Reinforcement Learning

Introduction, Markov Decision Process, and Q-Learning

N. Rich Nguyen, PhD

SYS 6016

Classes of Learning Problems

Supervised Learning:

Data: (x, y)

x is data, y is label

Goal: Learn function to map x → y

Example:

This thing is an apple.

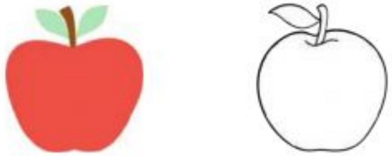
Unsupervised Learning:

Data: x

x is data, no labels!

Goal: Learn underlying structure

Example:



This thing is like the other thing.

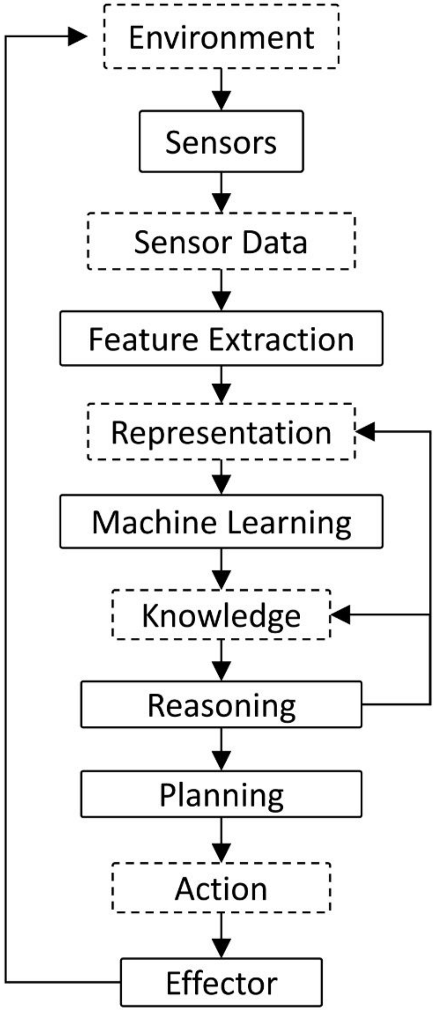
Reinforcement Learning:

Data: state-action pairs

Goal: Maximize future rewards over many steps

Example:

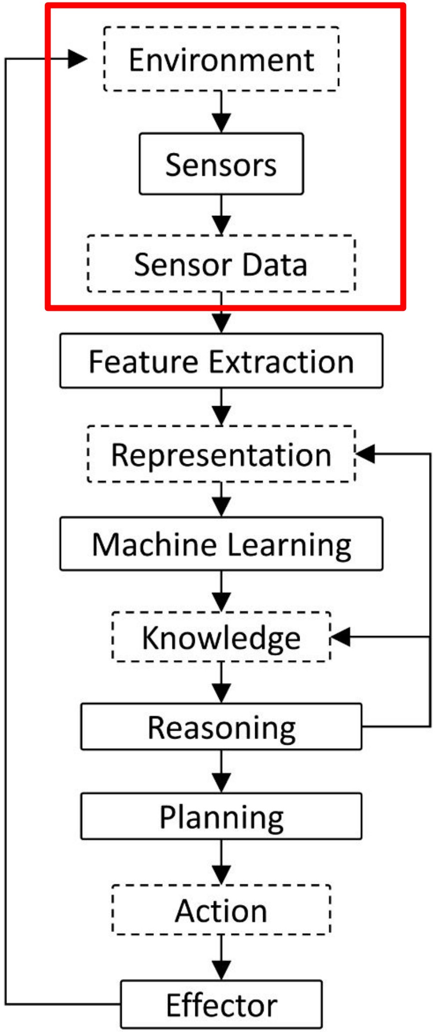
Eat this thing because it will keep you alive.

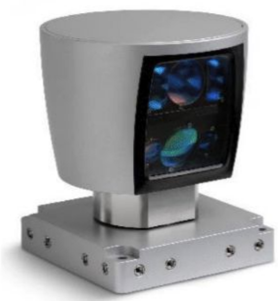


The Machine Learning Stack!

What can be learned?

Sensors





Lidar

Camera
(Visible, Infrared)

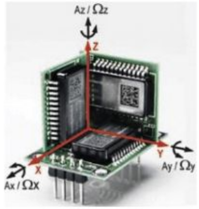
Radar

GPS

Stereo Camera

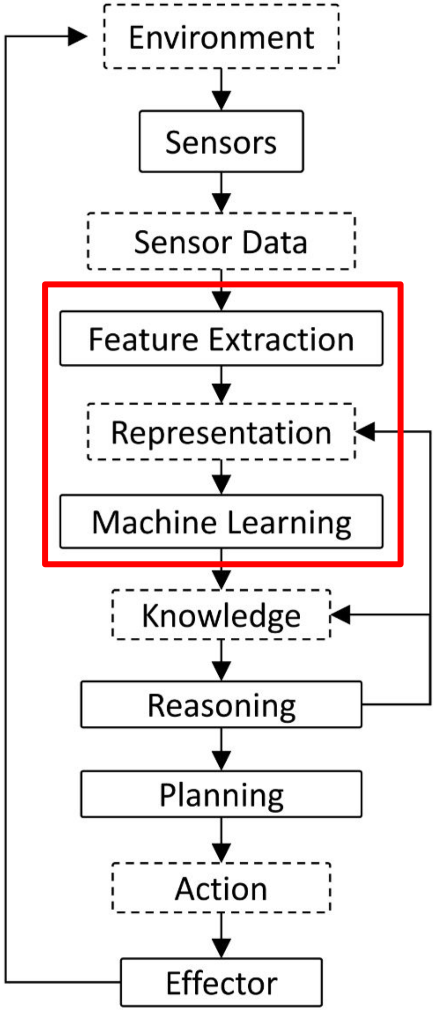
Microphone

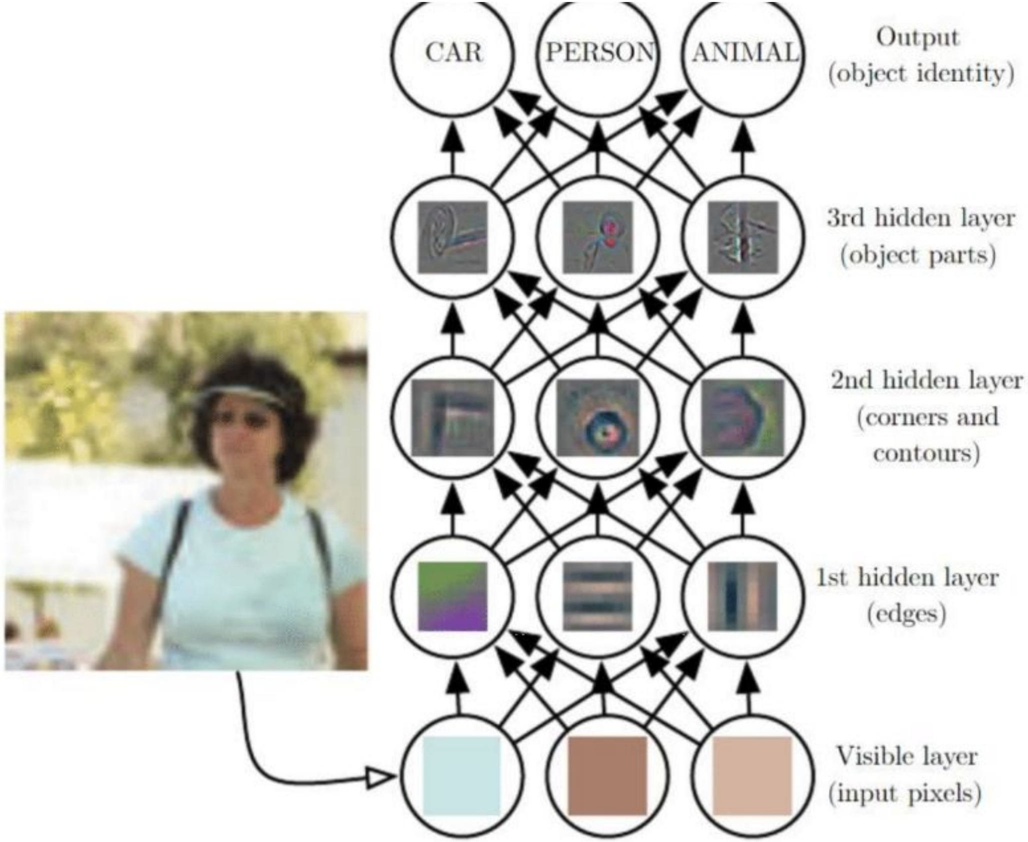
Networking
(Wired, Wireless)

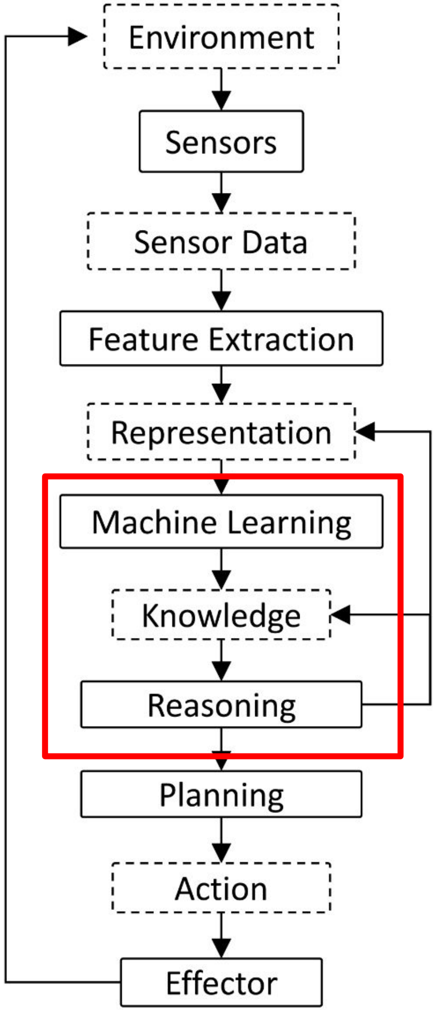


IMU

Representations



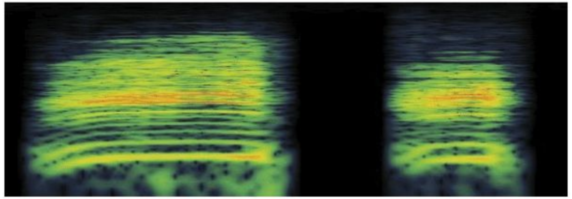




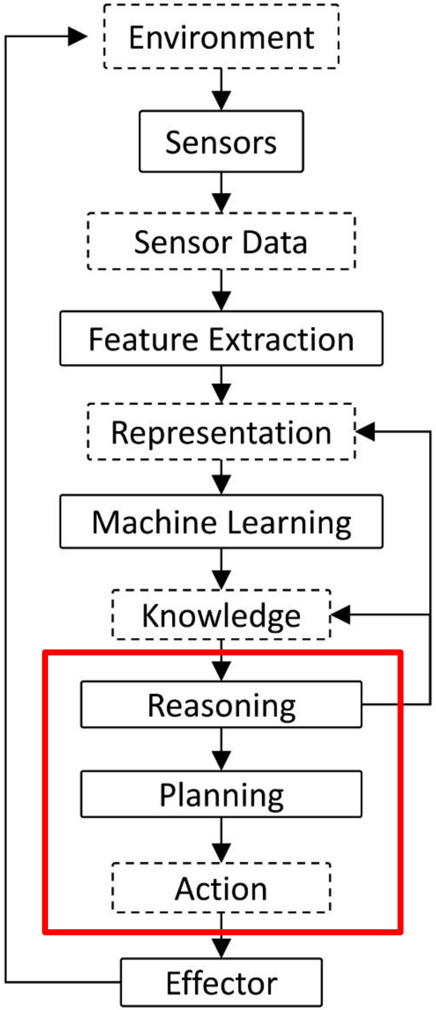
Knowledge / Reasoning

Image Recognition:
If it looks like a duck

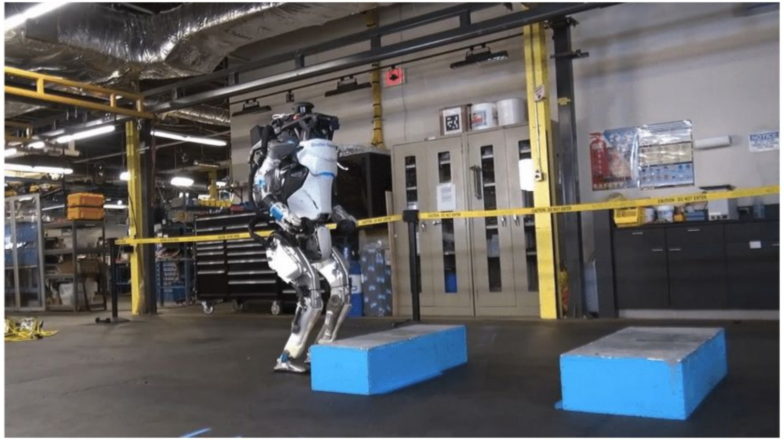
Audio Recognition:
Quacks like a duck

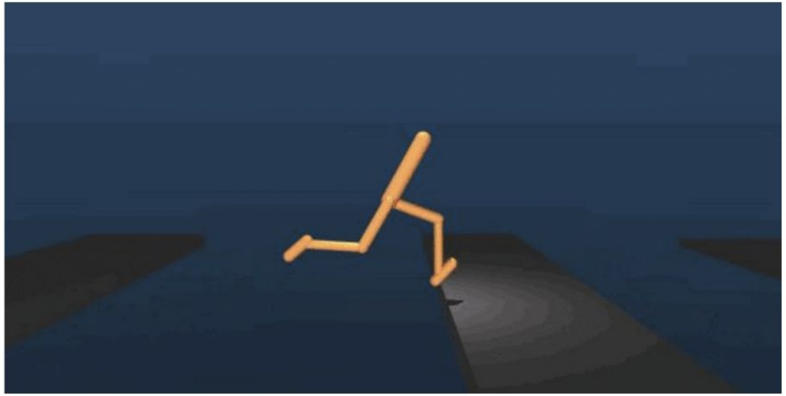


Activity Recognition:
Swims like a duck

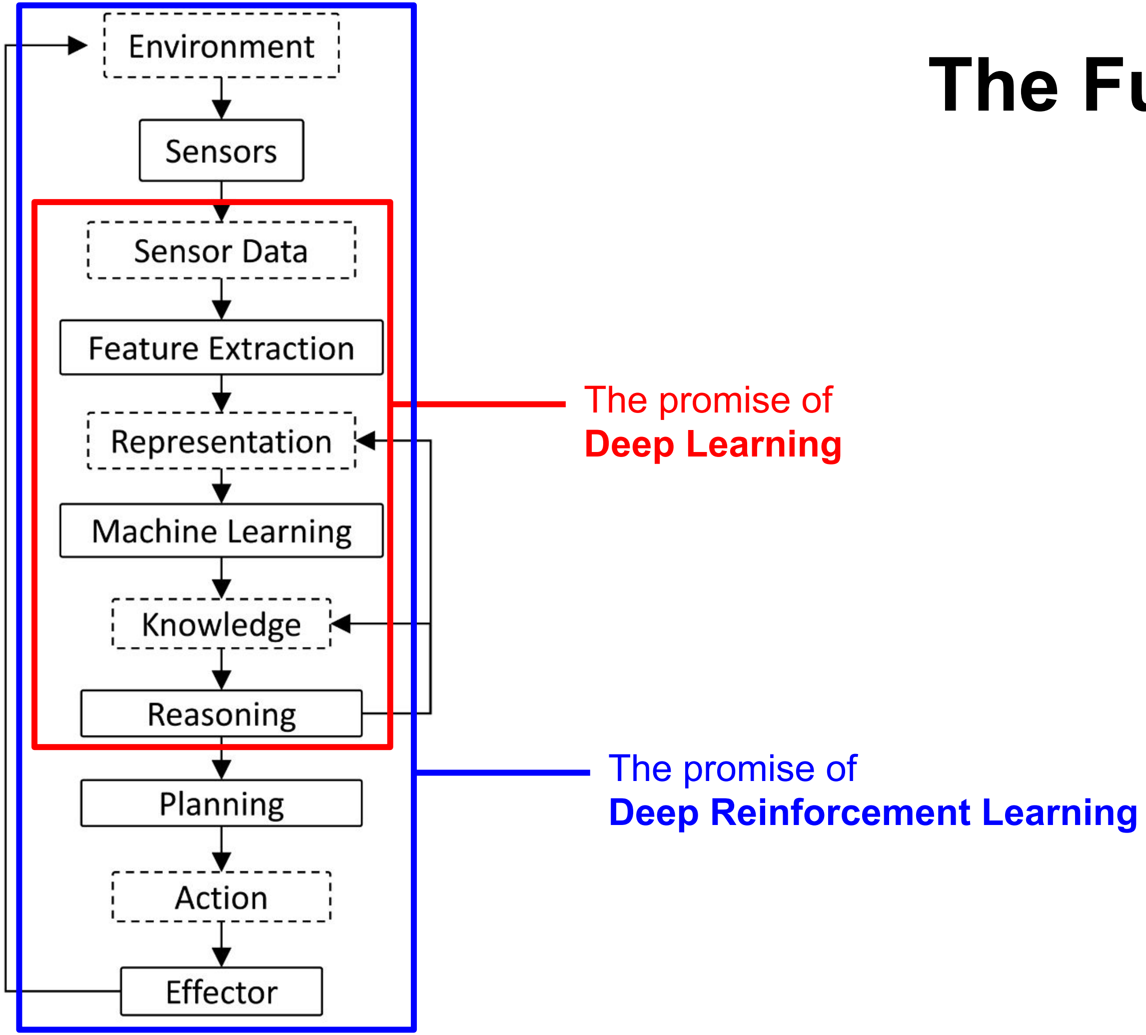

Environment
Sensors
Sensor Data
Feature Extraction
Representation
Machine Learning
Knowledge
Reasoning
Planning
Action
Effector

Actions





The Full Stack



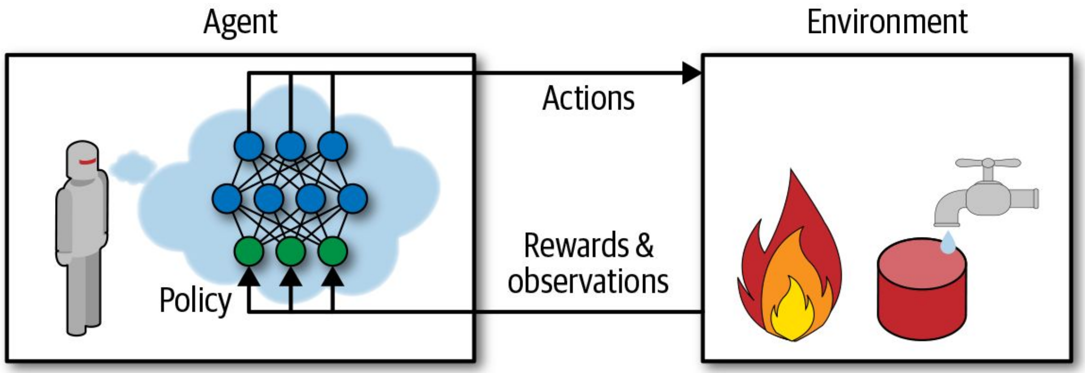
Reinforcement Learning

Reinforcement Learning (RL)

Problems involving an **agent** interacting with an **environment**, which provides numeric reward signals:

At each step, the agent:

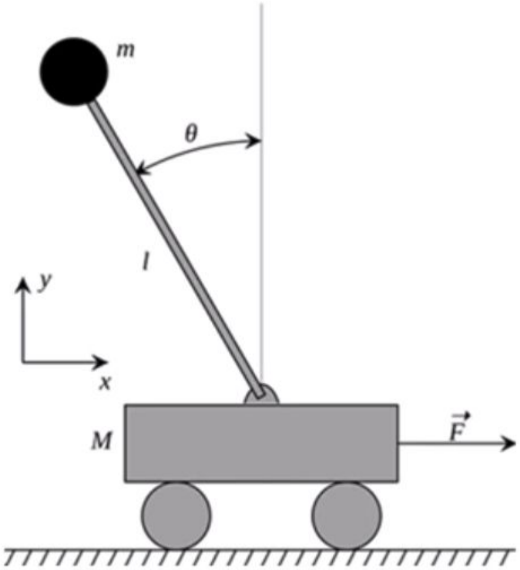
- Executes an **action**
- Observes a new **state**
- Receives some **reward**



Goal: Learn to take actions from a policy in order to **maximize** some reward

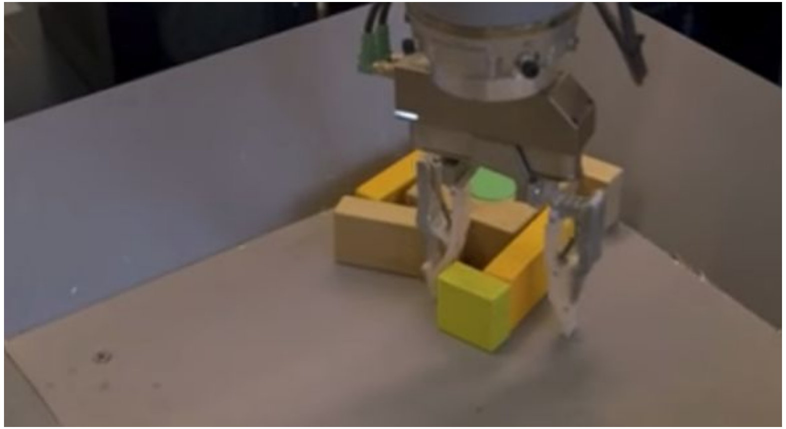
Example: Cart-Pole Balancing Problem

- **Goal**: Balance the pole of top of a moving cart
- **State**: Pole angle, angular speed, cart position, horizontal velocity
- **Actions**: Horizontal force to the cart
- **Reward**: +1 at each time step if the pole is upright



Example: Grasping Objects

- **Goal**: Pick an object with different shape
- **State**: Raw pixels from camera
- **Actions**: Move arm, grasp, and lift
- **Reward**: positive when a pickup is successful



Example: Doom

- **Goal**: eliminate all opponents
- **State**: raw game pixel of the game
- **Actions**: Shoot, Dodge, Run, Walk, Left, Right, No action, ect.
- **Reward**: positive when eliminating an opponent, negative when the agent is eliminated

Example: MIT DeepTraffic

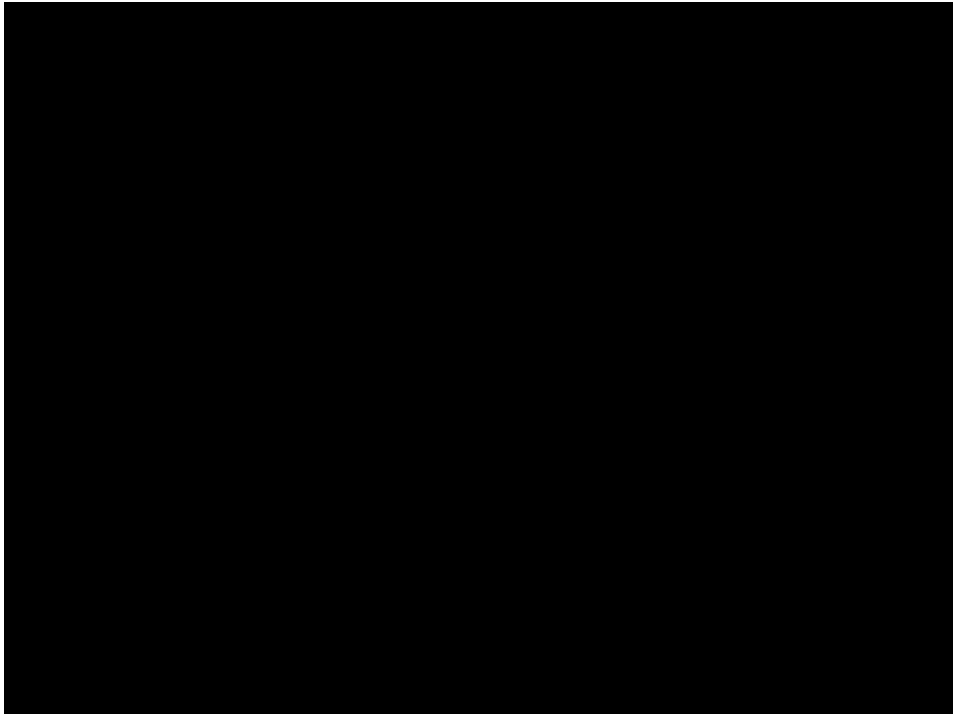
- **Goal**: train an RL agent that can successfully navigate through traffic
- **State**: as a grid, where each cell will be the speed of the vehicle inside it
- **Actions**: Accelerate, Break, Left, Right, No action
- **Reward**: positive when high speed is maintained.

RL in Humans

Humans appear to learn to walk through "very few examples" of trial and error.

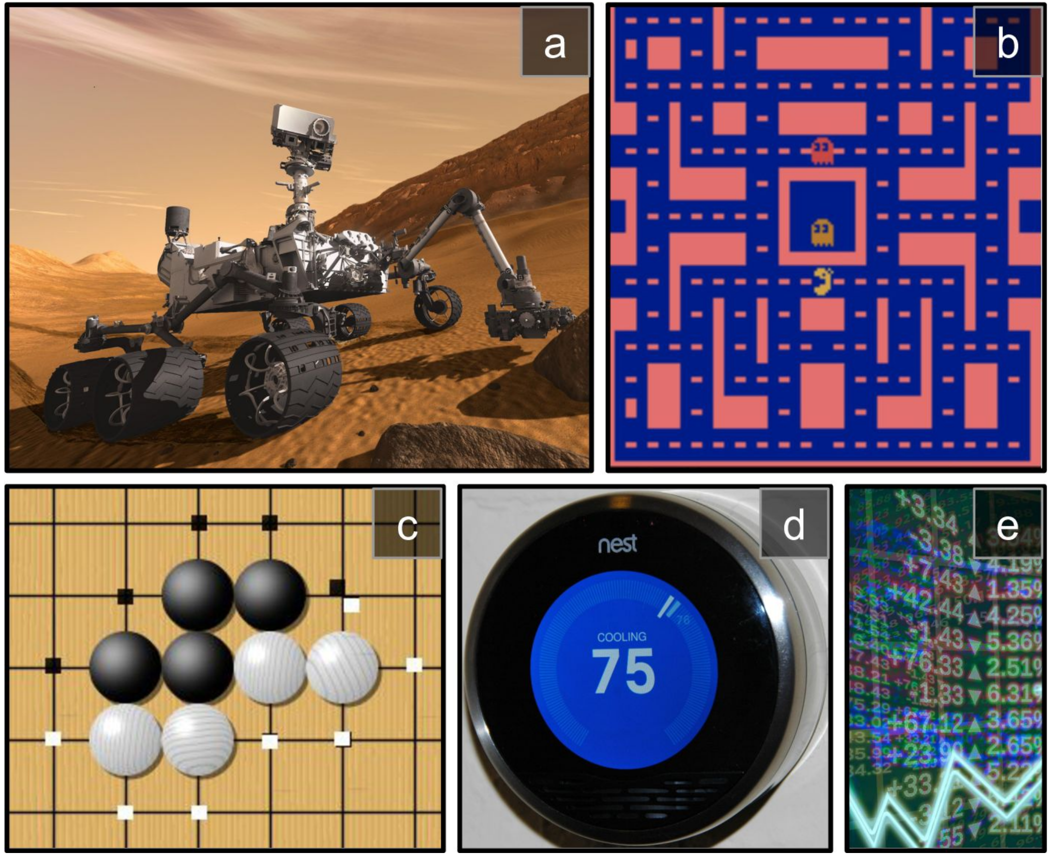
"**How** we learn how to walk" is an open question…some possible answers:

- **Hardware**: 230M years of bipedal movement data
- **Imitation Learning**: Observation of other humans walking
- **Algorithms**: probably better than backprop and SGD



RL Real-world Applications

a. Robotic Explorer
b. Ms. Pac-man
c. Go player
d. Thermostat
e. Automated Trader



Markov Decision Process

Markov Decision Process (MDP)

Markov Decision Process is the mathematical formulation for RL problem

Markov property: current state completely characterizes the state of the world.

$$(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$$

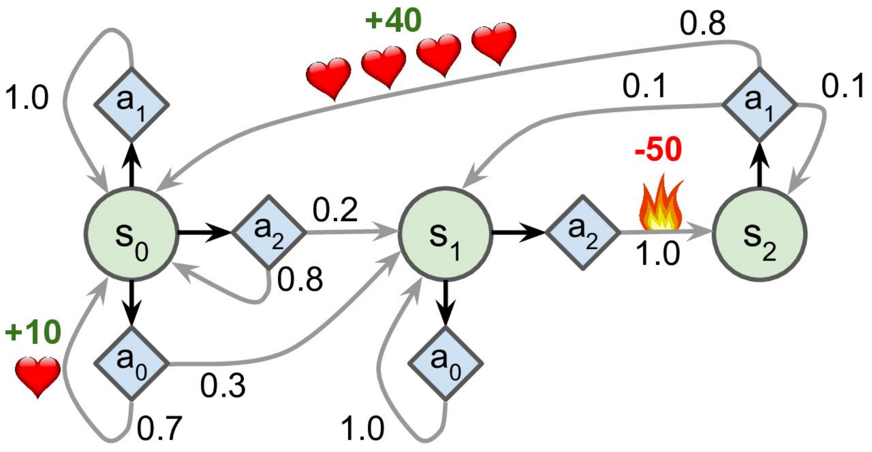
Set of possible states

Set of possible actions

Reward distribution given (state, action)

Transition probability distribution of next state

Discount factor



Markov Decision Process

At time step `t=0`, environment samples initial state $s_0 \sim \mathbb{P}(s_0)$, then repeat:

- Agent selects **action** a_t
- A **reward** is sampled from the environment $r_t \sim \mathcal{R}(.|s_t, a_t)$
- Next **state** is sampled from the environment $s_{t+1} \sim \mathbb{P}(.|s_t, a_t)$
- Agent receives the **reward** r_t and **next state** s_{t+1}

Objective: find a strategy that maximizes **future reward**

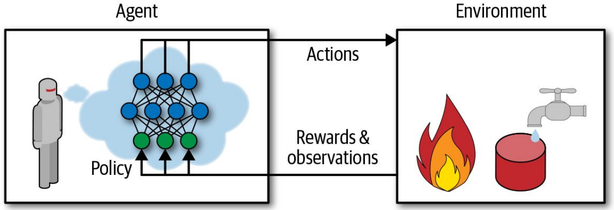
$$s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{n-1}, a_{n-1}, r_n, s_n$$

initial state

action

reward

terminal state



Maximize Future Reward

Future Reward: $R_t = r_t + r_{t+1} + r_{t+2} + \ldots + r_n$

Discounted Future Reward: $R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{n-t} r_n$

$0 < \gamma < 1$

A good strategy for **an agent** would be to always choose **an action** that **maximizes** the **discounted future reward**

Why?

- Uncertainty due the environment, partial observability
- Real life example: Either live it up today, or save $ for tomorrow?

Taking actions given a Q-function

$$\mathbf{Q}(s_t, a_t) = \mathbb{E}\left[\mathbf{R}_t \,|\, s_t, a_t\right]$$

state

action

reward

A policy π is a function from **S** to **A** that specifies what action to take in each state

The agent needs a policy π(s) to infer the **best action** to take at a **state** **s**.

The policy should choose an action that **maximizes** **the expected future reward (Q-value)**:

$$\pi(s) = \arg\max_a \mathbf{Q}(s, a)$$

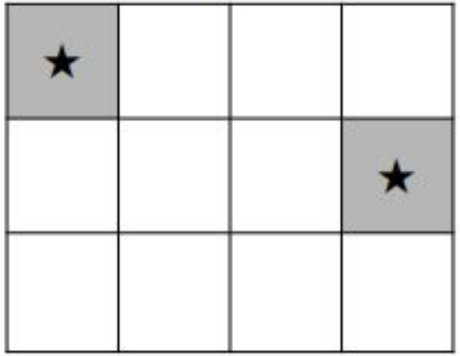
Moving in a grid world

Goal: reach one of the terminal states (stars) using the least number of actions.

actions = {
1. right
2. left
3. up
4. down

}

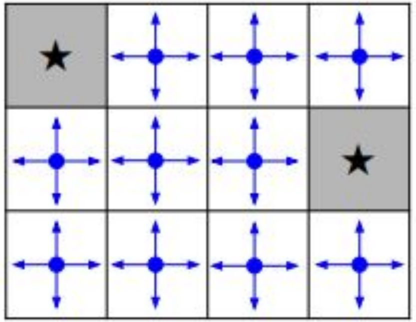
states



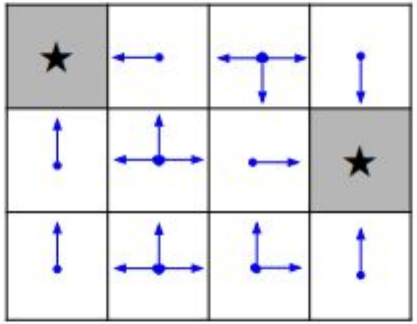
Set a negative "reward" for each transition (e.g. r = -1)

Policy to move in a grid world

Goal: reach one of the terminal states (stars) using the least number of actions.



Random Policy



Optimal Policy

3 Types of Reinforcement Learning

RL agents may be directly or indirectly trying to do one of the following:

1. **Model Learning**: model the environment then use and update it often (difficult because of the stochastic nature of the environment)
2. **Value Learning (Q-Learning)**: learn the state or state-action value, act by choosing best action, and explore if necessary
3. **Policy Learning (Policy Gradients)**: learn the stochastic policy function that maps state to action, act by sampling that policy

Value Learning / Q-learning

Q-Learning for Solving the Optimal Policy

Use a **state-action value function Q_π(s,a)** as **the expected return** when starting in s, performing a, and following π

Q-Learning: Use any policy to estimate **Q** that maximizes future reward:

- **Q** directly approximates **Q*** (Bellman's equation)
- Independent of the policy, only requirement is keep updating (s_t, a_t) pair

$$\underbrace{\mathbf{Q}_{t+1}(s_t, a_t)}_{\text{new Q-value}} = \underbrace{\mathbf{Q}_t(s_t, a_t)}_{\text{old Q-value}} + \underbrace{\alpha}_{\text{learning rate}}\Big(\underbrace{\mathbf{R}_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_a \mathbf{Q}_t(s_{t+1}, a)}_{\text{future Q-value upon selecting the best action}} - \mathbf{Q}_t(s_t, a_t)\Big)$$

Q-Learning: Value Iteration

$$\mathbf{Q}_{t+1}(s_t, a_t) = \mathbf{Q}_t(s_t, a_t) + \alpha\big(\mathbf{R}_{t+1} + \gamma \max_a \mathbf{Q}_t(s_{t+1}, a) - \mathbf{Q}_t(s_t, a_t)\big)$$

$$= (1-\alpha)\mathbf{Q}_t(s_t, a_t) + \alpha\big(\mathbf{R}_{t+1} + \gamma \max_a \mathbf{Q}_t(s_{t+1}, a)\big)$$

new Q-value

old Q-value

learning rate

reward

discount factor

future Q-value upon selecting the best action

	A1	A2	A3	A4
S1	+1	+2	-1	0
S2	+2	0	+1	-2
S3	-1	+1	0	-2
S4	-2	0	+1	+1

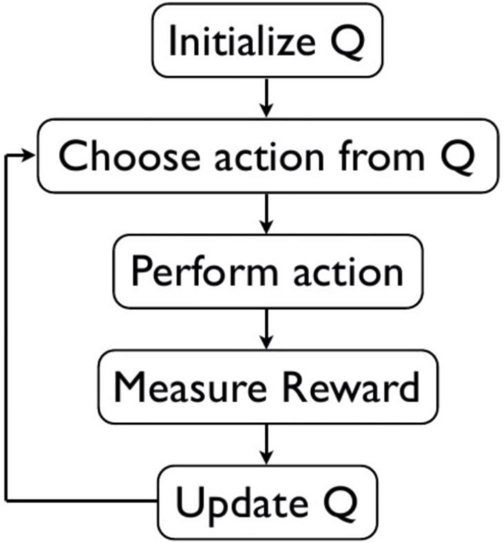
```
initialize Q[num_states,num_actions] arbitrarily
observe initial state s
repeat
    select and carry out an action a
    observe reward r and new state s'
    Q[s,a] = Q[s,a] + α(r + γ max_a' Q[s',a'] - Q[s,a])
    s = s'
until terminated
```

Example: Moving in a grid world

Objective: reach one of the terminal states (stars) using the least number of actions.

$$\text{Find } \mathbf{Q}(s, a)$$

$$a = \arg\max_{a} \mathbf{Q}(s, a)$$

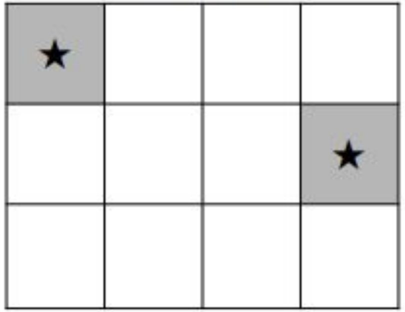


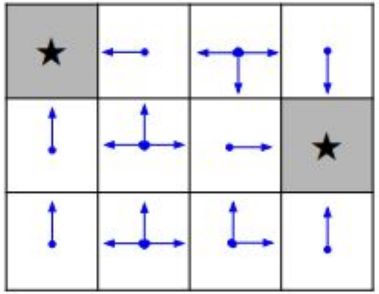
actions = {

1. right
2. left
3. up
4. down

}

states



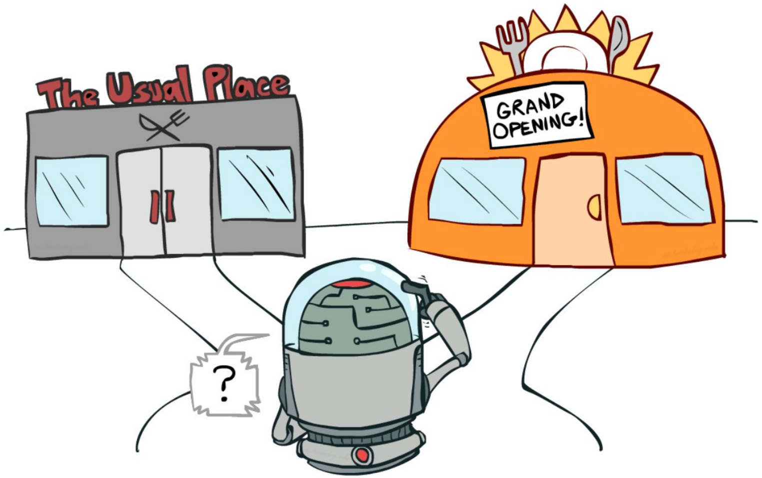


Optimal Policy

Exploration vs. Exploitation

Deterministic/greedy policy won't explore all actions:

- Don't know anything about the environment at the beginning
- Need to try all actions to find the optimal one

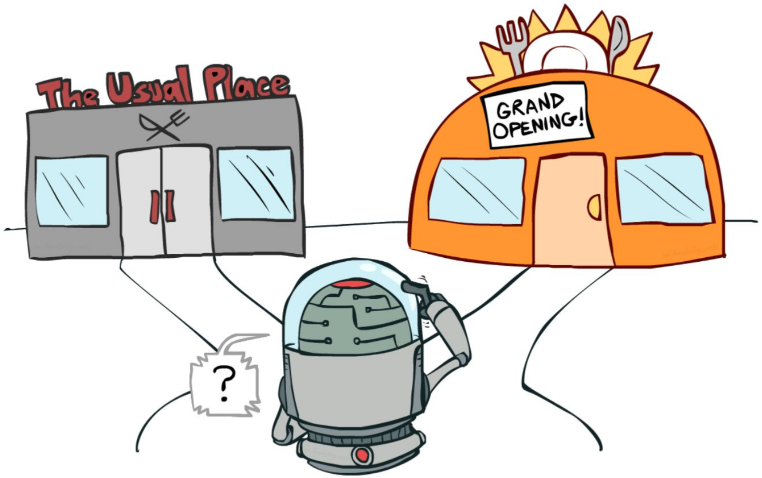


ε-greedy policy

- With probability $(1-\varepsilon)$ of performing the greedy action, otherwise random action
- Slowly move toward greedy policy: $\varepsilon \rightarrow 0$

Exploration vs. Exploitation Examples

- Restaurant Selection
  - **Exploitation**: Go to your favourite restaurant
  - **Exploration**: Try a new restaurant online
- Banner Ads
  - **Exploitation**: Show the most successful ads
  - **Exploration**: Show a different ads
- Oil Drilling
  - **Exploitation**: Drill at the best known location
  - **Exploration**: Drill at a new location
- Game Playing
  - **Exploitation**: Play the move you believe is best
  - **Exploration**: Play an experimental move



Summary

- Reinforcement Learning (RL) is one of the most exciting fields of ML
- In Markov Decision Process, a good strategy for an agent would be to always choose an action that maximizes the *discounted future reward*
- Q-Learning is to learn the state-action value, act by choosing best action, and explore if necessary

Next: Deep Q-Networks, Policy Gradients, and Actor Critic Algorithms

Acknowledgement

Slides contain some materials and figures reproduced from Lex Fridman (MIT) and Serena Yeung (Stanford) for educational purposes only.

Massachusetts Institute of Technology

Stanford University