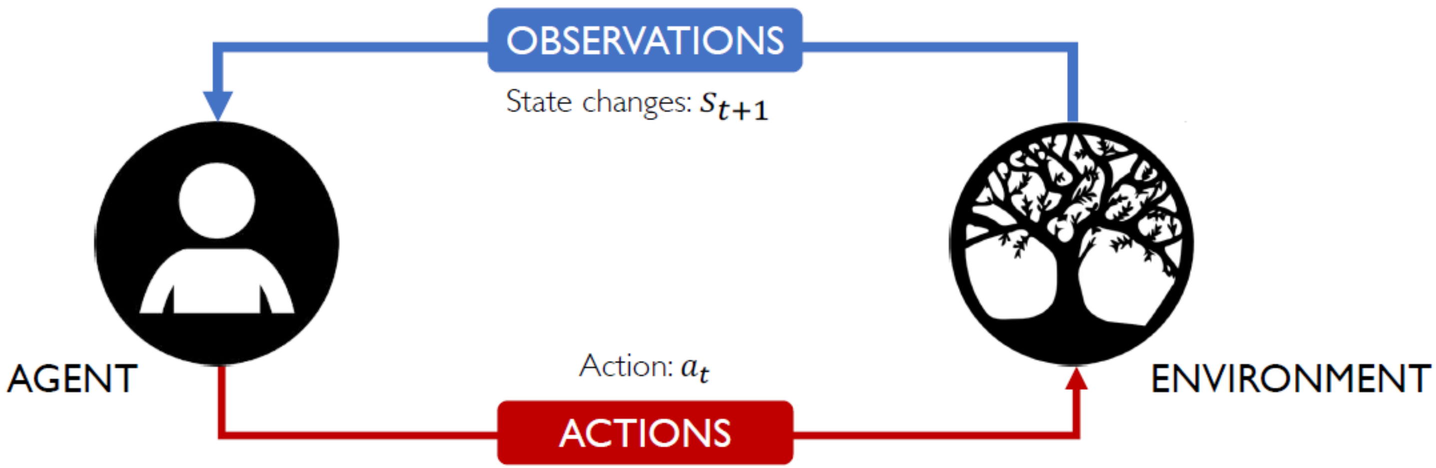
Action: a move the agent can make in the environment.

Reinforcement Learning (RL): Key Concepts



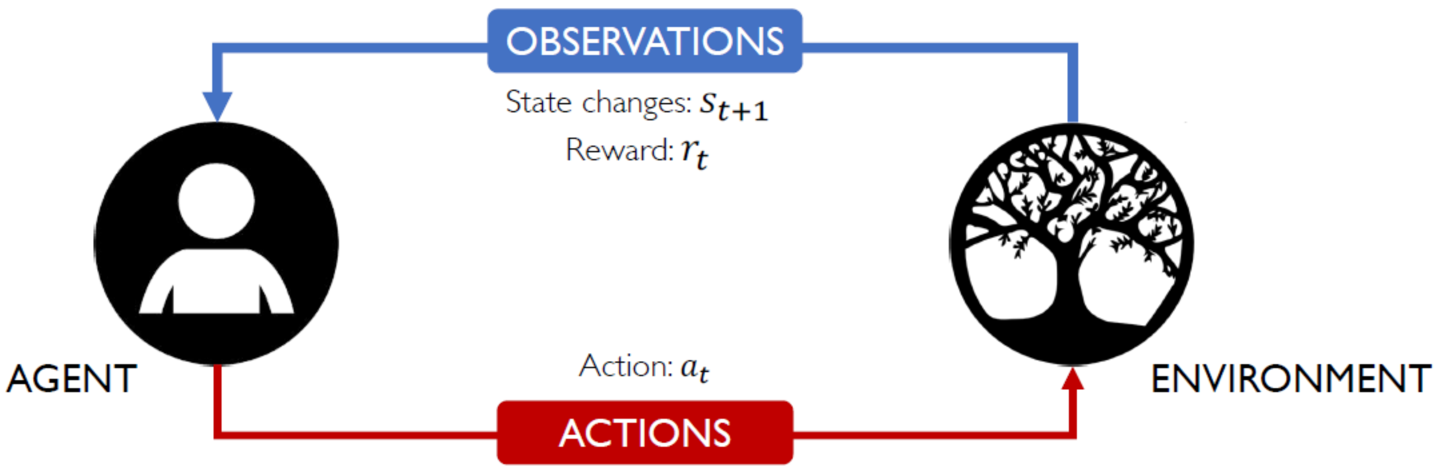
Observations: of the environment after taking actions.

Reinforcement Learning (RL): Key Concepts



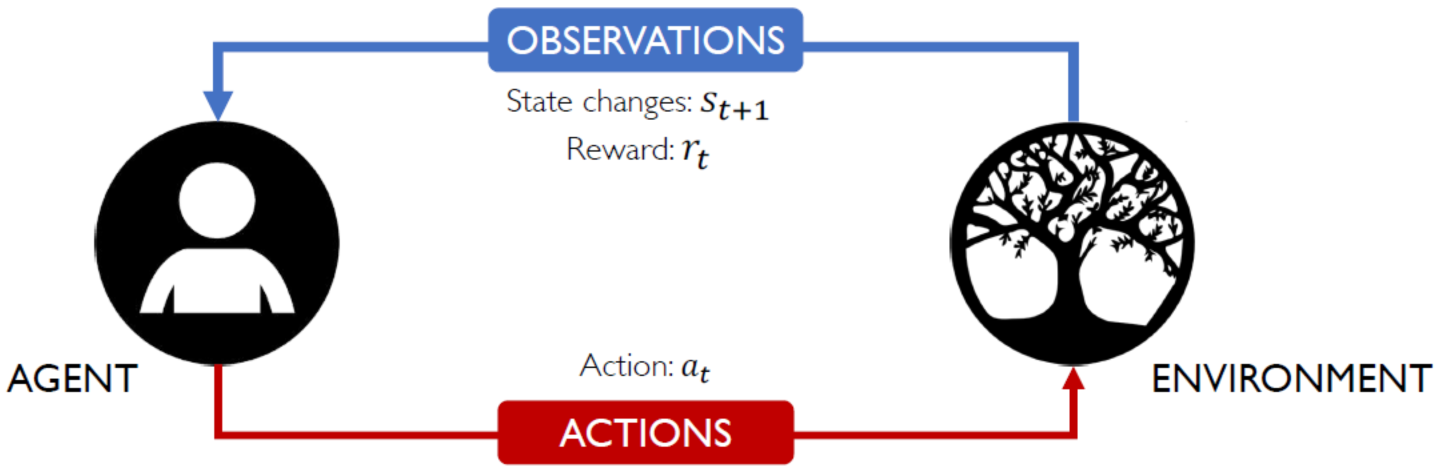
State: a situation which the agent perceives.

Reinforcement Learning (RL): Key Concepts



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

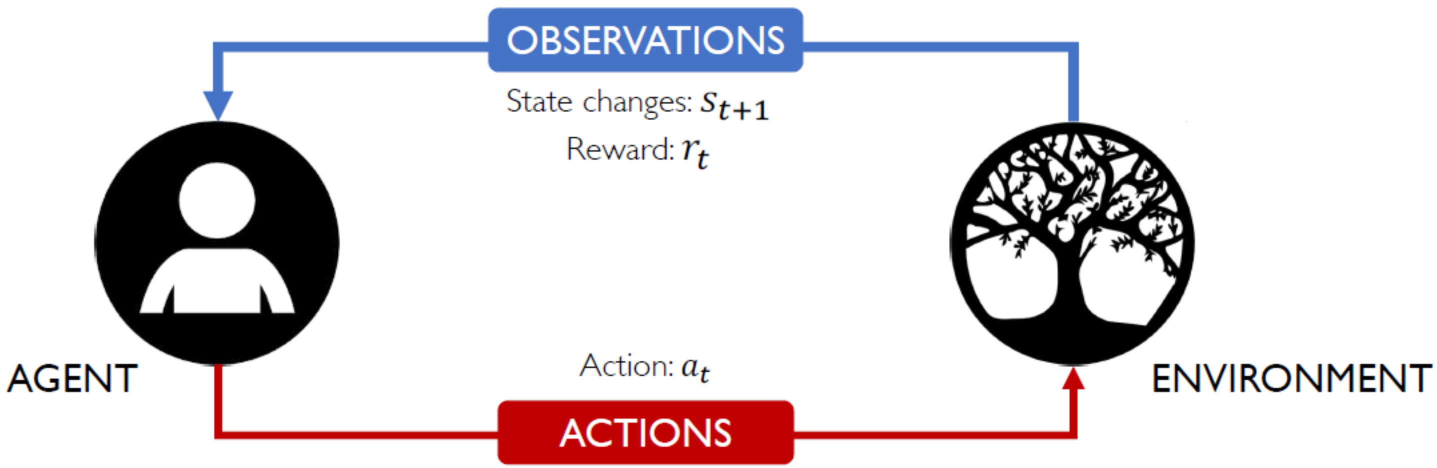
Reinforcement Learning (RL): Key Concepts



Total Reward

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

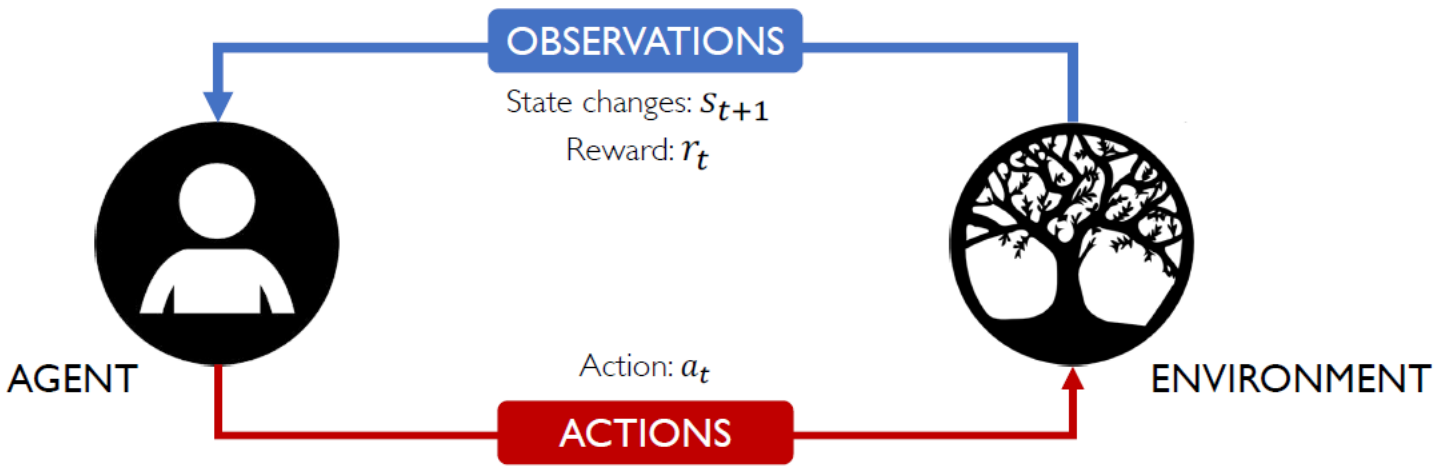
Reinforcement Learning (RL): Key Concepts



Total Reward

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

Reinforcement Learning (RL): Key Concepts

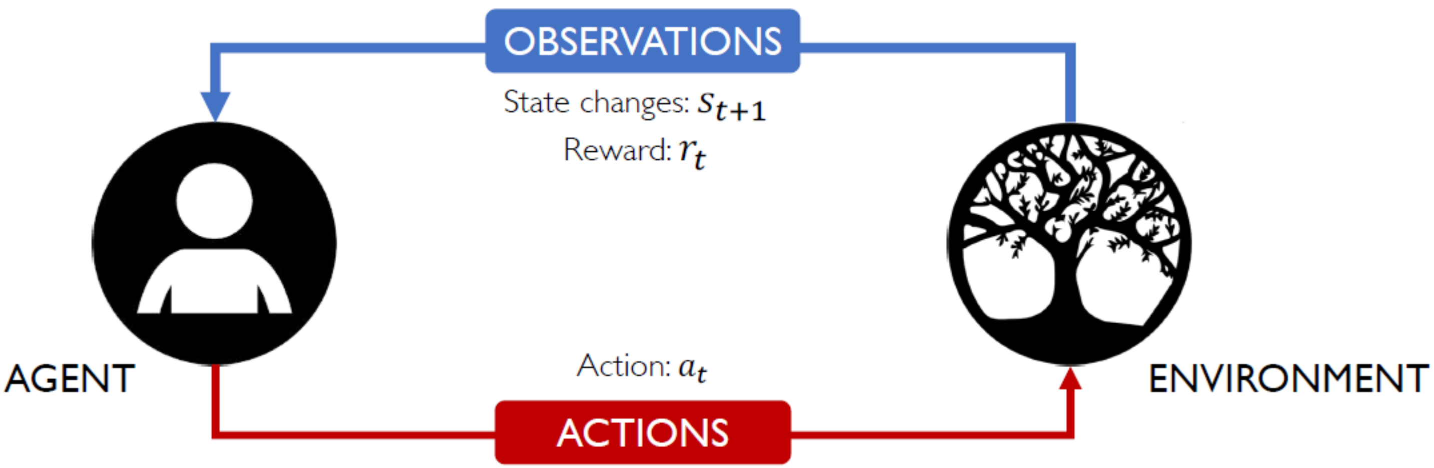


Discounted Total Reward

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

γ: discount factor

Reinforcement Learning (RL): Key Concepts



Discounted Total Reward

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

γ: discount factor

γ: 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, R_t, is the discounted sum of all rewards obtained from time t

$$Q(s, a) = \mathbb{E}[R_t]$$

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?

$$Q(s, a) = \mathbb{E}[R_t]$$

(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 Reinforcement Learning Algorithms

Value Learning

Find $Q(s, a)$

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

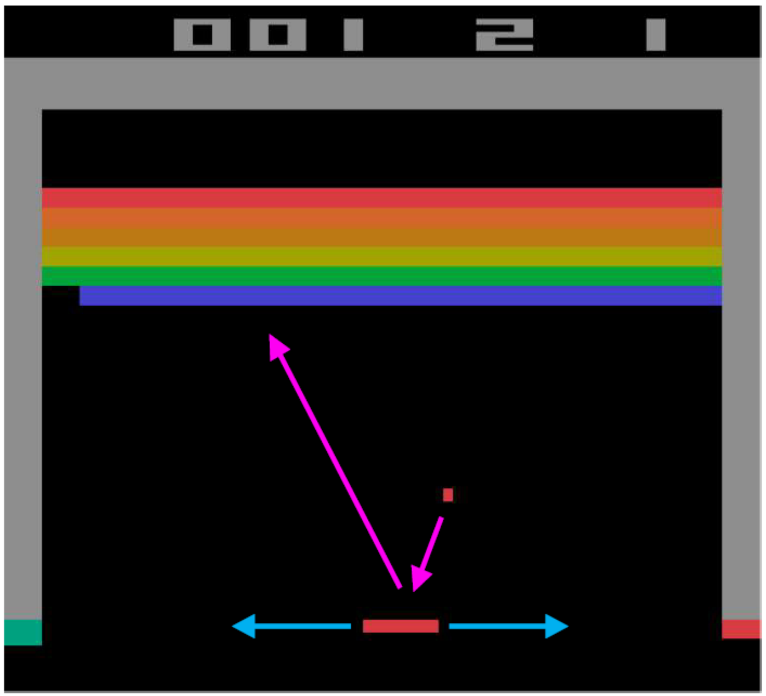
Policy Learning

Find $\pi(s)$

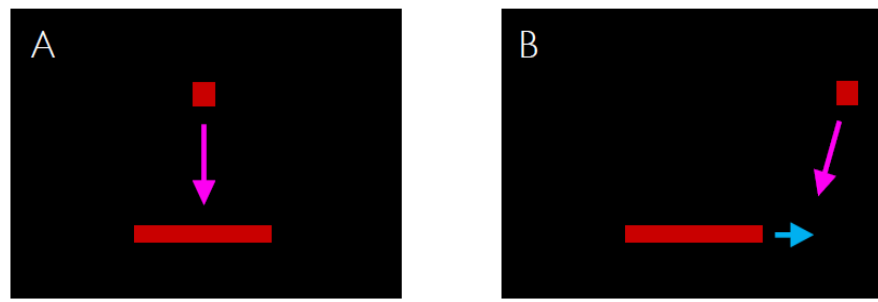
Sample $a \sim \pi(s)$

Digging deeper into the Q-function

Example: Atari Breakout



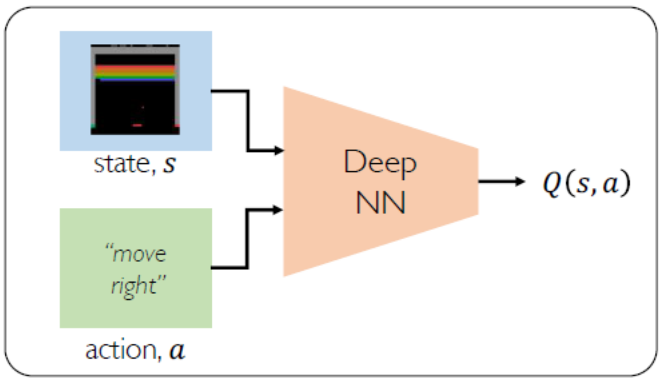
It can be very difficult for humans to accurately estimate Q-values



Which (s, a) pair has a higher Q-value?

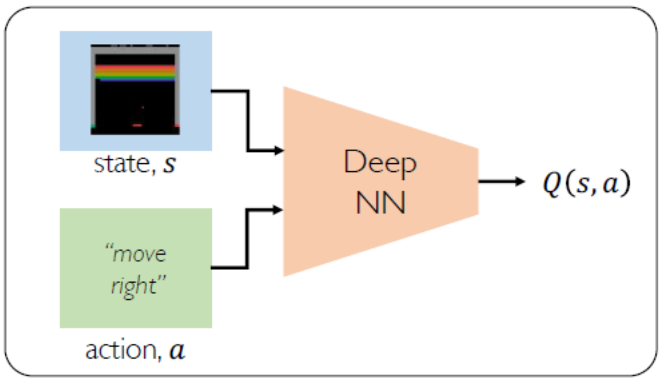
Deep Q Networks (DQN)

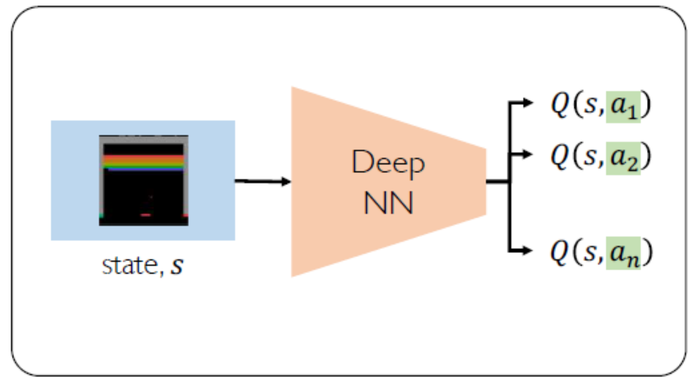
How can we use deep neural networks to model Q-functions?



Deep Q Networks (DQN)

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

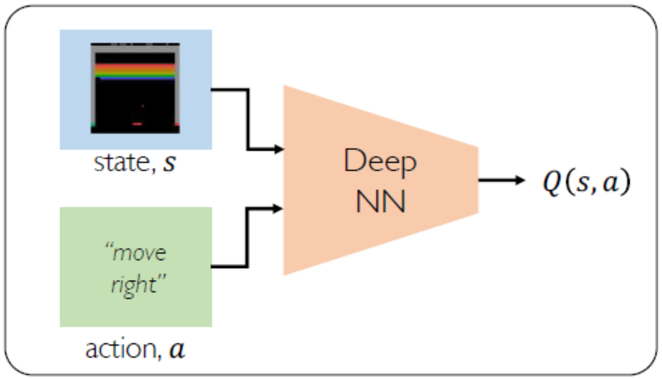


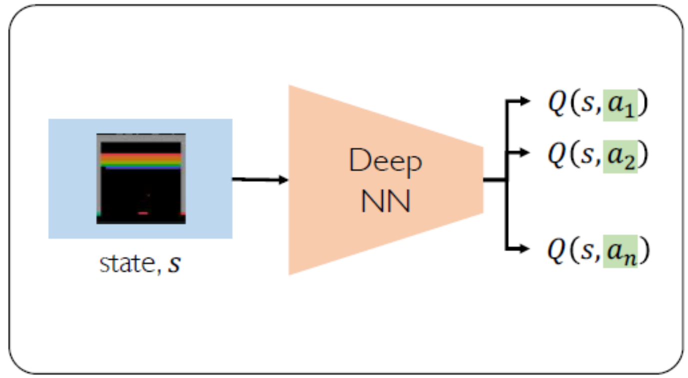


$$\left(\overbrace{r + \gamma \max_{a'} Q(s',a')}^{\text{target}}\right)$$

Deep Q Networks (DQN)

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

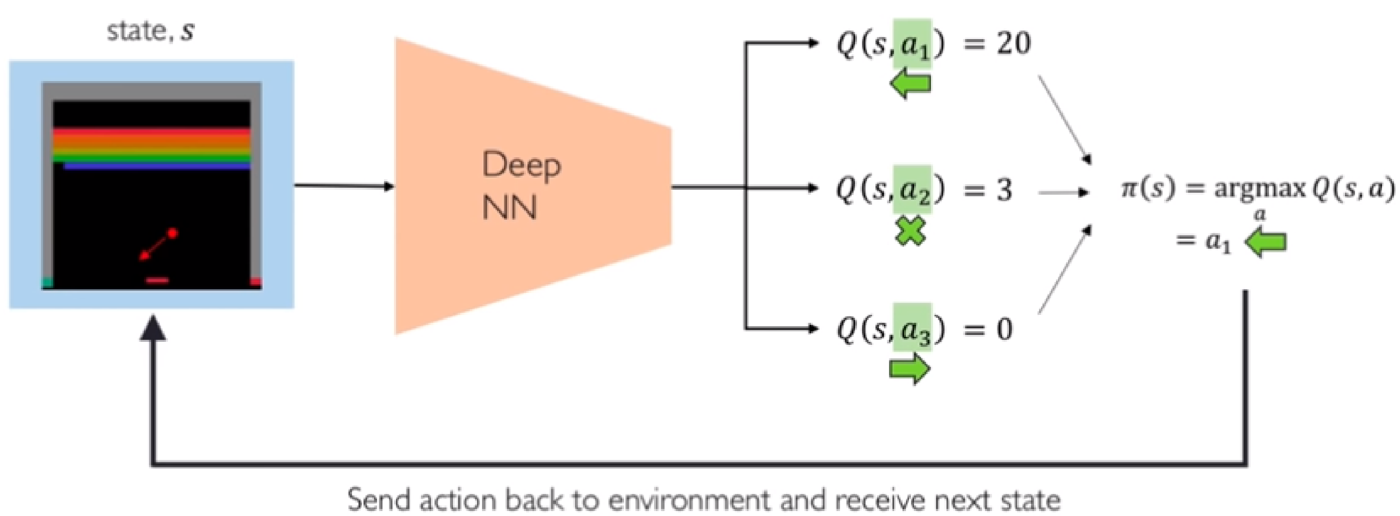




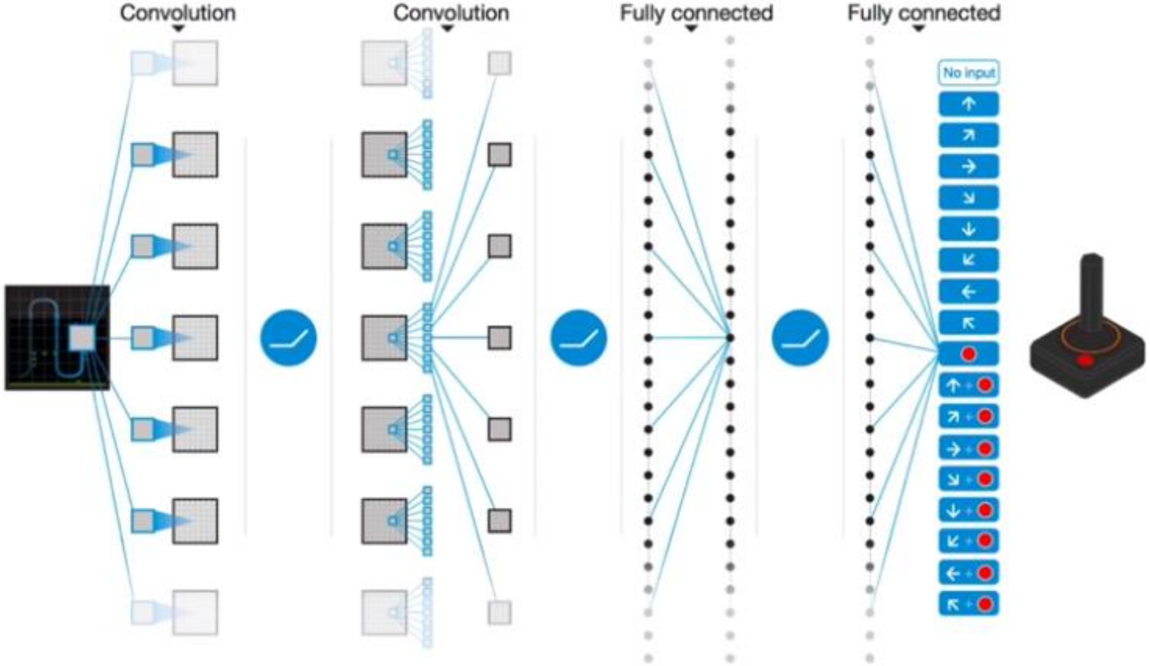
$$\mathcal{L} = \mathbb{E}\left[\left\| \overbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}^{\text{target}} - \overbrace{Q(s, a)}^{\text{predicted}} \right\|^2\right]$$ **Q-Loss**

Deep X Networks

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

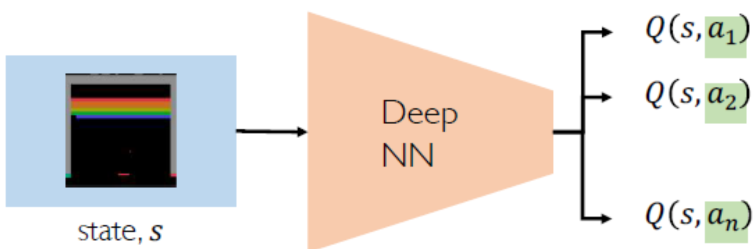
IMPORTANT:
Imagine you want to predict steering wheel angle of a car!

Flexibility:
- Cannot learn stochastic policies since policy is deterministically computed from the Q function

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

Policy Gradient (PG) : Key Idea

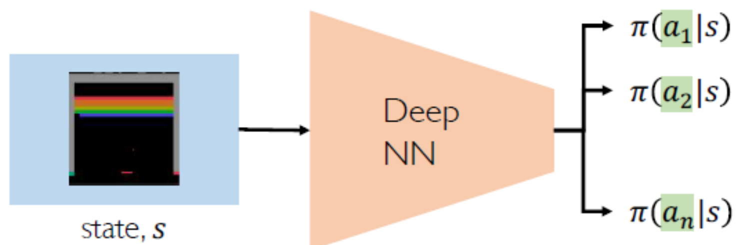
DQN (before): Approximating Q and inferring the optimal policy,



Policy Gradient (PG): Key Idea

DQN (before): Approximating Q and inferring the optimal policy,

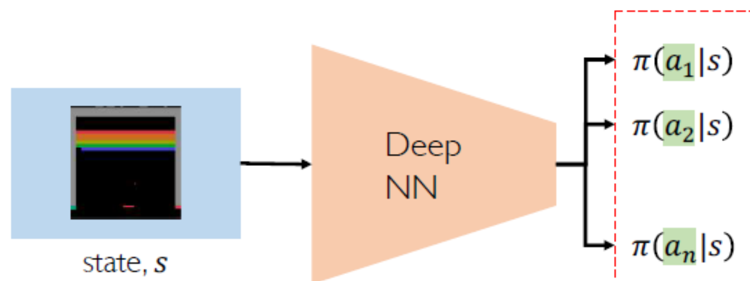
Policy Gradient: Directly optimize the policy!



Policy Gradient (PG): Key Idea

DQN (before): Approximating Q and inferring the optimal policy,

Policy Gradient: Directly optimize the policy!



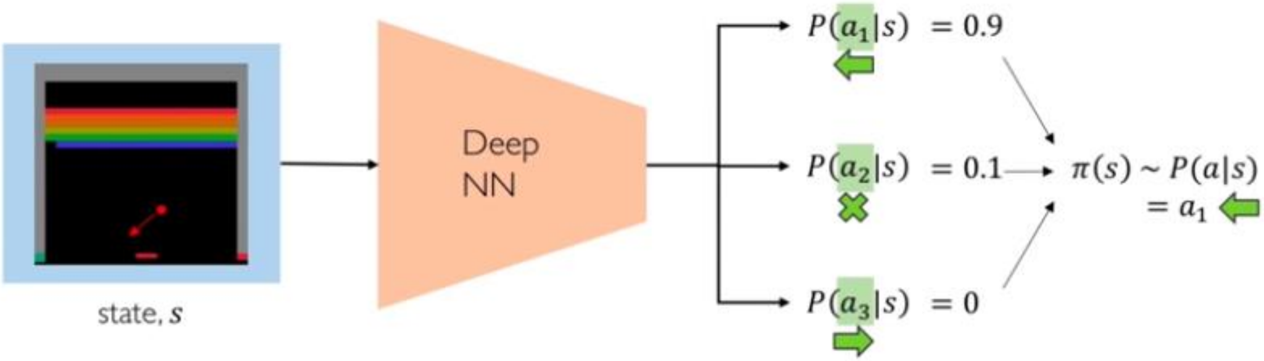
$$\sum_{a_i \in A} \pi(a_i|s) = 1$$

$$\pi(a|s) = P(action|state)$$

Policy Gradient (PG): Key Idea

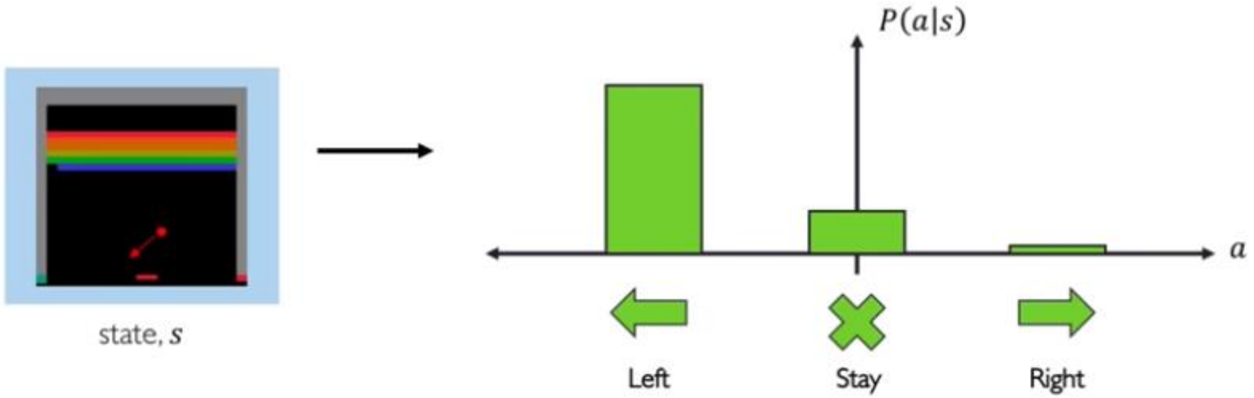
DQN: Approximate Q-function and use to infer the optimal policy, $\pi(s)$

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



Discrete vs Continuous Action Spaces

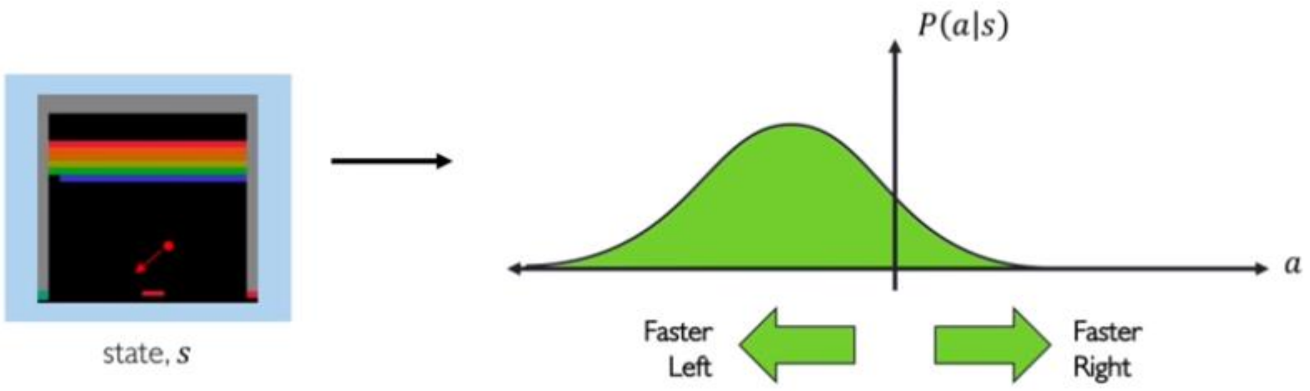
Discrete action space: which direction should I move?



Discrete vs Continuous Action Spaces

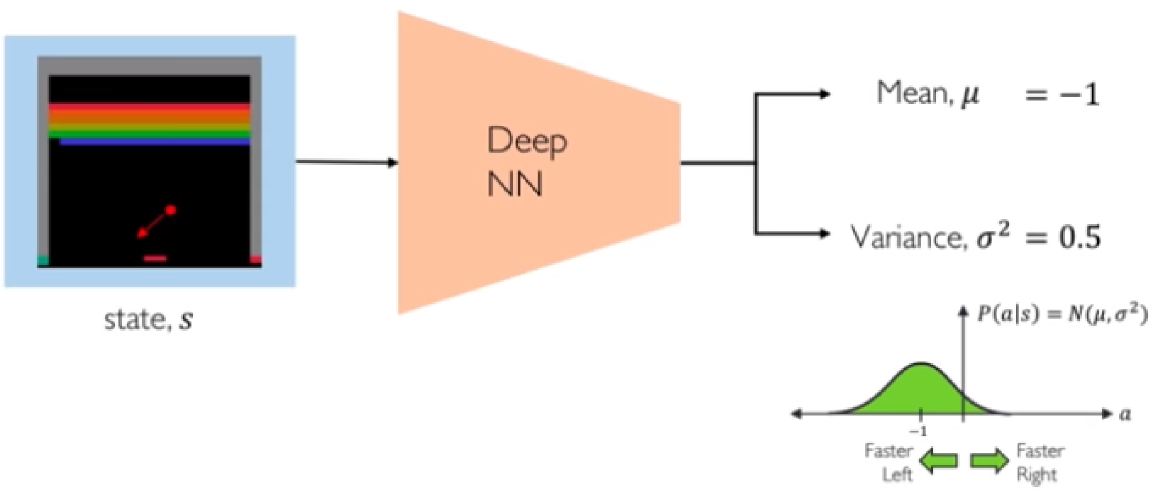
Discrete action space: which direction should I move?

Continuous action space: how fast should I move?



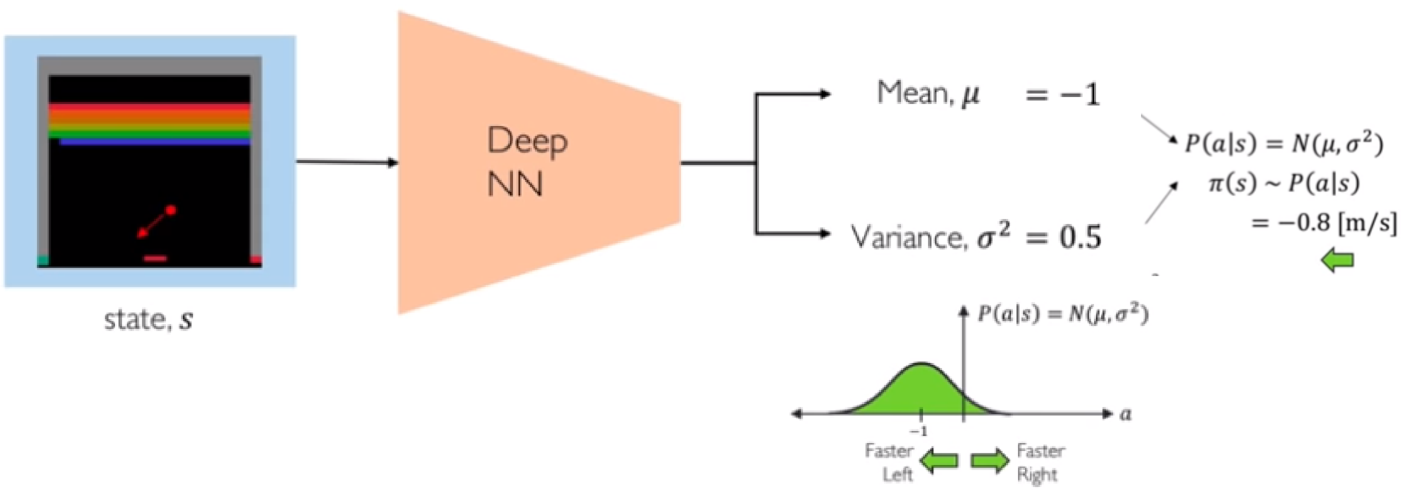
Policy Gradient (PG): Key Idea

Policy Gradient: Enables modeling of continuous action space



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

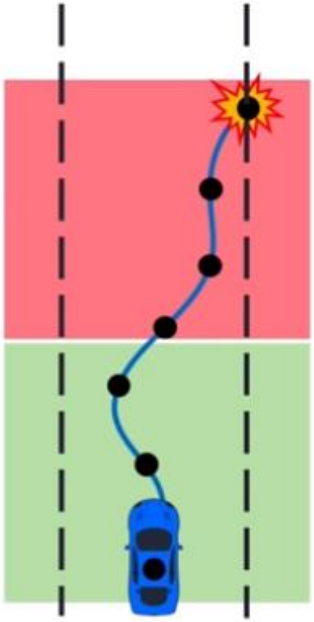
Action: steering wheel angle

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

$$\mathbf{loss} = -\underbrace{\overbrace{\log \mathrm{P}(a_t|s_t)}^{\text{log-likelihood of action}}\ \underbrace{R_t}_{\text{reward}}}_{}$$

Gradient descent update:

$$w' = w - \nabla\,\mathbf{loss}$$
$$w' = w + \underbrace{\boxed{\nabla \log \mathrm{P}(a_t|s_t)\ R_t}}_{\text{Policy gradient!}}$$

References

Andrew Ng's Reinforcement Learning course, lecture 16
https://www.youtube.com/watch?v=RtxI449ZjSc

Andrej Karpathy's blog post on policy gradient
http://karpathy.github.io/2016/05/31/rl/

Mnih et. al, Playing Atari with Deep Reinforcement Learning (DeepMind)
https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf

Intuitive explanation of deep Q-learning
https://www.nervanasys.com/demystifying-deep-reinforcement-learning/

Exam Guidelines

- **Date: Tue Jun 22**

- **Administering the exam:**
  - During lecture time (or ~12 hrs earlier if you are in a different time zone)
  - Open: Video camera + Microphone
  - Open exam pdf posted on Piazza
  - Take photos of your solutions on paper
  - Submit a pdf of the photos on Gradescope, just like you submit assignments
  - Confirm we received your submission before you leave (through private chat)

Exam Guidelines

- **What do you need?**
  - Internet + Pen/pencil + Empty sheets of paper (~10)
  - New question, new page
  - Charged cell phone to take photos of your solutions at the end for submission
  - Practice converting photos taken into a pdf for upload