Deep Q-Learning

Prof. Seungchul Lee

Industrial AI Lab.

Source

- David Silver's Lecture (DeepMind)
  - UCL homepage for slides (http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html)
  - DeepMind for RL videos (https://www.youtube.com/watch?v=2pWv7GOvuf0)
  - An Introduction to Reinforcement Learning, Sutton and Barto pdf

- CMU by Zico Kolter
  - http://www.cs.cmu.edu/~zkolter/course/15-780-s14/lectures.html
  - https://www.youtube.com/watch?v=un-FhSC0HfY&hd=1

- Deep RL Bootcamp by Rocky Duan
  - https://sites.google.com/view/deep-rl-bootcamp/home
  - https://www.youtube.com/watch?v=qO-HUo0LsO4

- Stanford Univ. by Serena Yeung
  - https://www.youtube.com/watch?v=lvoHnicueoE&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv&index=15&t=1337s

(Tabular) Q-Learning

- Value iteration algorithm: use Bellman equation as an iterative update

$$Q_{k+1}(s,a) \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s) + \gamma \max_{a'} Q_k(s', a') \right]$$

- Q_k will converge to Q_* as goes to ∞

$$\text{target}(s') = R(s) + \gamma \max_{a'} Q_k(s', a')$$

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha \left(\text{target}(s') - Q_k(s,a) \right)$$

- Q-table

$Q(s,a) \rightarrow$

	s_1	s_2	s_3	s_4	s_5
a_1	4.21	4.88	5.74	6.25	8.51
a_2	3.72	4.02	4.48	5.13	5.22

- Amazing result: Q-learning converges to optimal policy if all state-action pairs seen frequently enough
- However, what we have learned so far has nothing to do with deep learning

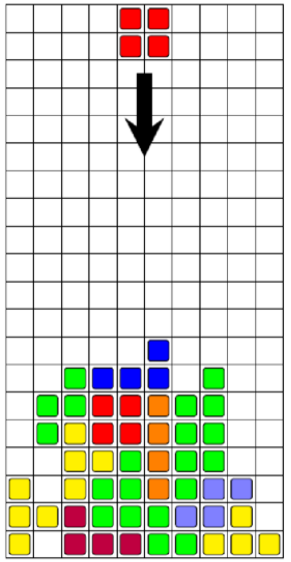
What is the Problem with Q-Learning?

- Not scalable. Must compute tabular $Q(s, a)$ for every state-action pair.

- If state is for instance current game state pixels, computationally infeasible to compute for entire state space !
  - Too many states to visit them all in training
  - Too many states to hold the Q-table in memory

- States in Tetris: naive board configuration + shape of the falling piece $\sim 10^{60}$ states !!!

- Solution: use a function approximator to estimate $Q(s, a)$
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations



Tetris

Approximate Q-Learning

$$\text{target}\,(s') = R(s) + \gamma \max_{a'} Q_\omega(s', a')$$

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha \underbrace{(\text{target}\,(s') - Q_k(s,a))}_{\text{difference}}$$

- Linear function approximator

$$Q_\omega(s,a) = \omega_0 f_0(s,a) + \omega_1 f_1(s,a) + \cdots + \omega_n f_n(s,a)$$

- Exact Q

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}]$$

- Approximate Q

$$\omega_i \leftarrow \omega_i + \alpha\,[\text{difference}]\, f_i(s,a) \qquad ?$$

Linear Function Approximator

- Approximate Q

$$\omega_i \leftarrow \omega_i + \alpha\,[\text{difference}]\, f_i(s,a)$$

- Loss function

$$\ell = \frac{1}{2}(y - \hat{y})^2$$

- Difference

$$y - \hat{y} = y - (\omega_0 f_0(x) + \omega_1 f_1(x) + \cdots + \omega_n f_n(x)) = \underbrace{y}_{\text{target}} - \underbrace{\sum_i \omega_i f_i(x)}_{\text{prediction}}$$

- For one point

$$\frac{\partial \ell(\omega)}{\partial \omega_k} = -\left(y - \sum_i \omega_i f_i(x)\right) f_k(x)$$

$$\omega_k \leftarrow \omega_i + \alpha \left(y - \sum_i \omega_i f_i(x)\right) f_k(x)$$

$$\omega_k \leftarrow \omega_i + \alpha\,[\text{difference}]\, f_k(x)$$

Non-Linear Function Approximator: Deep Q-Learning Network

- Difference

$$\text{difference} = \left[R(s) + \gamma \max_{a'} Q_\omega(s', a')\right] - Q_\omega(s, a)$$

- Loss function by mean-squared error in Q-value

$$\ell(\omega) = \left[\frac{1}{2}\left(\text{target}\,(s') - Q_\omega(s, a)\right)^2\right]$$

- GD update:

$$\omega_{k+1} \leftarrow \omega_k - \alpha\, \nabla_\omega \ell(\omega) \qquad \Longrightarrow \text{ Deep Learning Framework}$$

$$\leftarrow \omega_k - \alpha\, \nabla_\omega \left[\frac{1}{2}\left(\text{target}\,(s') - Q_\omega(s, a)\right)^2\right]$$

$$\leftarrow \omega_k + \alpha\, \left[\left(\text{target}\,(s') - Q_\omega(s, a)\right)\right] \nabla_\omega Q_\omega(s, a)$$

$$\omega_i \leftarrow \omega_i + \alpha\, [\text{difference}]\, f_i(s, a)$$

TD Method with Value Function Approximation

- Consider some parameterized value function approximator

$$Q_\omega(s,a) \approx Q_*(s,a)$$

where Q is e.g. a neural network, and ω denote the network weights

- Then TD update is given by

$$\omega_{k+1} \leftarrow \omega_k + \alpha \left[(\text{target}\,(s') - Q_\omega(s,a))\right] \nabla_\omega Q_\omega(s,a)$$

which is a generalization of traditional TD update

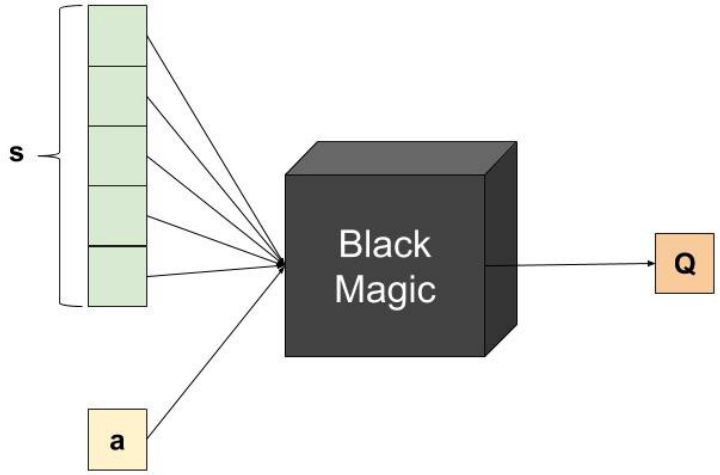
- Equivalently, loss function for DQN $\ell(\omega) = \left[\frac{1}{2}(\text{target}\,(s') - Q_\omega(s,a))^2\right]$

```
Q = net(ph_states)

loss = tf.reduce_mean([tf.square(ph_Q_target[i] - Q[i][action]) for i, action in enumerate(tf.unstack(ph_action))])

optm = tf.train.AdamOptimizer(LR).minimize(loss)
```

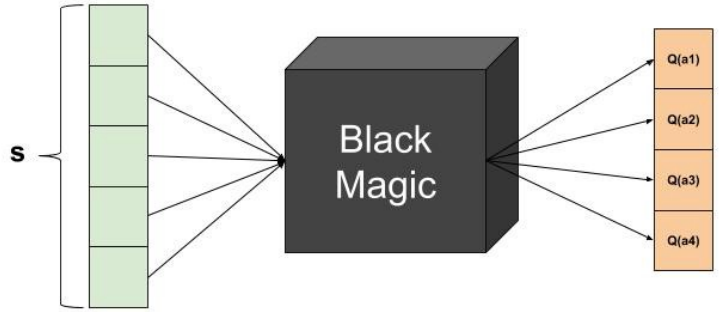
Deep Q-Networks (DQN)

- Approximate Q learning using a neural network → Deep Q-learning Network

- (Tabular) Q-Learning like DQN, but inefficient



- Better DQN



Deep Q-Learning

- Instead of a table, we have a parameterized Q function:

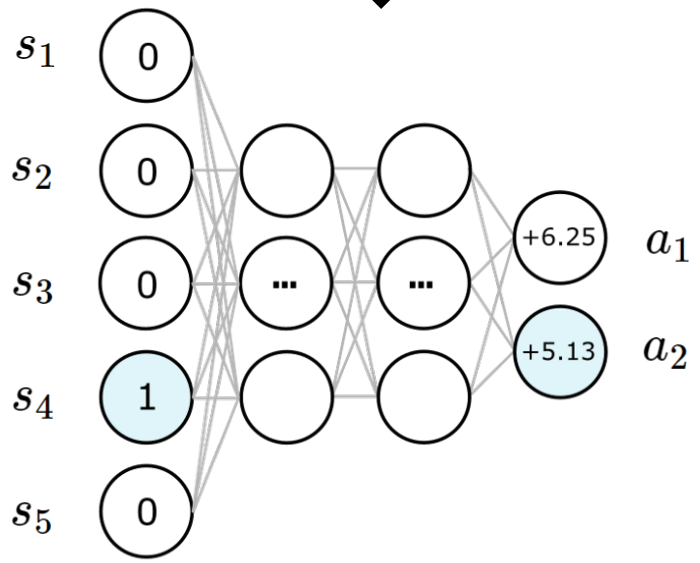
$Q(s, a) \rightarrow$

	s_1	s_2	s_3	s_4	s_5
a_1	4.21	4.88	5.74	6.25	8.51
a_2	3.72	4.02	4.48	5.13	5.22

- Neural network as universal function approximates

Finally Deep Learning

$Q(s, a) \rightarrow$



Deep Q-Networks (DQN)

- DQN method (Q-learning with deep network as Q function approximator) became famous in 2013 for learning to play a wide variety of Atari games better than humans.

$$\ell(\omega) = \left[\frac{1}{2}(\text{target}\,(s') - Q_\omega(s,a))^2\right]$$

$$\omega_{k+1} \leftarrow \omega_k + \alpha\ [(\text{target}\,(s') - Q_\omega(s,a))]\,\nabla_\omega Q_\omega(s,a)$$

- Deep Mind, 2015
  - Used a deep learning network to represent Q
  - A few additional tricks:
    - keep first Q function above fixed and only update every C iterations
    - keep replay buffer of past actions

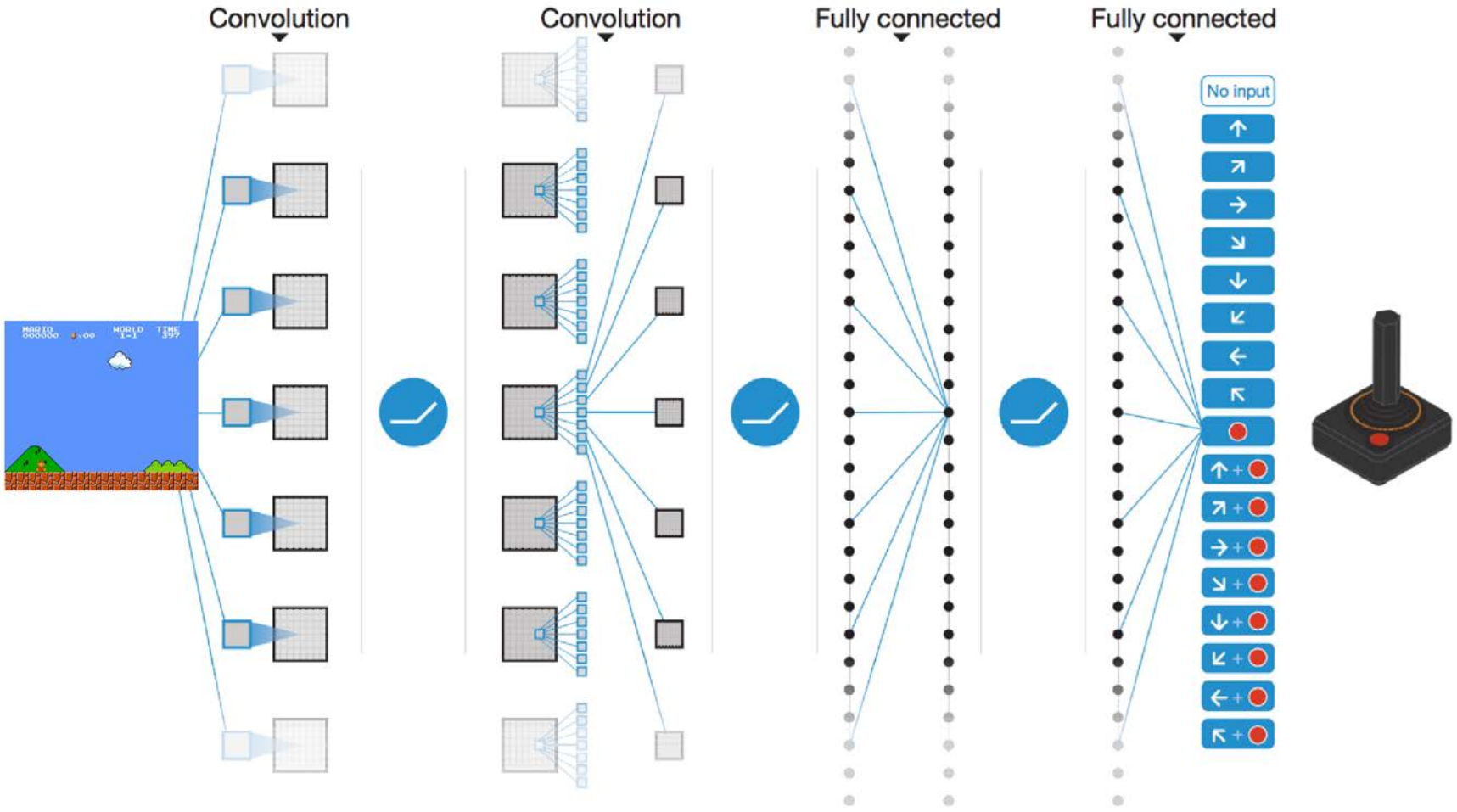
LETTER

doi:10.1038/nature14236

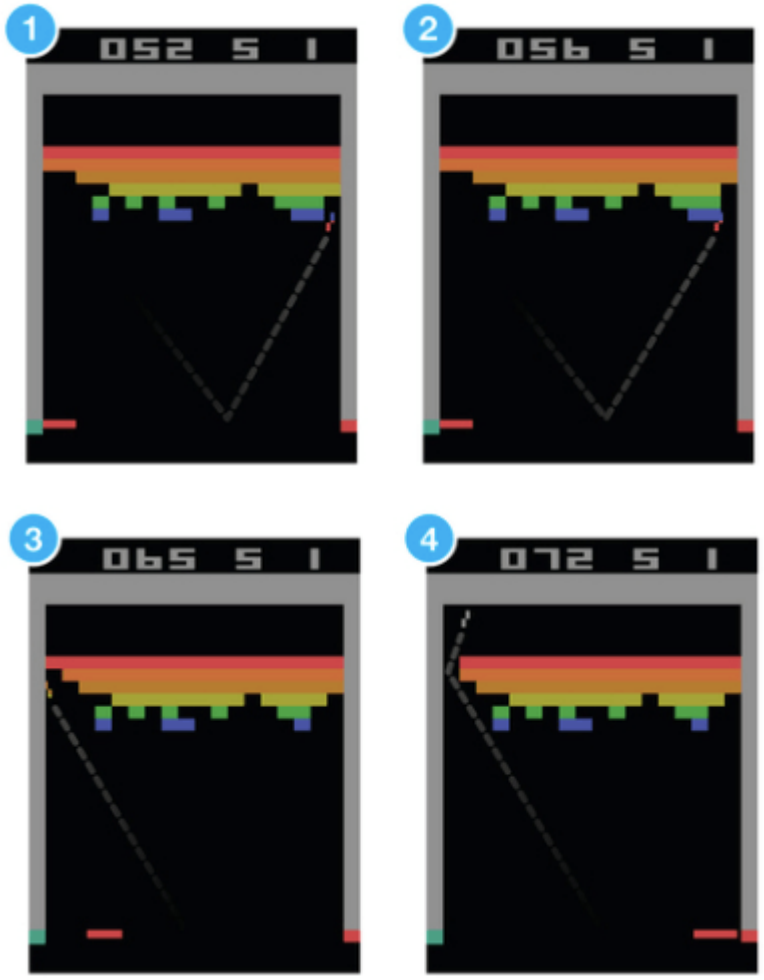
Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

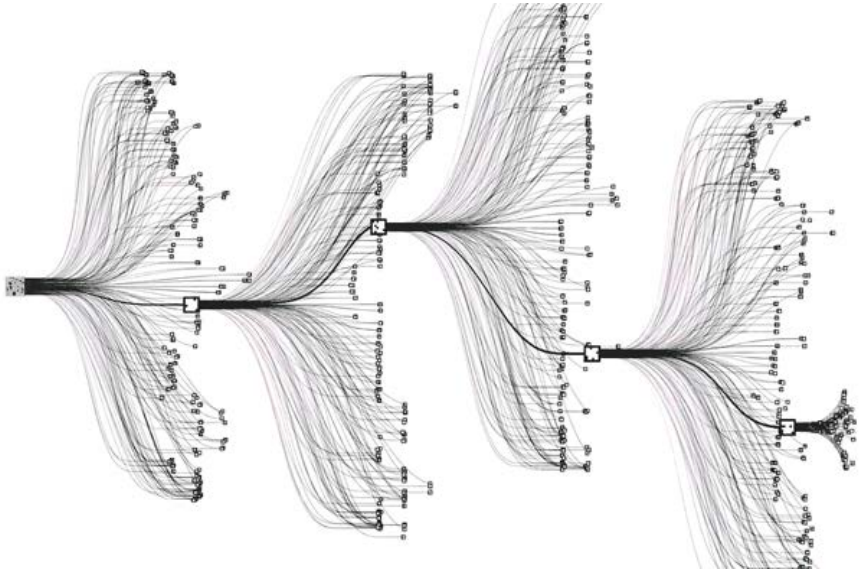
Deep Convolutional Reinforcement Learning



Google DeepMind's Atari Breakout

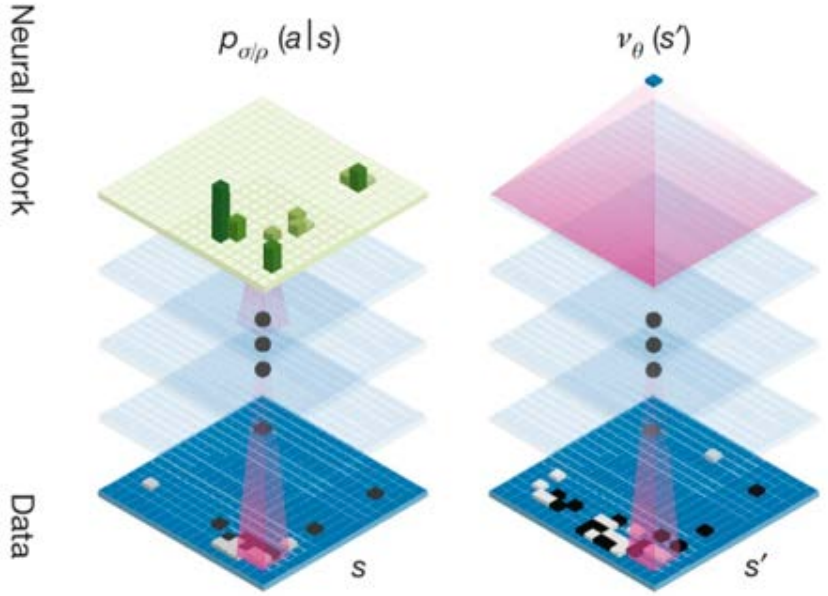


AlphaGo



- Think about all the possible cases ?

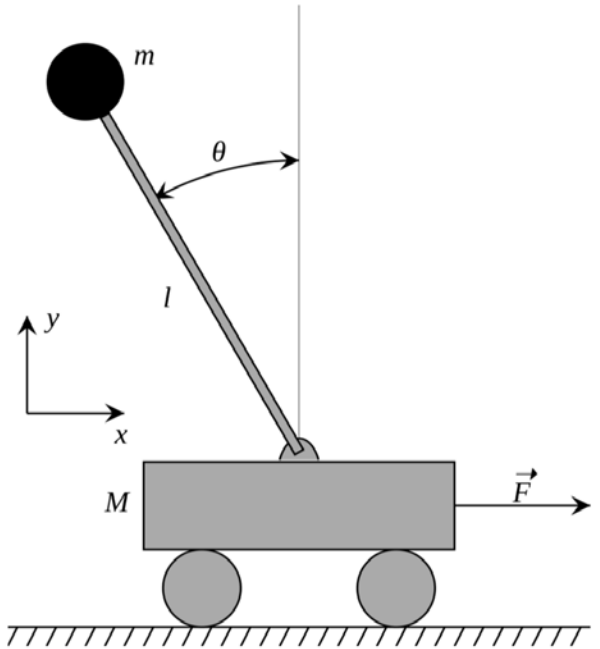
- Deep Convolutional Reinforcement Learning



Lab: DQN with Gym Environment

- Cart Pole Problem
  - Objective:
    - Balance a pole on top of a movable cart
  - State:
    - position, horizontal velocity, angle, angular speed
  - Action:
    - horizontal force applied on the cart
  - Reward:
    - 1 at each time step if the pole is upright

- Hands-on at the lab session



Summary

From MDP To Reinforcement Learning

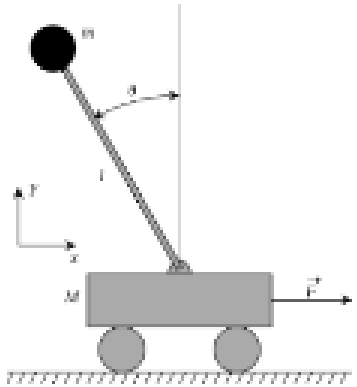
- You should take good actions to get rewards, but in order to know which actions are good, we need to explore and try different actions.

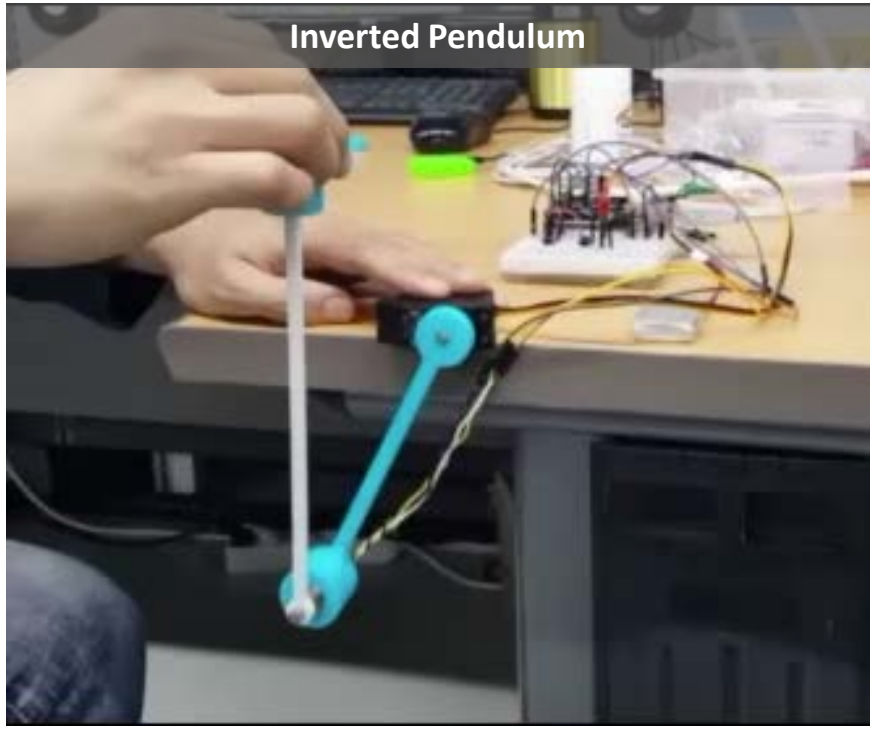
- Markov decision process (offline)
  - Have a model of how the world works.
  - Find policy to collect maximum rewards.

- Reinforcement learning (online)
  - Don't know how the world works.
  - Perform actions in the world to find out and collect rewards.

Control Inverted Pendulum

- From open-loop to closed-loop systems



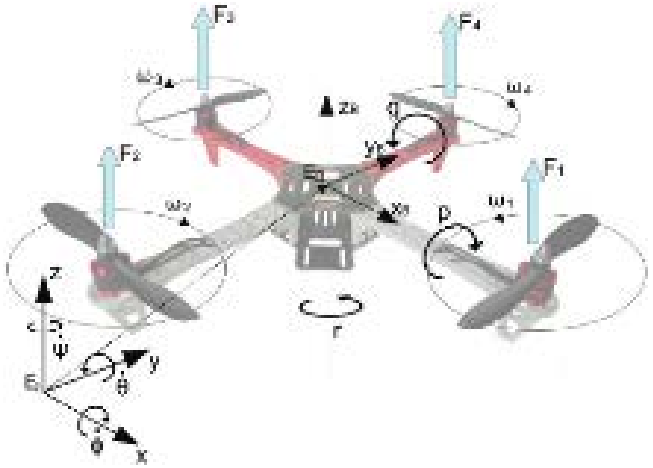


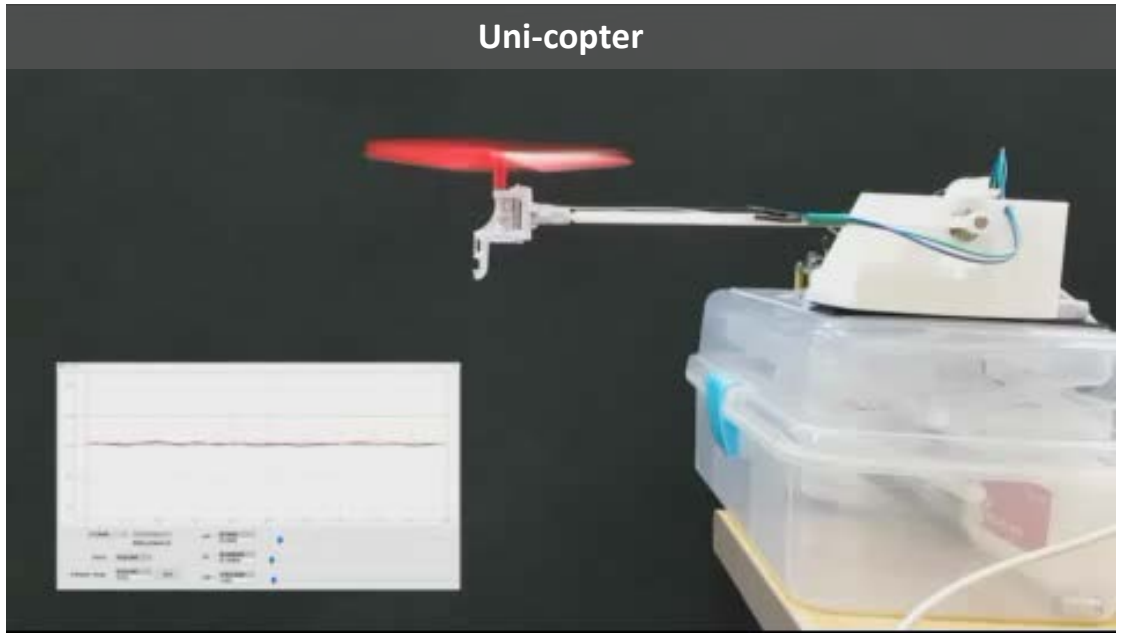
Reinforcement Learning

- Software-in-the-loop

Control Uni-copter

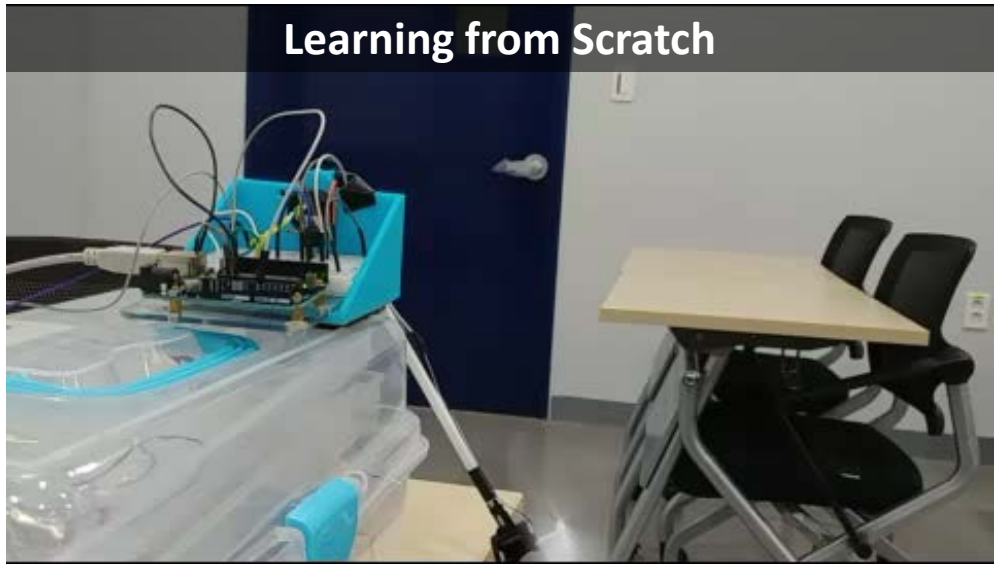
- From open-loop to closed-loop systems

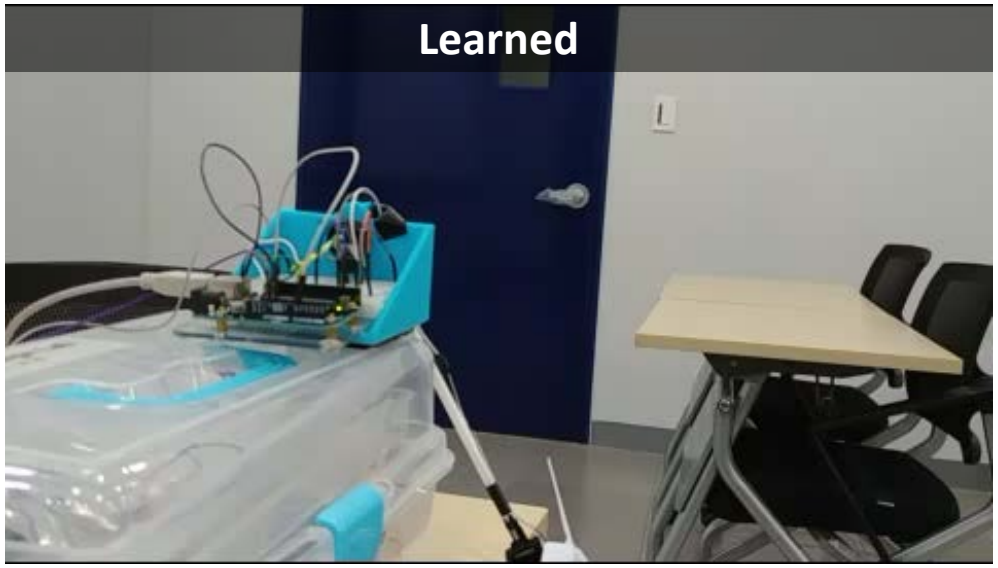




Reinforcement Learning

- Hardware-in-the-loop





Reinforcement Learning

- Learned knowledge can be transferred