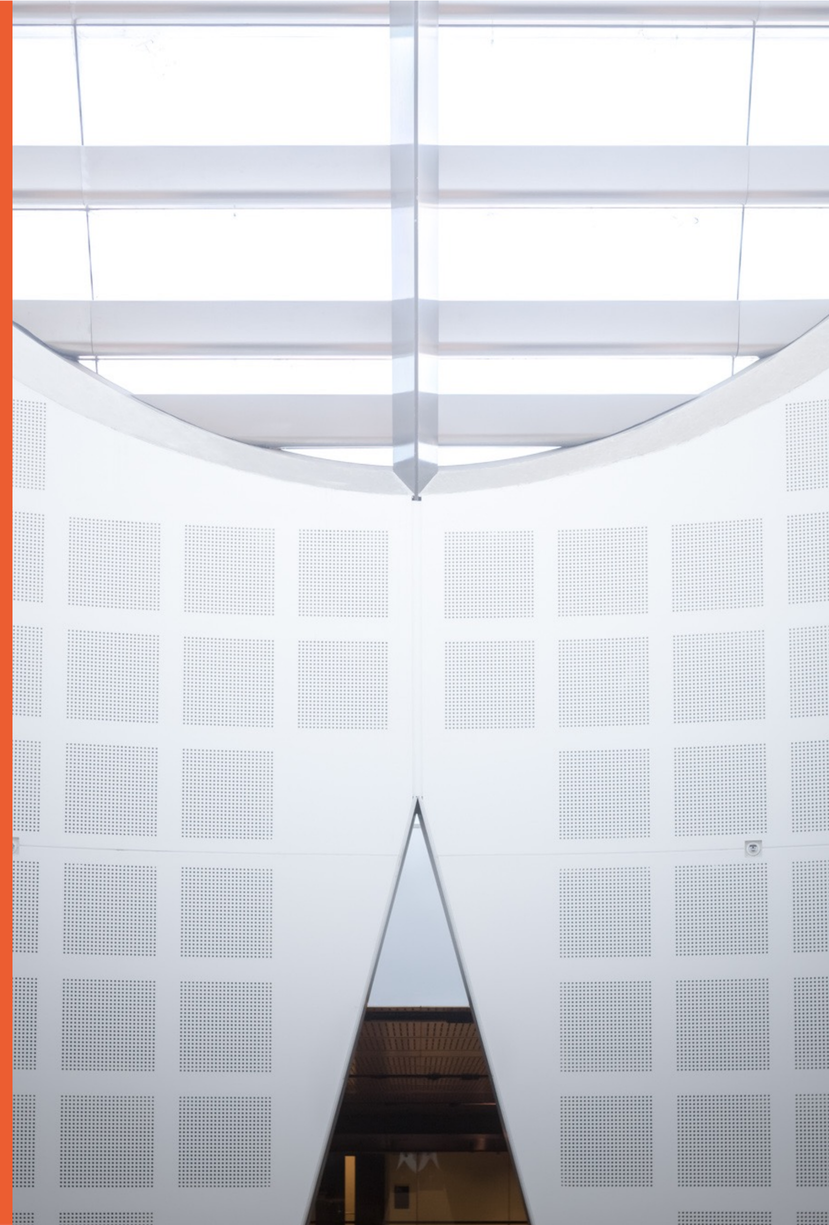
# Deep Reinforcement Learning

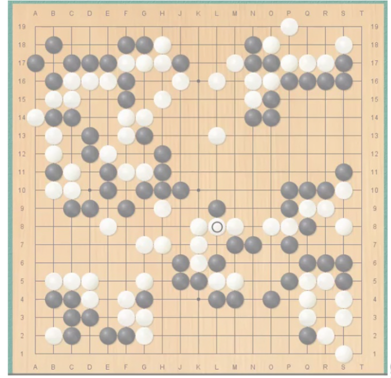
Dr Chang Xu

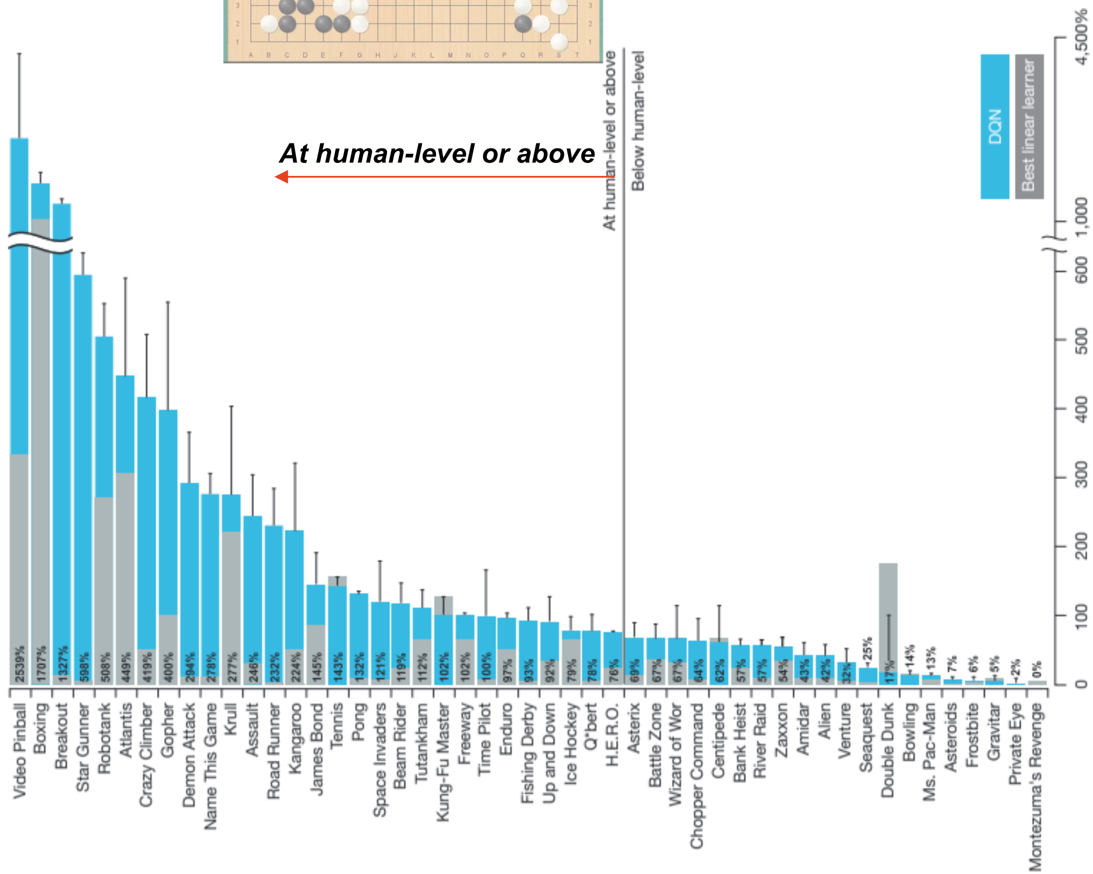
School of Computer Science

THE UNIVERSITY OF
SYDNEY



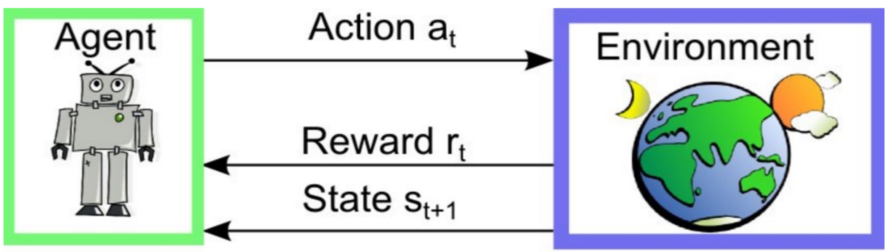
# Deep RL Breakthroughs

# Reinforcement Learning (RL) in a nutshell

- RL is a general-purpose framework for decision making
  - RL is for an agent to act with an environment
  - Each action influences the agent's future state
  - Feedback is given by a scalar reward signal
  - Goal: select actions to maximize future reward

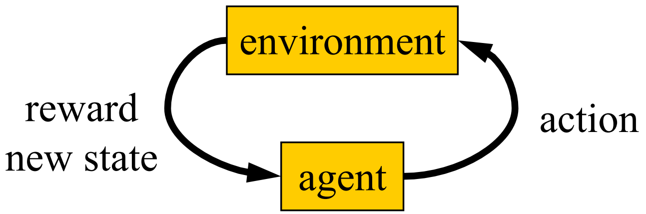



Actions: muscle contractions
Observations: sight, smell
Rewards: food

From the talk *Introduction to Deep Reinforcement Learning From Theory to Applications* by Siyi Li (HKUST)

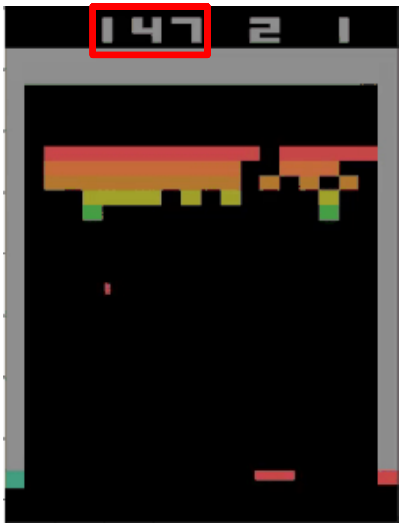
# Markov Decision Process (MDP)

- set of states S, set of actions A, initial state $S_0$
- transition model P(s, a, s')
  - P( $\text{frame}_{(t)}$, right, $\text{frame}_{(t')}$) = 0.8
- reward function r(s)
  - r($\text{frame}_{(t)}$) = +1
- goal: maximize cumulative reward in the long run

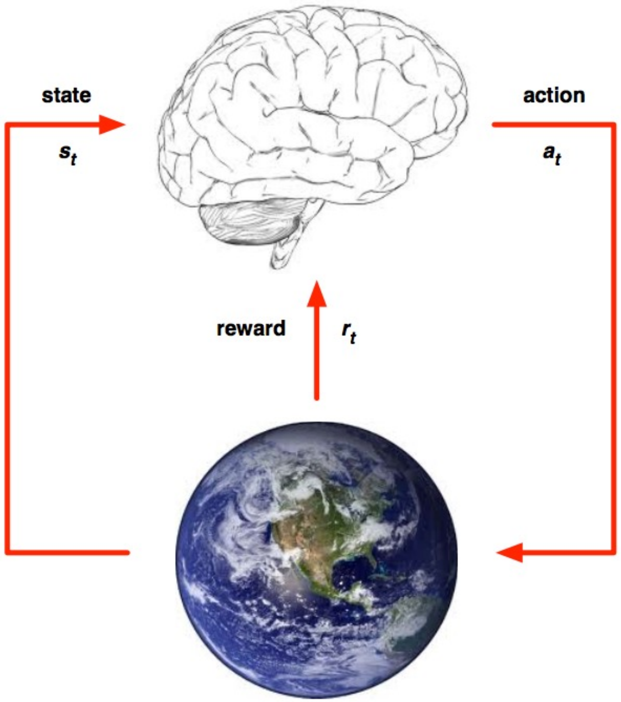



- policy: mapping from S to A
  - $a=\pi(s)$ or $\pi(s, a)$ (deterministic vs. stochastic)

- reinforcement learning
  - transitions and rewards usually not available
  - how to change the policy based on experience
  - how to explore the environment



# Markov Decision Processes (MDPs)




- At each step $t$ the agent:
  - Receives state $s_t$
  - Receives scalar reward $r_t$
  - Executes action $a_t$
- The environment:
  - Receives action $a_t$
  - Emits state $s_t$
  - Emits scalar reward $r_t$

From the *Tutorial: Deep Reinforcement Learning* by David Silver, Google DeepMind

# Computing return from rewards

- episodic (vs. continuing) tasks
  - "game over" after N steps
  - optimal policy depends on N; harder to analyze

- additive rewards
  - $V(s_0, s_1, \ldots) = r(s_0) + r(s_1) + r(s_2) + \ldots$
  - infinite value for continuing tasks

- discounted rewards
  - $V(s_0, s_1, \ldots) = r(s_0) + \gamma * r(s_1) + \gamma^2 * r(s_2) + \ldots$
  - value bounded if rewards bounded

The goal of RL is to find the policy which maximizes the expected return.

# Value Function

- A value function is the prediction of the future return
- Two definitions exist for the value function
  - State value function
  - "How much reward will I get from state s?"
  - expected return when starting in s and following π

$$V^{\pi}(s) = \mathbb{E}\left\{ \sum_{k=0}^{\infty} \gamma^k r_k \,\middle|\, s_0 = s, \pi \right\}$$

$$V^{\pi}(s) = \mathbb{E}\left\{ Q^{\pi}(s,a) \middle| \, a \sim \pi(s,\cdot) \right\}$$

  - State-action value function
  - "How much reward will I get from action a in state s?"
  - expected return when starting in s, performing a, and following π

$$Q^{\pi}(s,a) = \mathbb{E}\left\{ \sum_{k=0}^{\infty} \gamma^k r_k \,\middle|\, s_0 = s, a_0 = a, \pi \right\}$$

# Bellman Equation and Optimality

- Value functions decompose into Bellman equations, i.e. the value functions can be decomposed into

  *immediate reward* plus *discounted value of successor state*

$$V^\pi(s) = \mathbb{E}\left\{ \sum_{k=0}^{\infty} \gamma^k r_k \,\middle|\, s_0 = s, \pi \right\} \longrightarrow V^\pi(s) = \mathbb{E}\left\{ R(s, a, s') + \gamma V^\pi(s') \right\}$$

$$Q^\pi(s, a) = \mathbb{E}\left\{ \sum_{k=0}^{\infty} \gamma^k r_k \,\middle|\, s_0 = s, a_0 = a, \pi \right\}$$

$$Q^\pi(s, a) = \mathbb{E}\left\{ R(s, a, s') + \gamma Q^\pi(s', a') \right\}$$

- An optimal value function is the maximum achievable value.

$$V^*(s) = \max_\pi V^\pi(s) \quad Q^*(s, a) = \max_\pi Q^\pi(s, a)$$

# Bellman Optimality Equation

- Optimality for value functions is governed by the Bellman optimality equations.

- Two equations:

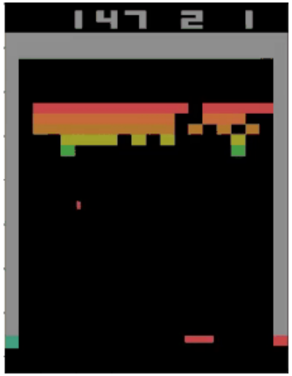
$$V^*(s) = \max_a \mathbb{E}\left\{R(s,a,s') + \gamma V^*(s')\right\}$$

$$Q^*(s,a) = \mathbb{E}\left\{R(s,a,s') + \max_{a'} \gamma Q^*(s',a')\right\}$$

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

# Q-Learning

# Initializing the Q(s, a) function




states

actions

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Noop | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Fire | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Right | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Left | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# Q-Learning

$$Q^*(s,a) = \mathbb{E}\left\{R(s,a,s') + \max_{a'} \gamma Q^*(s',a')\right\}$$

Initialize $Q(s,a)$ arbitrarily.

Start with $s$.

Before taking action a, we calculate the current expected return as

$$Q(s,a)$$

After taking action $a$, and observing $r$ and $s'$, we calculate the target expected return as

$$R(s,a,s') + \max_{a'} \gamma Q(s',a')$$

$$\Delta Q(s,a) = R(s,a,s') + \max_{a'} \gamma Q(s',a') - Q(s,a)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha \Delta Q(s,a)$$

# Q-Learning

$$Q^*(s,a) = \mathbb{E}\left\{R(s,a,s') + \max_{a'} \gamma Q^*(s',a')\right\}$$

Initialize Q(s, a) arbitrarily.

    Repeat (for each episode)

        Initialize *s*

        Repeat (for each step of the episode)

            Choose *a* from *s* using a policy

            Take action *a*, observe *r, s'*

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

$$s \leftarrow s'$$

# Exploration and Exploitation

Initialize Q(s, a) arbitrarily.

Repeat (for each episode)

Initialize *s*

Repeat (for each step of the episode)

Choose *a* from *s* using a policy

Take action *a*, observe *r, s'*

*Update Q and s….*

Random policy; Exploration

Greedy policy; Exploitation

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

$\epsilon$-greedy policy: With probability ϵ select a random action

# Deep RL: Deep Learning + RL

- Traditional RL
  - low-dimensional state spaces
  - handcrafted features

- DL's representation power + RL's generalization ability
  - RL defines the objective
  - Deep Learning learns the representation

# Deep Q-Learning

# Deep Q-Learning


states

actions

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Noop | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Fire | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Right | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Left | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

#states = $256^{210 \times 160}$

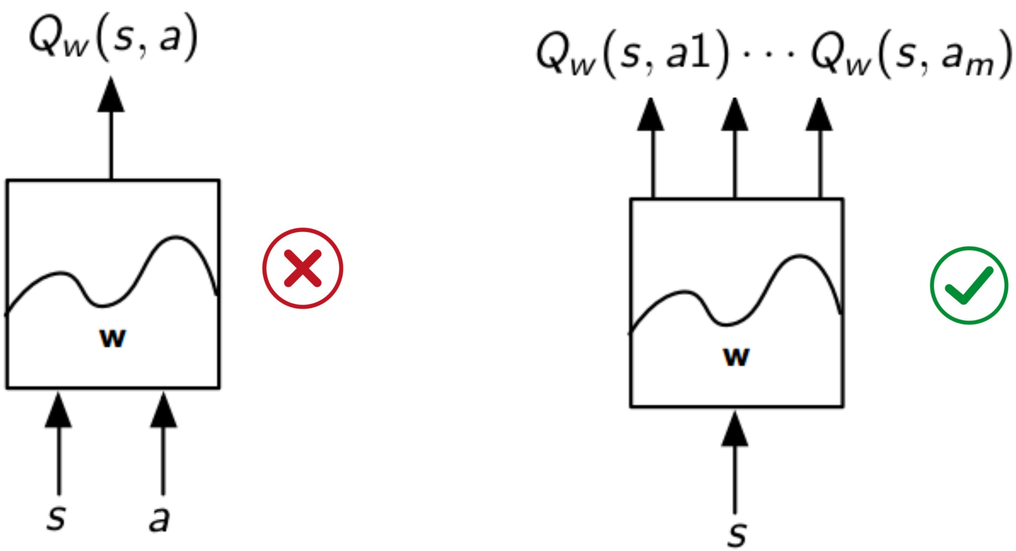
Value Function Approximation

$$Q(s, a) = f(s, a, w)$$

# Q-Networks

- Represent the state-action value function （discrete actions） by Q-network with weights $w$

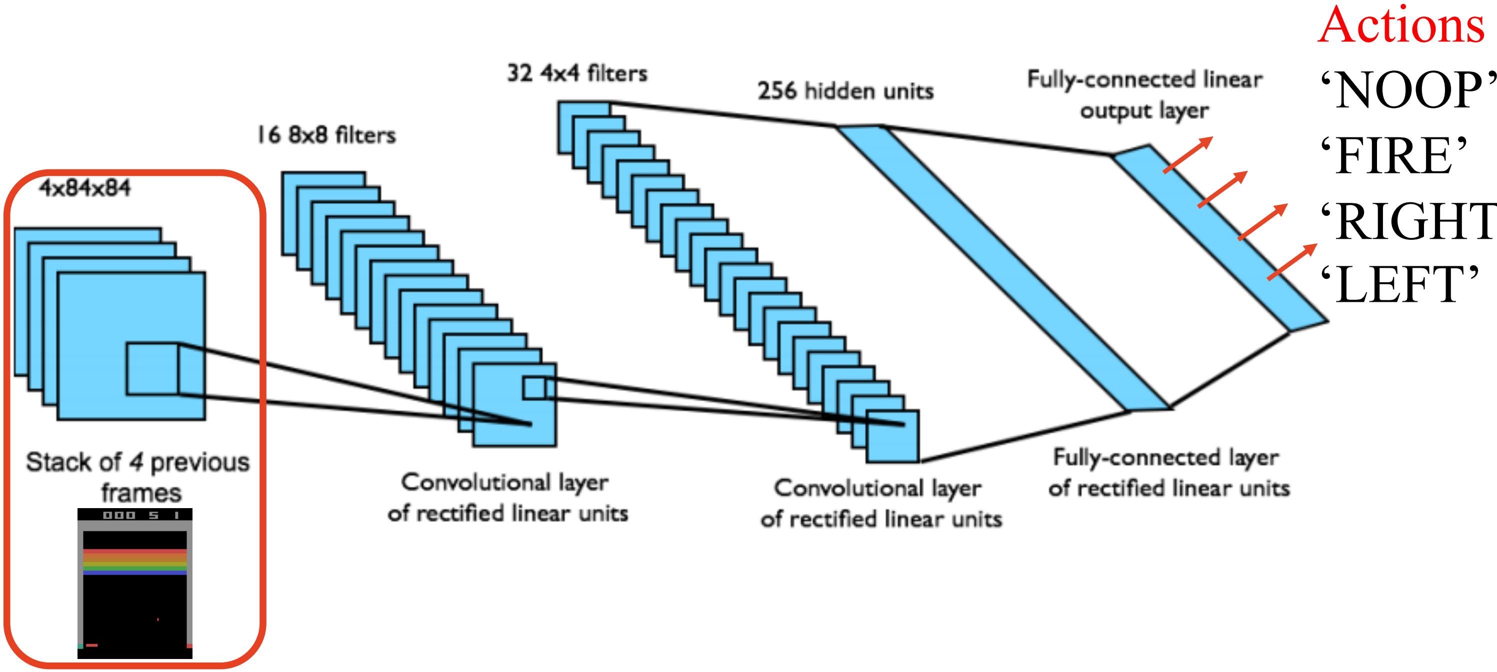
$$Q_w(s, a) \approx Q^*(s, a)$$




From the *Tutorial: Deep Reinforcement Learning* by David Silver, Google DeepMind

# Deep Q-Learning

- End-to-end learning of state-action values from raw pixels
- Input state is stack of raw pixels from last 4 frames
- Output are state-action values from all possible actions




From the *Tutorial: Deep Reinforcement Learning* by David Silver, Google DeepMind

# Deep Q-Networks (DQN)

- Optimal Q-values obey Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\left\{\underline{r + \gamma \max_{a'} Q(s',a')}\,|\, s,a\right\}$$

- Treat right-hand size as a target and minimize MSE loss by SGD

$$l = \left(\underbrace{r + \gamma \max_{a'} Q_w(s',a')}_{\text{Target}} - Q_w(s,a)\right)^2$$

- Divergence issues using neural networks due to
  - Correlations between samples
  - Non-stationary targets

From the *Tutorial: Deep Reinforcement Learning* by David Silver, Google DeepMind

# Experience replay

- Build data set from agent's own experience
- Sample experiences uniformly from data set to remove correlations

| |
|---|
| $s_1, a_1, r_2, s_2$ |
| $s_2, a_2, r_3, s_3$ |
| $s_3, a_3, r_4, s_4$ |
| $\ldots$ |
| $s_t, a_t, r_{t+1}, s_{t+1}$ |

$\rightarrow \quad s, a, r, s'$

$s_t, a_t, r_{t+1}, s_{t+1} \quad \rightarrow$

# Algorithm

**Algorithm 1** Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
**for** episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  **for** $t = 1, T$ **do**
    With probability $\epsilon$ select a random action $a_t$
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to
  **end for**
**end for**

$\epsilon$-greedy

From the *Tutorial: Deep Reinforcement Learning* by David Silver, Google DeepMind

# Improvements: Target Network

- To deal with non-stationarity, target parameters $\hat{w}$ are held fixed

$$l = \left( r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a) \right)^2$$

$$l = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left\{ \left( r + \gamma \max_{a'} Q_{\hat{w}}(s', a') - Q_w(s, a) \right)^2 \right\}$$

# Improvements: Double DQN

- Q-learning is known to overestimate state-action values
  - The max operator uses the same values to select and evaluate an action

$$Q^*(s,a) = \mathbb{E}_{s'}\left\{ r + \gamma \max_{a'} Q(s',a') | s,a \right\}$$

- The upward bias can be removed by decoupling the selection from the evaluation
  - Current Q-network is used to select actions
  - Older Q-network is used to evaluate actions

$$l = \mathbb{E}_{(s,a,r,s')\sim U(D)} \left\{ \left( r + \gamma Q_{\hat{w}_i}(s', \arg\max_{a'} Q_{w_i}(s',a')) - Q_{w_i}(s,a) \right)^2 \right\}$$

From the talk *Introduction to Deep Reinforcement Learning From Theory to Applications* by Siyi Li (HKUST)

# Improvements: Prioritized Replay

- Uniform experience replay samples transitions regardless of their significance
- Can weight experiences according to their significance
- Prioritized replay stores experiences in a priority queue according to the TD error

- Use stochastic sampling to increase sample diversity

$$\left| r + \gamma \max_{a'} Q_{\hat{w}}(s', a') - Q_w(s, a) \right|$$

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

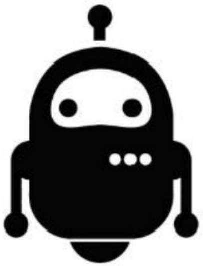
$$p_i = |\delta_i| + \epsilon$$

From the talk *Introduction to Deep Reinforcement Learning From Theory to Applications* by Siyi Li (HKUST)

# Imitation Learning

# Imitation Learning

## Reinforcement Learning



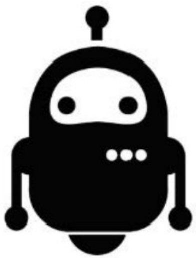
**Agent**

$\pi(a|s)$

**Delayed reward**

$r \sim R(s, a)$

## Imitation Learning



**Agent**

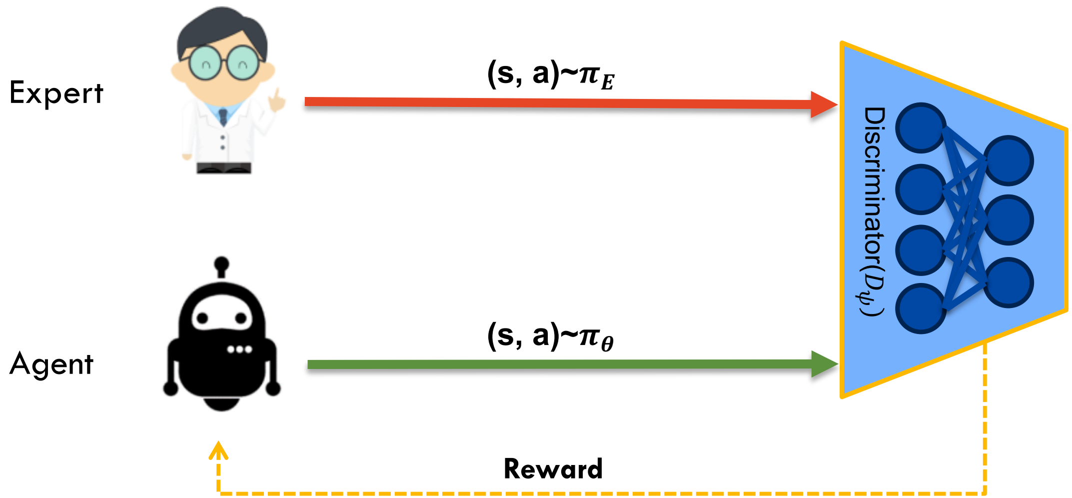
$\pi(a|s)$

**Expert dataset**

$(s, a) \sim \pi_{\mathrm{E}}$

Imitation learning aims to let the agent mimic the behavior of the expert, without any reward signal.

# Generative Adversarial Imitation Learning




$$\min_{\theta} \max_{\psi} \mathbb{E}_{(s,a)\sim\pi_\theta}\left[\log D_\psi(s,a)\right] + \mathbb{E}_{(s,a)\sim\pi_E}\left[\log(1 - D_\psi(s,a))\right]$$

# Imperfect Demonstrations Issues

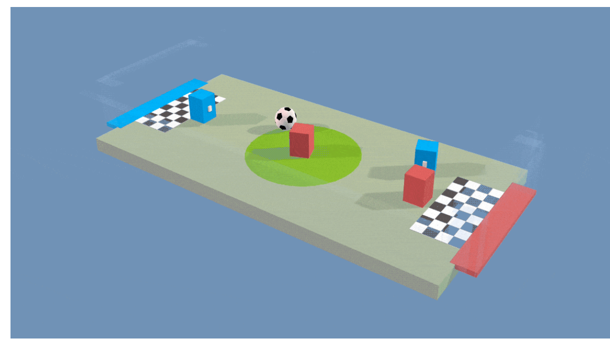
**Optimal demonstrations** —GAIL→ **Optimal agent's policy**

**Imperfect demonstrations** —GAIL→ **Non-optimal agent's policy**

However in many real-world tasks, it is hard to collect plenty of optimal demonstrations.

**Imperfect**

**Demonstrations**



**Expert** | **Agent**

# Proposed Method: Weighted GAIL

[ICML 2021] Learning to Weight Imperfect Demonstrations

$$\min_{\theta} \max_{\psi} \mathbb{E}_{(s,a)\sim\pi_\theta}\left[\log D_\psi(s,a)\right] + \mathbb{E}_{(s,a)\sim\pi_E}\left[w(s,a)\log(1 - D_\psi(s,a))\right]$$

How to formulate a proper weight without auxilary information?

# Proposed Method

## Learn from A New Policy

$$\min_{\theta} \max_{\psi} \mathbb{E}_{(s,a)\sim\rho_{\pi_\theta}}\left[\log D_\psi(s,a)\right] + \mathbb{E}_{(s,a)\sim\rho_{\pi_E}}\left[w(s,a)\log(1 - D_\psi(s,a))\right]$$

$$\Big\Downarrow \text{ Variational Inference}$$

$$\min_{\theta} Div_{JS}(\rho_{\pi_\theta}, \rho_{\tilde{\pi}})$$

The objective function of weighted GAIL is equal to imitate a new policy $\tilde{\pi}$, where $\rho_{\tilde{\pi}}(s,a) = w(s,a)\rho_{\pi_E}(s,a)$.

# Proposed Method

## Analysis on the Weight

**Theorem 1.** *Consider an f-divergence constrained policy optimization problem as,*

$$\max_{\tilde{\pi}} \; L^{d_\pi}(\tilde{\pi}) - \beta D_f(\rho_{\tilde{\pi}} || \rho_\pi)$$

$$\text{s.t.} \sum_a \tilde{\pi}(a|s) = 1, \tilde{\pi}(a|s) \in [0,1]$$

where $\beta$ is a hyper-parameter to balance the influence of there two terms. We have,

$$\tilde{\pi}(a|s) = \pi(a|s) f_*'\left(\frac{A_\pi(s,a) + C(s)}{\beta}\right)$$

---

Consider KL divergence, we have $\tilde{\pi}(a|s) = \pi(a|s)\exp\left(\frac{A_\pi(s,a)+C(s)}{\beta}\right)$

# Proposed Method

## Analysis on the Weight

**Theorem 2.** *Given two policies $\tilde{\pi}$ and $\pi$ which satisfies*

$$\tilde{\pi}(a|s) = \pi(a|s)\exp((1/\beta)(A_\pi(s,a) + C(s)))$$

where $\beta$ is a hyper-parameter and $C(s)$ is a a function of state $s$. We can conclude that $\tilde{\pi}$ is generally better than $\pi$, that is,

$$V_{\tilde{\pi}}(\mathrm{s}) > V_\pi(\mathrm{s}), \forall\ \mathrm{s} \in S$$

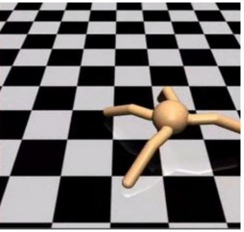
Theorem 2 suggests that new policy is indeed better, when $\mathrm{w(s,a)} = \exp(\frac{A_\pi(s,a)}{\beta})$.

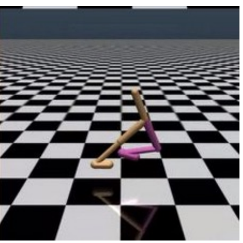
$\boldsymbol{A_\pi(s,a) = \log w(s,a)^\beta}$

$$\mathrm{w(s,a)} = [(1/D_\psi^*(s,a) - 1)\pi_\theta(a|s)]^{\frac{1}{\beta+1}}$$

# Experiment

## Mujoco





## Quantitive Results

Table 1: Performance of the learned agent, measured by the final x-position of the agent's body. The final x-position of WGAIL and GAIL is calculated by the average of the last 50 runs.

| Method | Ant-v2 | | HalfCheetah-v2 | | Walker2d-v2 | | Hopper-v2 | |
|---|---|---|---|---|---|---|---|---|
| | Stage 1 | Stage 2 | Stage 1 | Stage 2 | Stage 1 | Stage 2 | Stage 1 | Stage 2 |
| WGAIL | **111.81** | **182.00** | **120.66** | **190.17** | 5.29 | **18.11** | **12.08** | **14.54** |
| GAIL | 84.50 | 113.58 | 102.21 | 85.33 | 6.66 | 10.80 | 10.06 | 11.81 |
| BC | 91.49 | 135.75 | 99.2 | 118.15 | **7.61** | 12.22 | 0.53 | 0.95 |
| D-REX | 48.63 | 63.43 | 28.73 | 84.57 | 3.10 | 9.61 | 2.59 | 2.17 |
| T-REX | 65.08 | 9.25 | 95.57 | 32.70 | -0.12 | -0.46 | 5.70 | 1.14 |
| 2IWIL | 89.03 | 130.14 | 101.43 | 85.94 | 7.63 | 11.66 | 11.02 | 17.54 |
| Expert (TRPO) | 177.93 | | 195.79 | | 24.71 | | 18.63 | |

## Atari


## Quantitive Results

Table 2: The final result of the learned policy in five Atari tasks.

| | *BeamRider* | *Pong* | *Q*bert* | *Seaquest* | *Hero* |
|---|---|---|---|---|---|
| WGAIL | **1834.8** | **-6.0** | **15140.0** | **649.0** | 20042.0 |
| GAIL | 1541.2 | -6.7 | 13955.0 | 590.9 | **20260.0** |
| BC | 1034.4 | -7.2 | 2720.0 | 598.0 | 7670.0 |
| Expert(PPO) | 2637.45 | 21.0 | 15268.18 | 1840.0 | 27814.09 |

# Thanks!