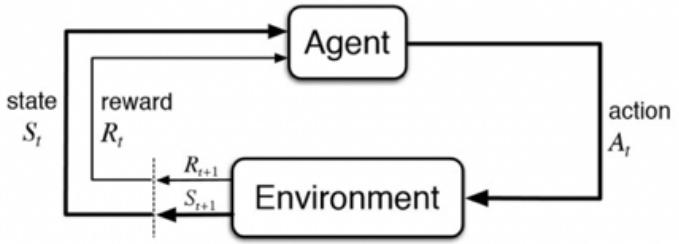


2020 Fall IERG5350 Reinforcement Learning

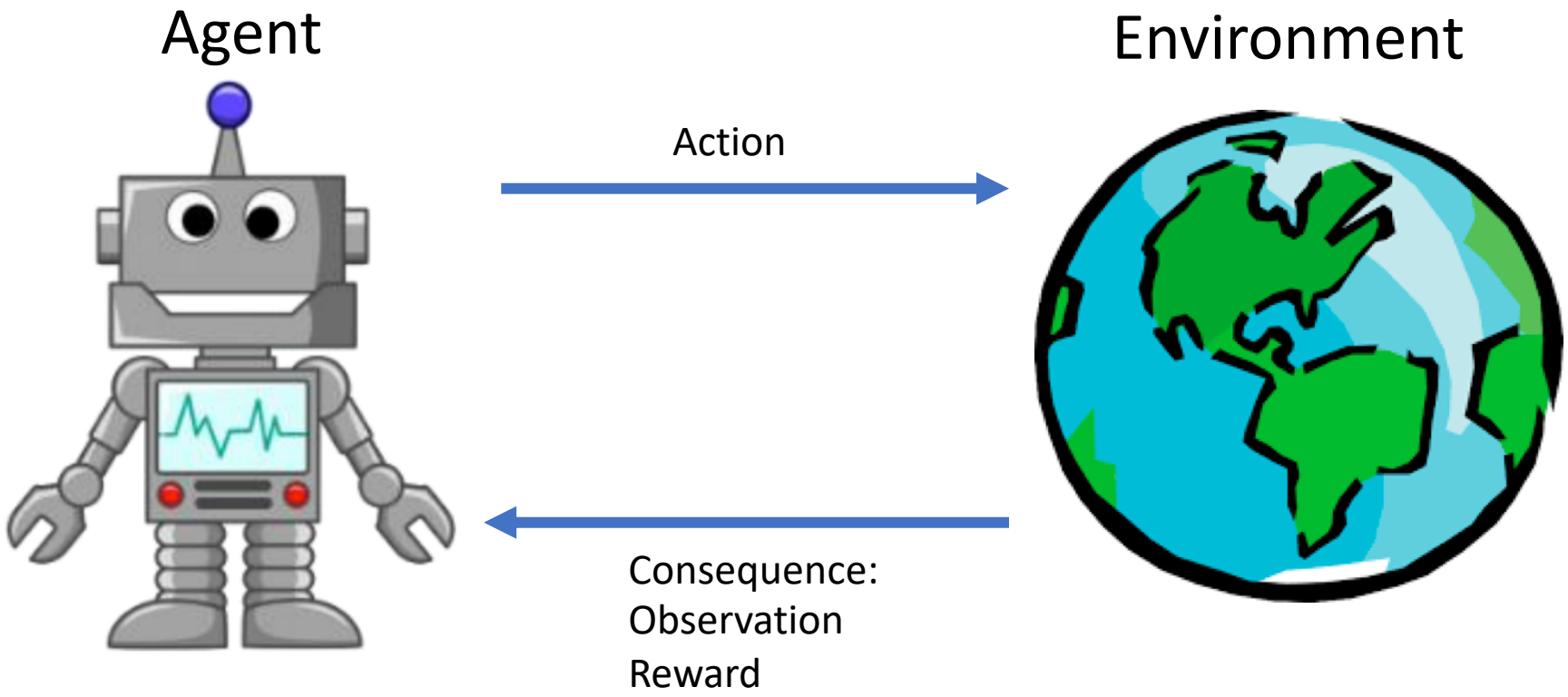
Lecture 2: RL basics and Coding with RL

Bolei Zhou

The Chinese University of Hong Kong

Agent and Environment

- The agent learns to interact with the environment



Rewards

- A reward is a scalar feedback signal
- Indicate how well agent is doing at step t
- Reinforcement Learning is based on the maximization of rewards:

All goals of the agent can be described by the maximization of expected cumulative reward.

Examples of Rewards

- Chess players play to win:

+/- reward for wining or losing a game

- Gazelle calf struggles to stand:

+/- reward for running with its mom or being eaten

- Manage stock investment

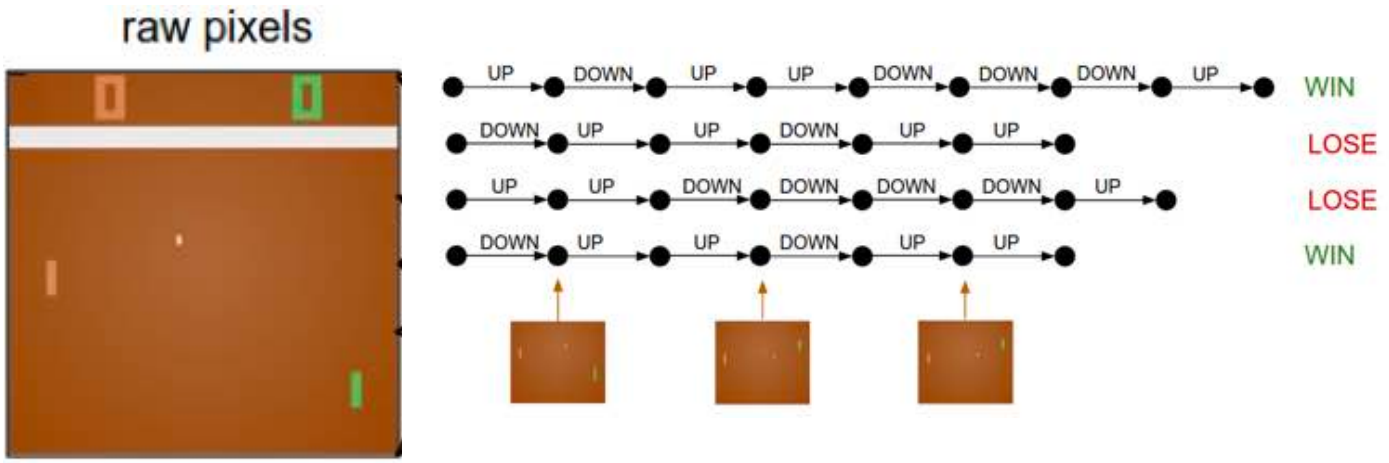
+/- reward for each profit or loss in $

- Play Atari games

+/- reward for increasing or decreasing scores

Sequential Decision Making

- Objective of the agent: select a series of actions to maximize total future rewards
- Actions may have long term consequences
- Reward may be delayed
- Trade-off between immediate reward and long-term reward



Sequential Decision Making

- The history is the sequence of observations, actions, rewards.

$$H_t = O_1, R_1, A_1, ..., A_{t-1}, O_t, R_t$$

- What happens next depends on the history
- State is the function used to determine what happens next

$$S_t = f(H_t)$$

Sequential Decision Making

- Environment state and agent state

$$S_t^e = f^e(H_t) \qquad\qquad S_t^a = f^a(H_t)$$

- **Full observability**: agent directly observes the environment state, formally as Markov decision process (MDP)

$$O_t = S_t^e = S_t^a$$

Sequential Decision Making

- Environment state and agent state

$$S_t^e = f^e(H_t) \qquad\qquad S_t^a = f^a(H_t)$$

- **Full observability**: agent directly observes the environment state, formally as Markov decision process (MDP)

$$O_t = S_t^e = S_t^a$$

- **Partial observability**: agent indirectly observes the environment, formally as partially observable Markov decision process (POMDP)
  - Black jack (only see public cards), Atari game with pixel observation,

Agent must construct its own state representation, as the beliefs of the environment state:

$$S_t^a = (P(S_t^e = s_1), ..., P(s_t^e = s_n))$$

$$S_t^a = \sigma(S_{t-1}^a W_s + O_t W_o)$$

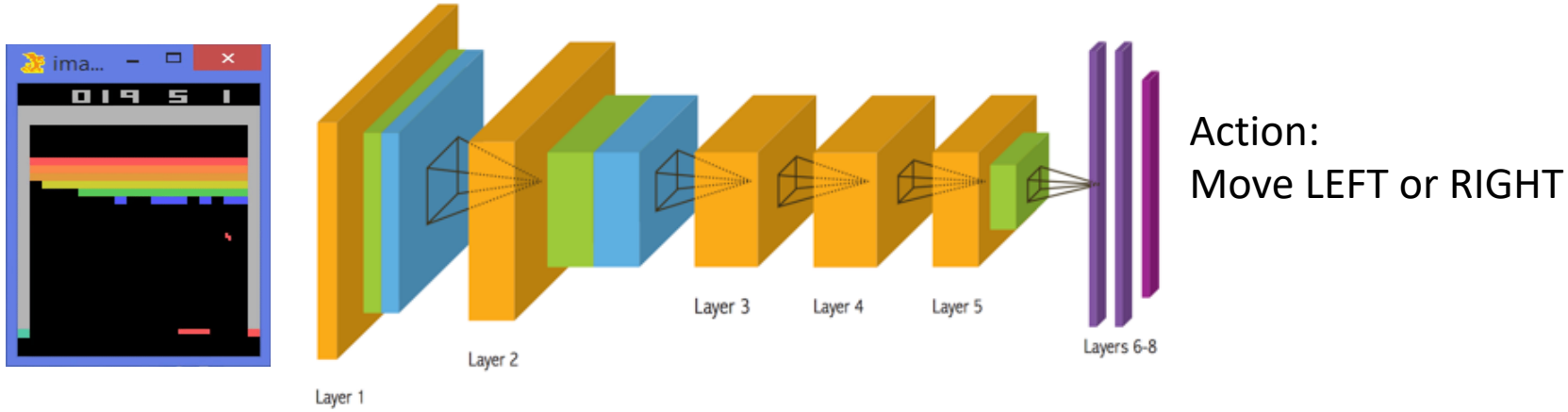
Major Components of an RL Agent

An RL agent may include one or more of these components:

- Policy: agent's behavior function
- Value function: how good is each state or action
- Model: agent's state representation of the environment

Policy

- A policy is the agent's behavior model
- It is a map function from state/observation to action.
- Stochastic policy: Probabilistic sample $\pi(a|s) = P[A_t = a|S_t = s]$
- Deterministic policy: $a^* = \arg\max_a \pi(a|s)$



Value function

- Value function: expected discounted sum of future rewards under a particular policy π
- Discount factor weights immediate vs future rewards
- Used to quantify goodness/badness of states and actions

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S},$$

- Q-function (could be used to select among actions)

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

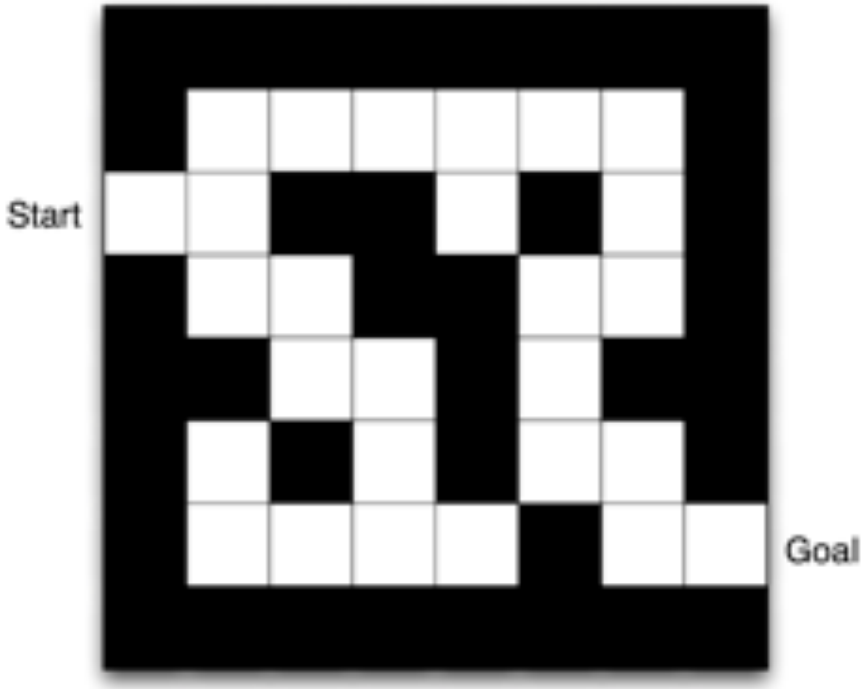
Model

A model predicts what the environment will do next

Predict the next state: $\mathcal{P}^{a}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$

Predict the next reward: $\mathcal{R}^{a}_{s} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$

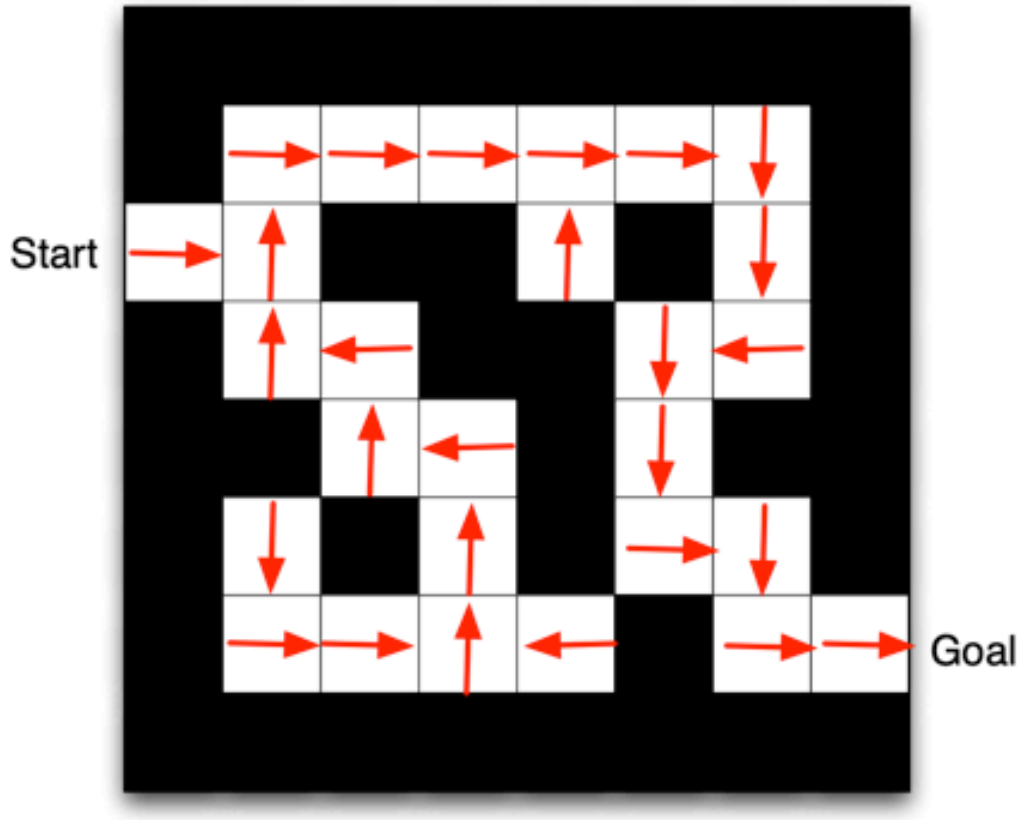
Maze Example



- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

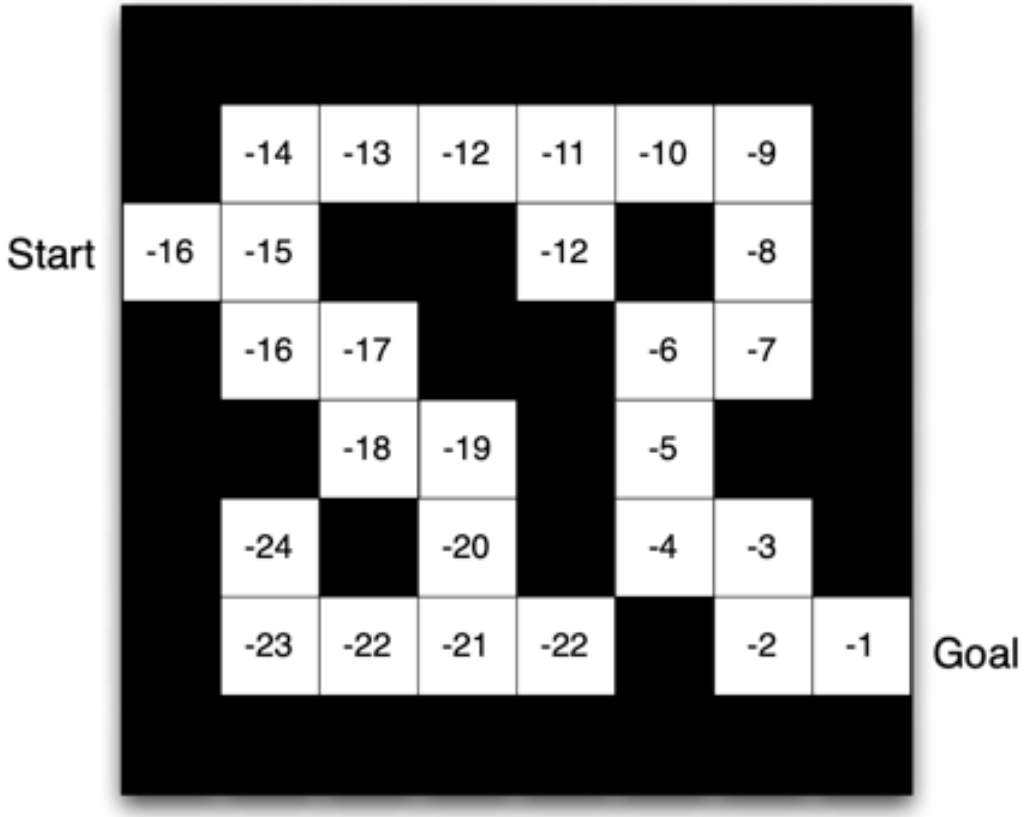
From David Silver Slide

Maze Example: Policy-based



- Arrows represent policy $\pi(s)$ for each state s

Maze Example: Value function-based



- Numbers represent value $v_\pi(s)$ of each state s

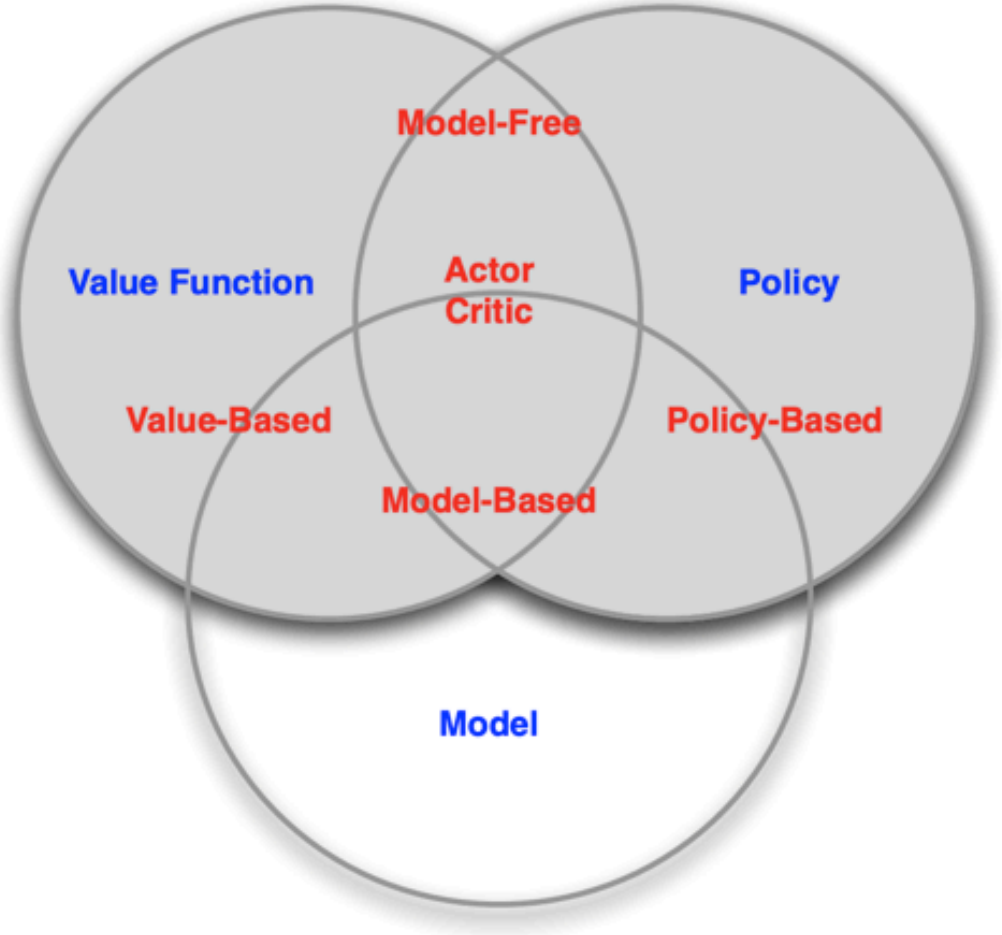
Types of RL Agents based on What the Agent Learns

- Value-based
  - Explicit: Value function
  - Implicit: Policy (can derive a policy from value function)
- Policy-based
  - Explicit: policy
  - No value function
- Actor-Critic:
  - Explicit: policy and value function

Types of RL Agents on if there is model

- Model-based
  - Explicit: model
  - May or may not have policy and/or value function
- Model-free
  - Explicit: value function and/or policy function
  - No model.

Types of RL Agents



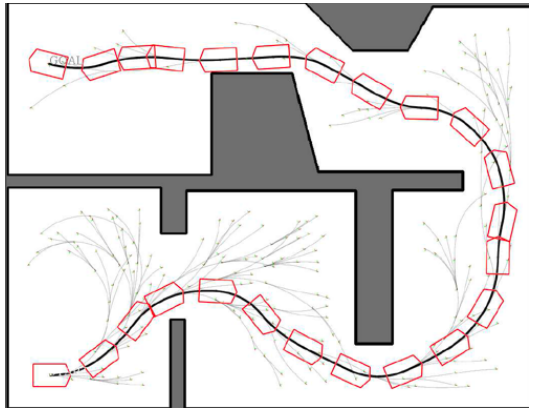
Credit: David Silver's slide

Two Fundamental Problems in Sequential Decision Making

- Planning
  - Given model about how the environment works.
  - Compute how to act to maximize expected reward without external interaction.
- Reinforcement learning
  - Agent doesn’t know how world works
  - Interacts with world to implicitly learn how world works
  - Agent improves policy (also involves planning)

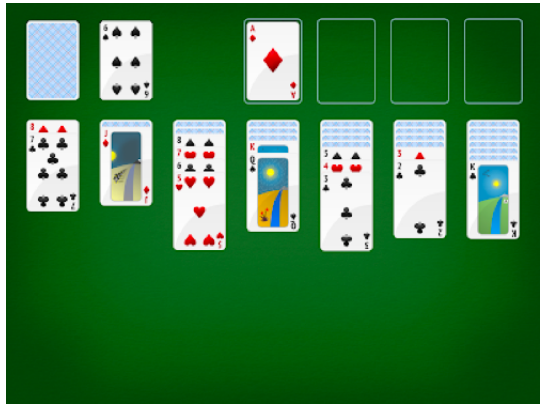
Planning

Path planning



Map is known
All the rules of the vehicle are known
Planning algorithms: dynamic programming, A* search, tree search, …

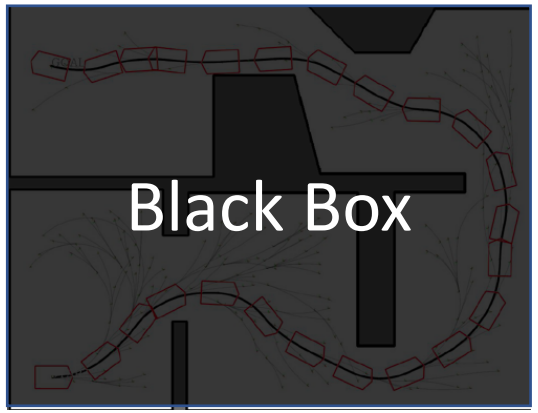
Patience (单人纸牌游戏)



Rules of the game are known.
Planning algorithms: dynamic programming, tree search

Reinforcement Learning

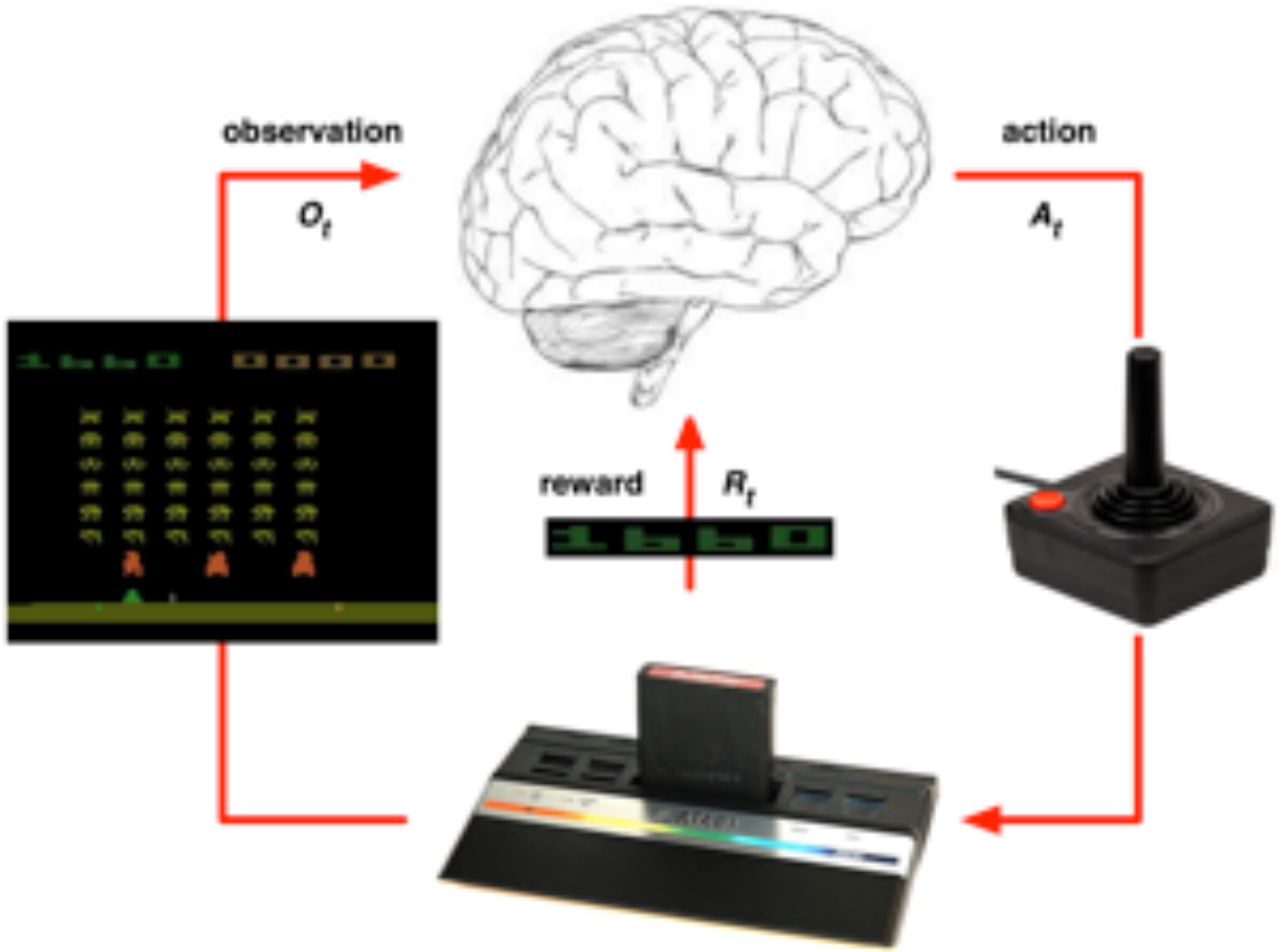
Path planning



Patience (单人纸牌游戏)

- No rules or knowledge about the environment.
- Learn directly by taking actions and seeing what happens.
- Try to find a good policy over time that yields high reward.
- Planning is needed in inference or forward pass.

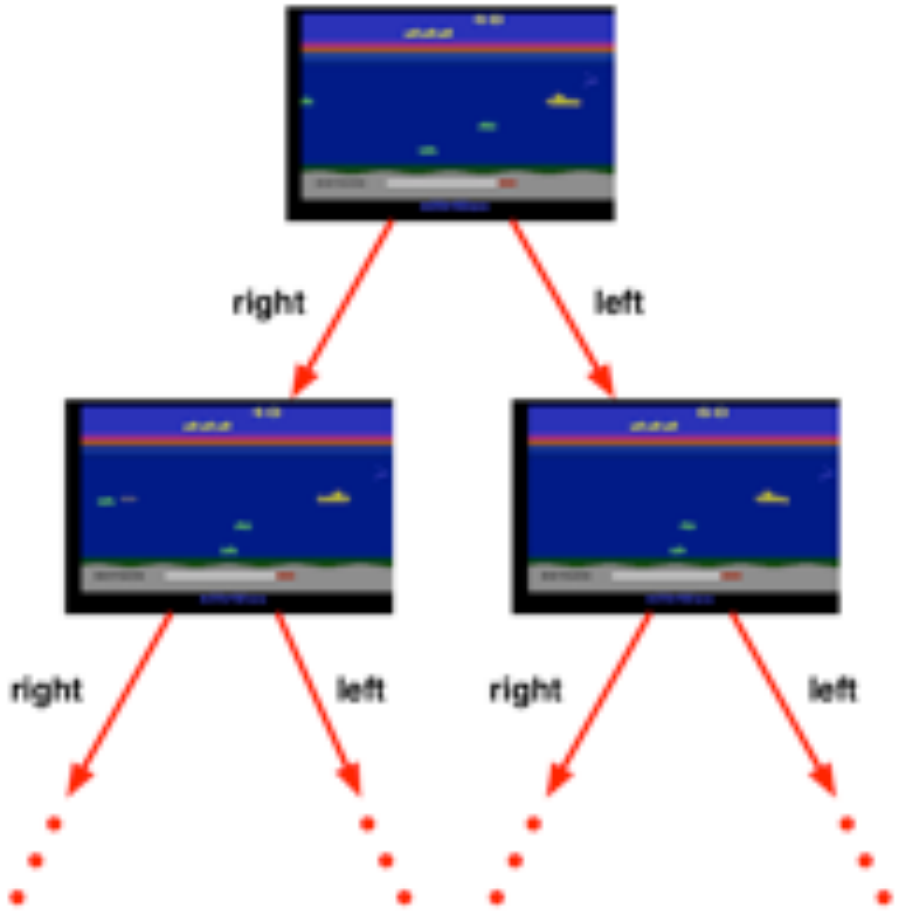
Atari Example: Reinforcement Learning



- Rules of the game are unknown
- Learn directly from interactive game-play
- Pick actions on joystick, see pixels and scores

Atari Example: Planning

- Rules of the game are known
- Can query emulator
  - perfect model inside agent's brain
- If I take action *a* from state *s*:
  - what would the next state be?
  - what would the score be?
- Plan ahead to find optimal policy
  - e.g. tree search



Exploration and Exploitation

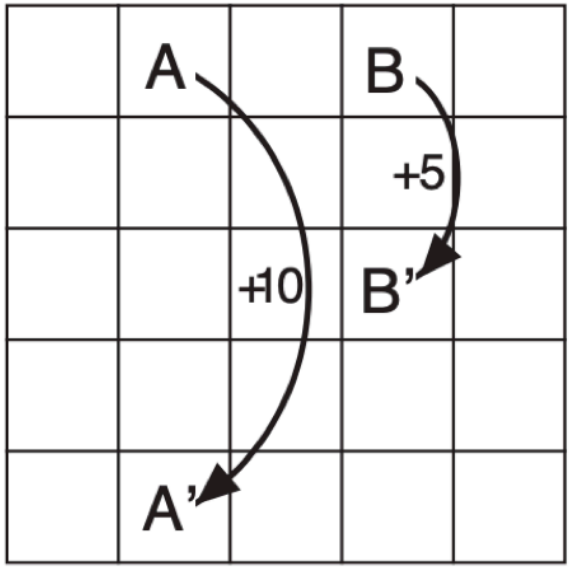
- Agent only experiences what happens for the actions it tries!
- How should an RL agent balance its actions?
  - Exploration: trying new things that might enable the agent to make better decisions in the future
  - Exploitation: choosing actions that are expected to yield good reward given past experience
- Often there may be an exploration-exploitation tradeoff
  - May have to sacrifice reward in order to explore & learn about potentially better policy

Exploration and Exploitation

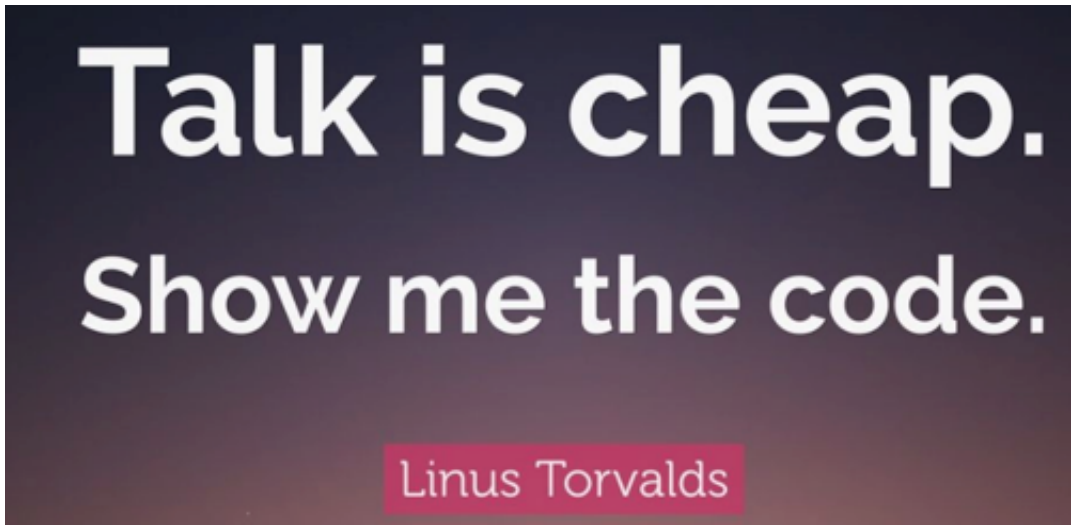
- Restaurant Selection
  - Exploitation: Go to your favourite restaurant
  - Exploration: Try a new restaurant
- Online Banner Advertisements
  - Exploitation: Show the most successful advert
  - Exploration: Show a different advert
- Oil Drilling
  - Exploitation: Drill at the best known location
 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

One exercise (Gridworld example)

- Sutton & Barto: Example 3.5, Exercise 3.14 – Exercise 3.16, Example 3.8.



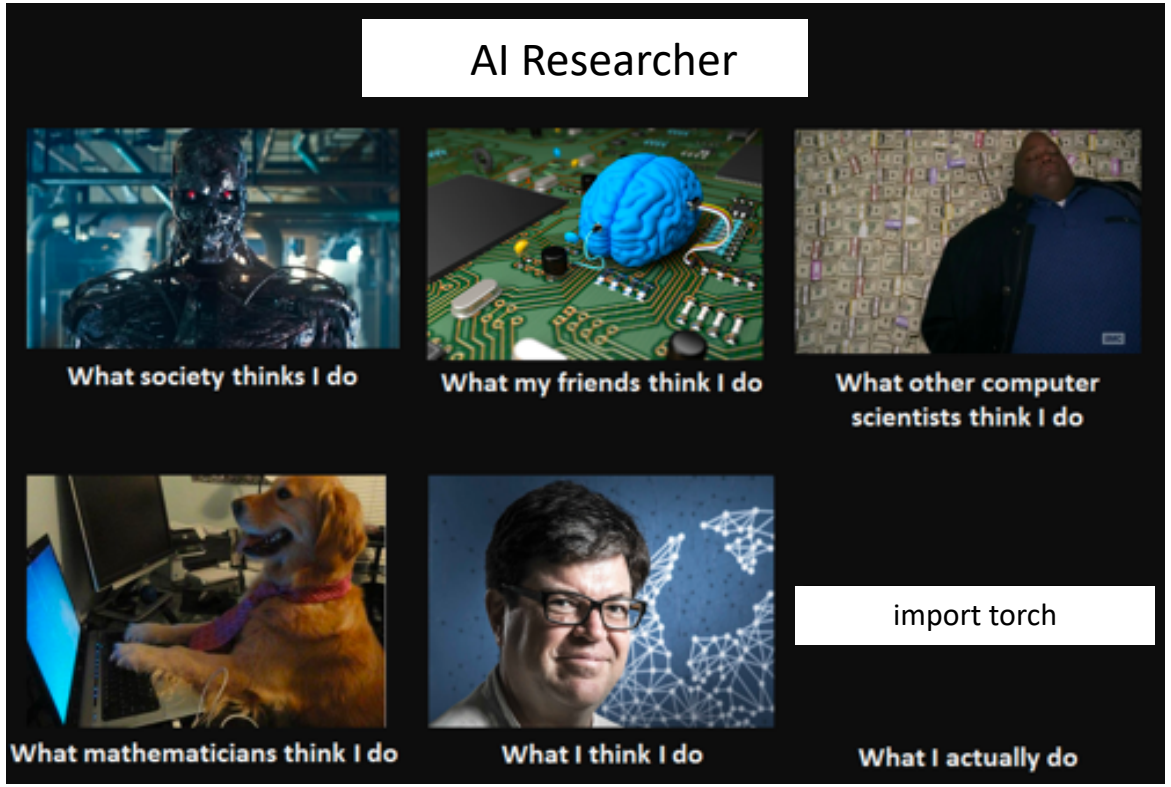
Coding with RL



- Getting hand dirty on reinforcement learning is very important
- Deep learning and AI become more and more empirical
- Trial and error approach to learn reinforcement learning

Coding

- Python coding
- Deep learning libraries: PyTorch or TensorFlow
- https://github.com/cuhkrlcourse/RLexample

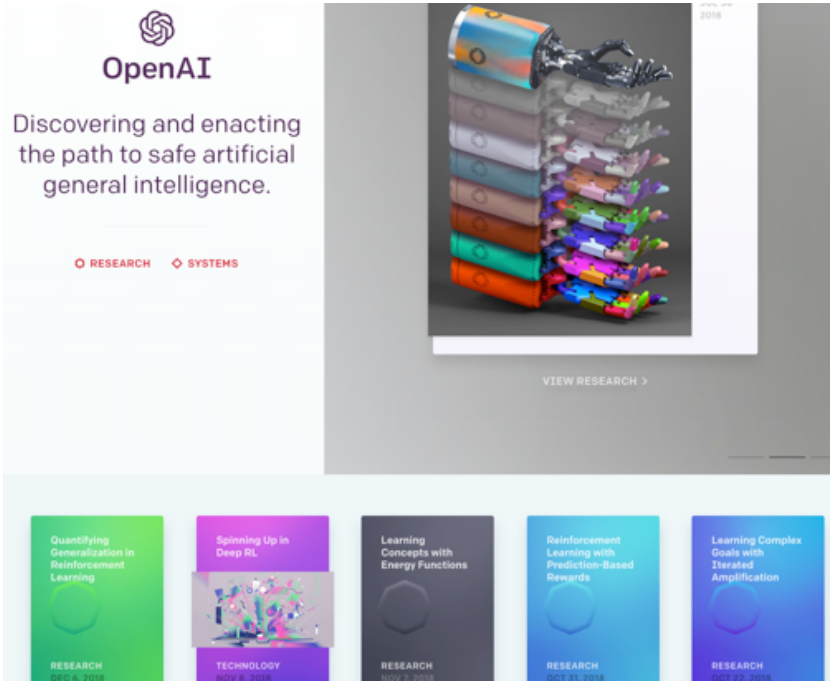


Reinventing Wheels? (造轮子？)

No. Start with existing libraries and pay more attentions to the specific algorithms

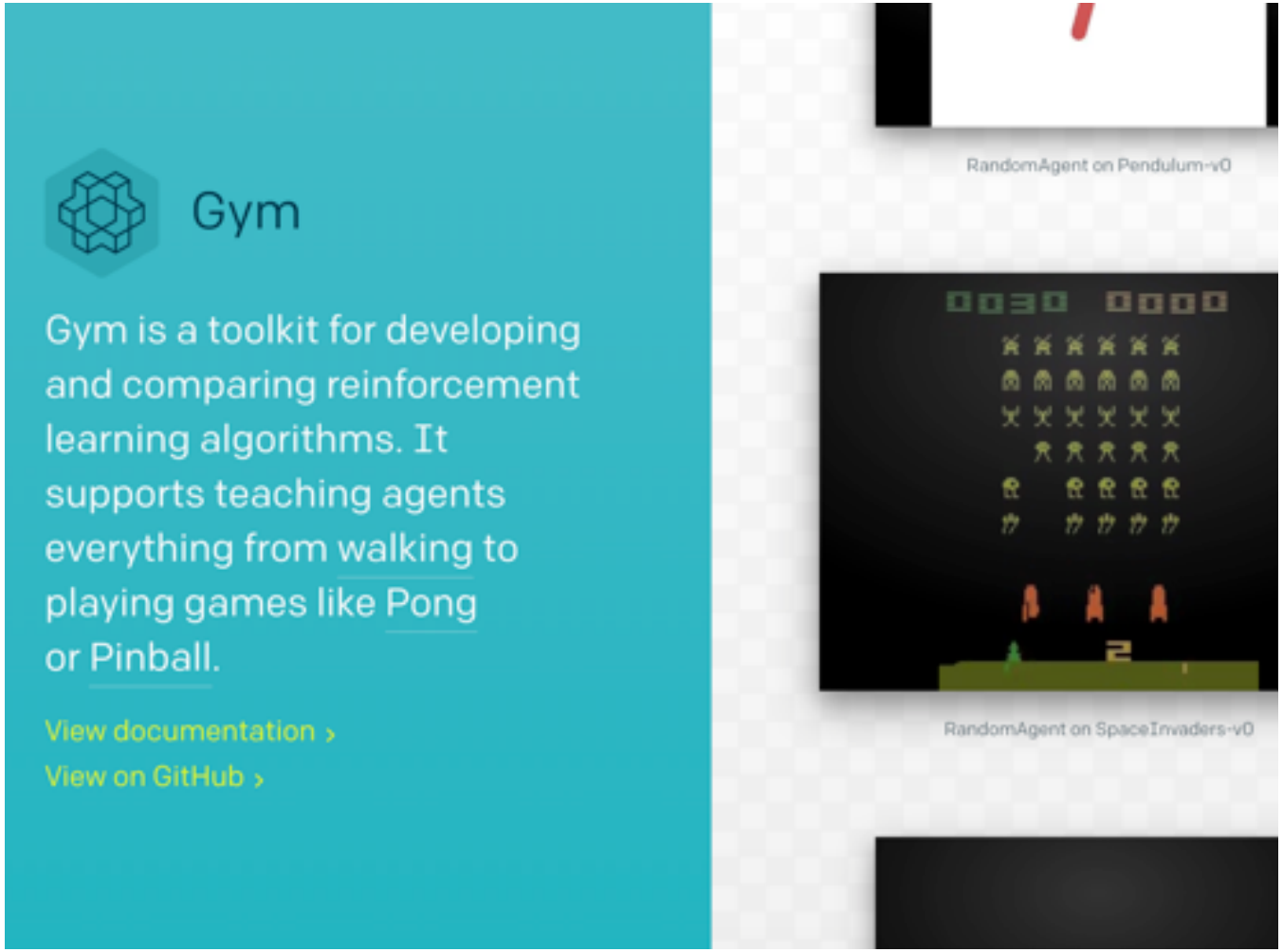
OpenAI: specialized in Reinforcement Learning

- https://openai.com/
- OpenAI is a non-profit AI research company, discovering and enacting the path to safe artificial general intelligence (AGI).



OpenAI gym library

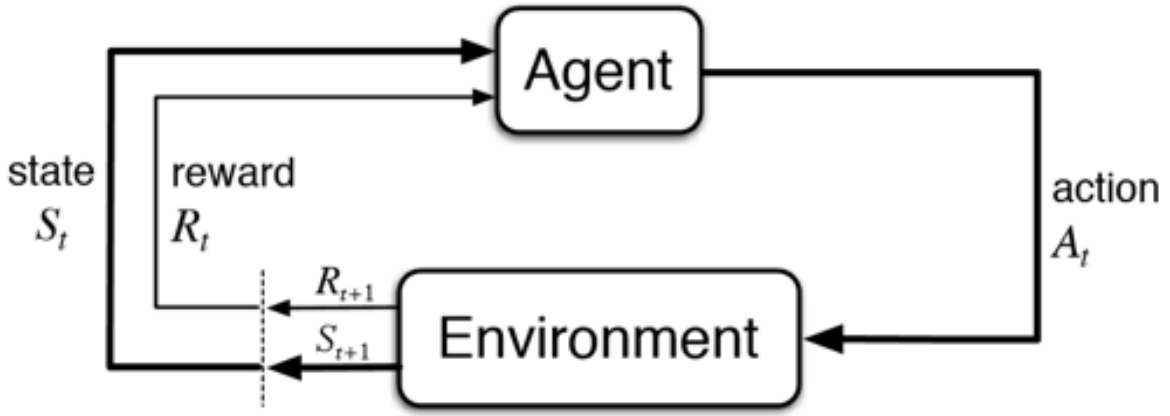
https://gym.openai.com/



https://github.com/openai/retro

We're releasing the full version of Gym Retro, a platform for reinforcement learning research on games. This brings our publicly-released game count from around 70 Atari games and 30 Sega games to over 1,000 games across a variety of backing emulators. We're also releasing the tool we use to add new games to the platform.

Algorithmic interface of reinforcement learning




```
import gym
env = gym.make("Taxi-v2")
observation = env.reset()
agent = load_agent()
for step in range(100):
    action = agent(observation)
    observation, reward, done, info = env.step(action)
```

Classic Control Problems

Acrobot-v1
Swing up a two-link robot.

CartPole-v1
Balance a pole on a cart.

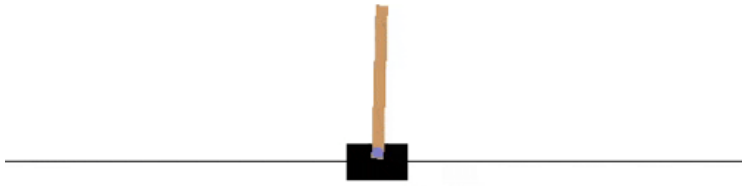
MountainCar-v0
Drive up a big hill.

MountainCarContinuous-v0
Drive up a big hill with continuous control.

Pendulum-v0
Swing up a pendulum.

https://gym.openai.com/envs/#classic_control

Example of CartPole-v0



Actions

Type: Discrete(2)

Num	Action
0	Push cart to the left
1	Push cart to the right

Observation

Type: Box(4)

Num	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -41.8°	~ 41.8°
3	Pole Velocity At Tip	-Inf	Inf

Reward

Reward is 1 for every step taken, including the termination step

Episode Termination

1. Pole Angle is more than ±12°
2. Cart Position is more than ±2.4 (center of the cart reaches the edge of the display)
3. Episode length is greater than 200

https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py

Example code

```
import gym
env = gym.make('CartPole-v0’)
env.reset()
env.render() # display the rendered scene
action = env.action_space.sample()
observation, reward, done, info = env.step(action)
```

Example code: Random Agent

python my_random_agent.py CartPole-v0

python my_random_agent.py Pong-ram-v0

python my_random_agent.py Breakout-v0

What is the difference in the format of the observations?

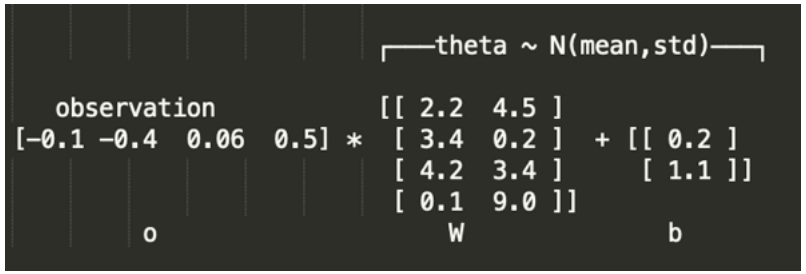
Example code: Naïve learnable RL agent

python my_random_agent.py CartPole-v0

python my_random_agent.py Acrobot-v1

python my_learning_agent.py CartPole-v0

python my_learning_agent.py Acrobot-v1



P_a = oW+b

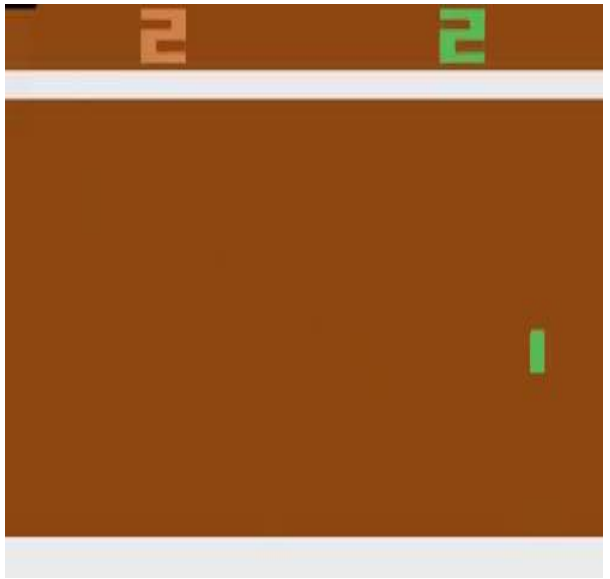
What is the algorithm?

Cross Entropy method (CEM)
https://gist.github.com/kashif/5dfa12d80402c559e060d567ea352c06

Deep Reinforcement Learning Example

- Pong example

```
import gym
env = gym.make('Pong-v0')
env.reset()
env.render() # display the rendered scene
```



```
python my_random_agent.py Pong-v0
```

Deep Reinforcement Learning Example

- Pong example

python pg-pong.py

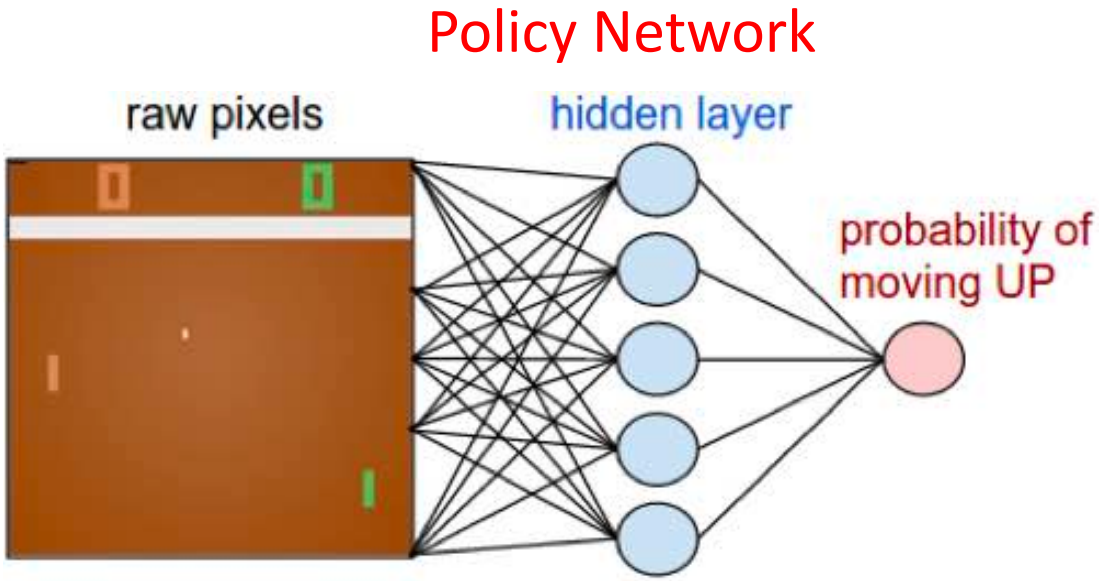
Loading weight: pong_bolei.p (model trained over night)

Deep Reinforcement Learning Example

- Look deeper into the code

```
observation = env.reset()

cur_x = prepro(observation)
x = cur_x - prev_x
prev_x = cur_x
aprob, h = policy_forward(x)
```



Randomized action:

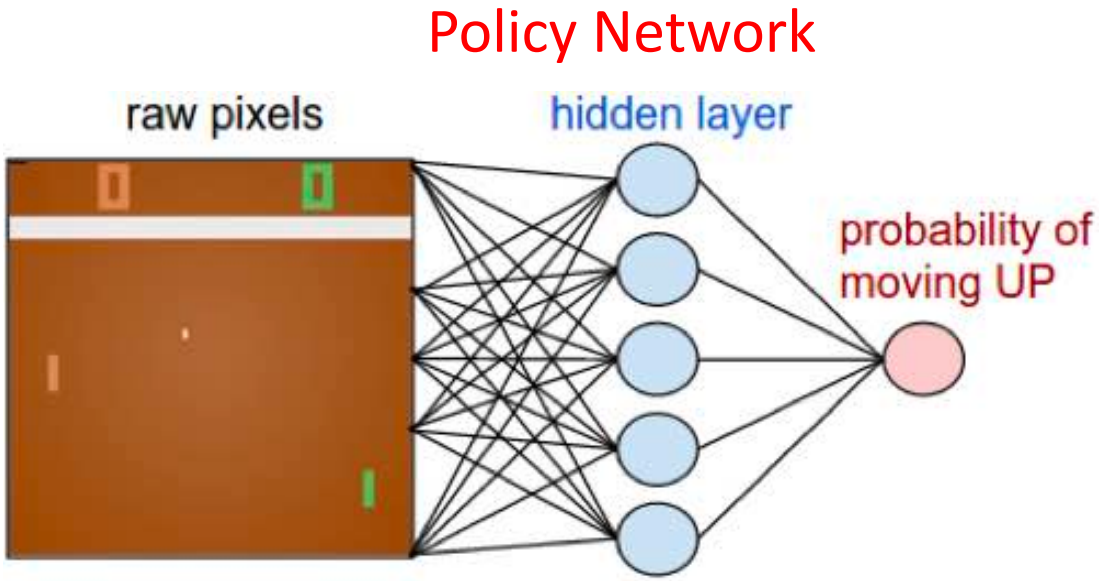
```
action = 2 if np.random.uniform() < aprob else 3 # roll the dice!
```

http://karpathy.github.io/2016/05/31/rl/

Deep Reinforcement Learning Example

- Look deeper into the code

```
h = np.dot(W1, x)
h[h<0] = 0 # ReLU nonlinearity: threshold at zero logp = np.dot(W2, h) # compute log probability of going up
p = 1.0 / (1.0 + np.exp(-logp)) # sigmoid function (gives probability of going up)
```



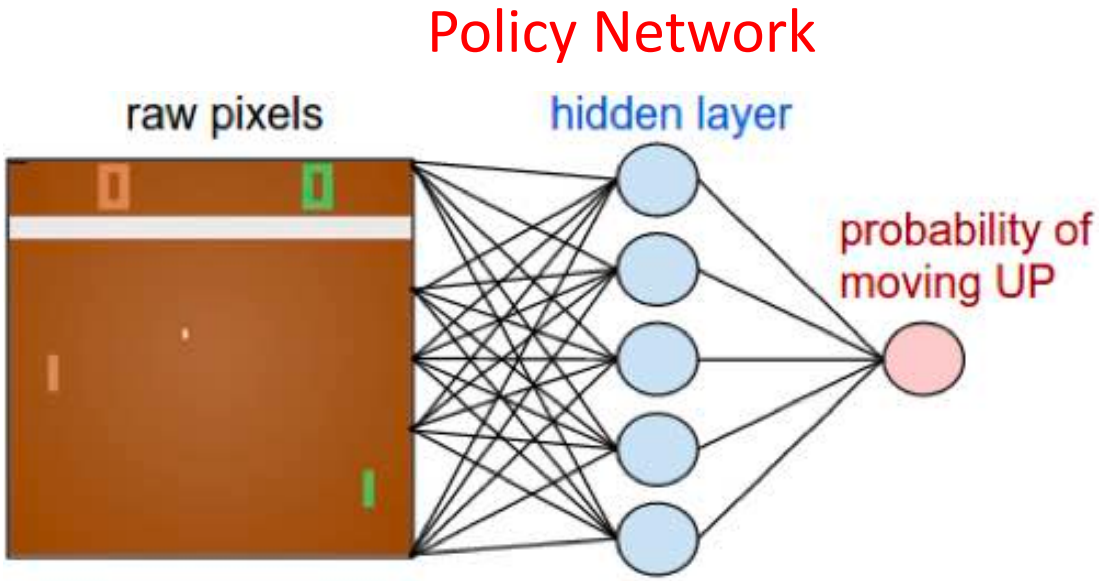
http://karpathy.github.io/2016/05/31/rl/

Deep Reinforcement Learning Example

- Look deeper into the code

How to optimize the W1 and W2?

Policy Gradient! (To be introduced in future lecture)



http://karpathy.github.io/2016/05/31/rl/

What could be the potential problems?

```
import gym
env = gym.make("Taxi-v2")
observation = env.reset()
agent = load_agent()
for step in range(100):
    action = agent(observation)
    observation, reward, done, info = env.step(action)
```

Speed, multiple agents, structure of agent?

Competitive Pong Environment Demo

- https://github.com/cuhkrlcourse/competitive-rl

Homework and What's Next

- Play with OpenAI gym and the example code
https://github.com/cuhkrlcourse/RLexample
- Play with the competitive pong env
https://github.com/cuhkrlcourse/competitive-rl
- Try to understand my_learning_agent.py
- Go through this blog in detail to understand pg-pong.py
http://karpathy.github.io/2016/05/31/rl/

- Next week: Markov Decision Process, policy iteration, and value iteration
- Please read **Sutton and Barton: Chapter 1 and Chapter 3**

In case you need it

- Python tutorial: http://cs231n.github.io/python-numpy-tutorial/
- Tensorflow tutorial: https://www.tensorflow.org/tutorials/
- PyTorch tutorial: https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html