Deep Reinforcement Learning

Lecture 6 — Deep Q-Network and Its Variants

國立清華大學
NATIONAL TSING HUA UNIVERSITY

National Tsing Hua University
Department of Computer Science

Prof. Chun-Yi Lee

Outline

- **Deep Q-Learning**
- The variants of DQN

Game Playing

Atari-2600

- *Atari-2600, raw image inputs*
- High-dimensional state space
- Embrace the success in deep neural networks (DNN)

(210 × 160 × 3)

Recall: Q-Learning

Fundamentals of Q-learning

- **Q-learning** with function approximator
  - Choose the next action using the greedy policy for the next state
- **Algorithm** :
  1. **Collect data samples** $\{s_t, a_t, r_t, s_{t+1}\}$ by a policy π
  2. **Calculate the update target**: $y_i = r_t + \gamma \max_a Q_\theta(s_{t+1}, a)$
  3. **Update parameters**: $\Delta w = w - \alpha \nabla Q_\theta(s_t, a_t)(y_i - Q_w(s_t, a_t))$

Recall: Q-Learning

Correlated samples

- First step : **Collect data samples** $\{s_t, a_t, r_t, s_{t+1}\}$ by a policy π

$$\{s_t, a_t, r_t, s_{t+1}\} \rightarrow \{s_{t+1}, a_{t+1}, r_{t+1}, s_{t+2}\} \rightarrow \cdots$$

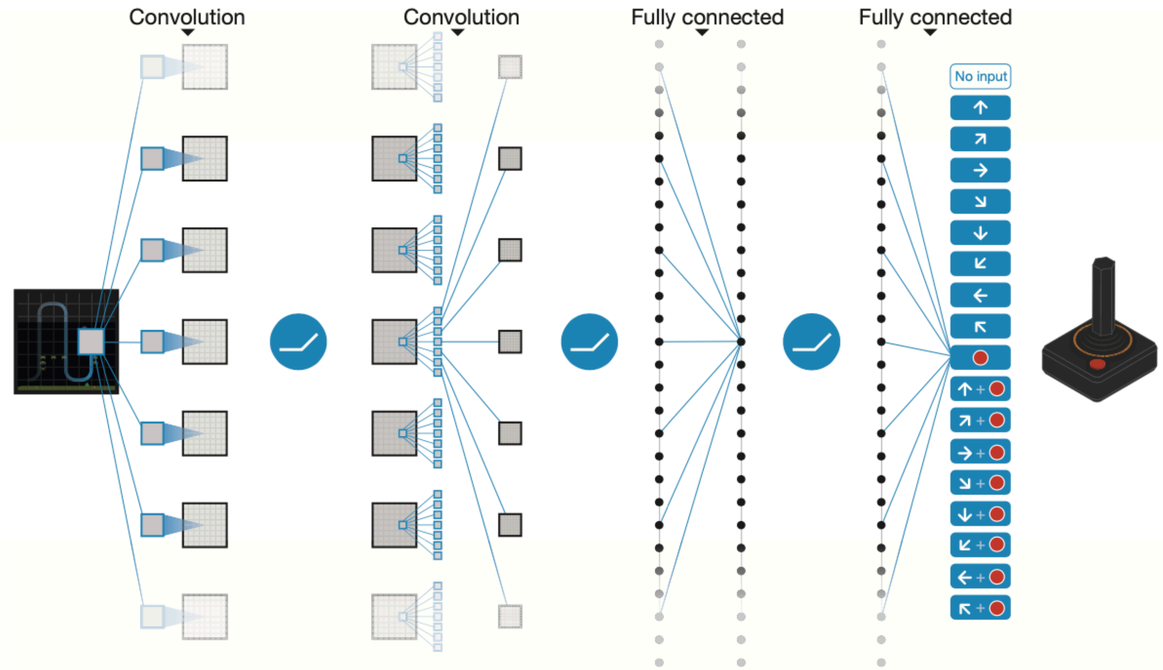
Strongly correlated data

- **Issue: correlated data** is not good for training DNNs

Deep Q-Network (DQN)

The pioneering algorithm using DNNs in RL

- First introduced by **DeepMind** as a paper: “Human-level control through deep reinforcement learning”
  - **Link**



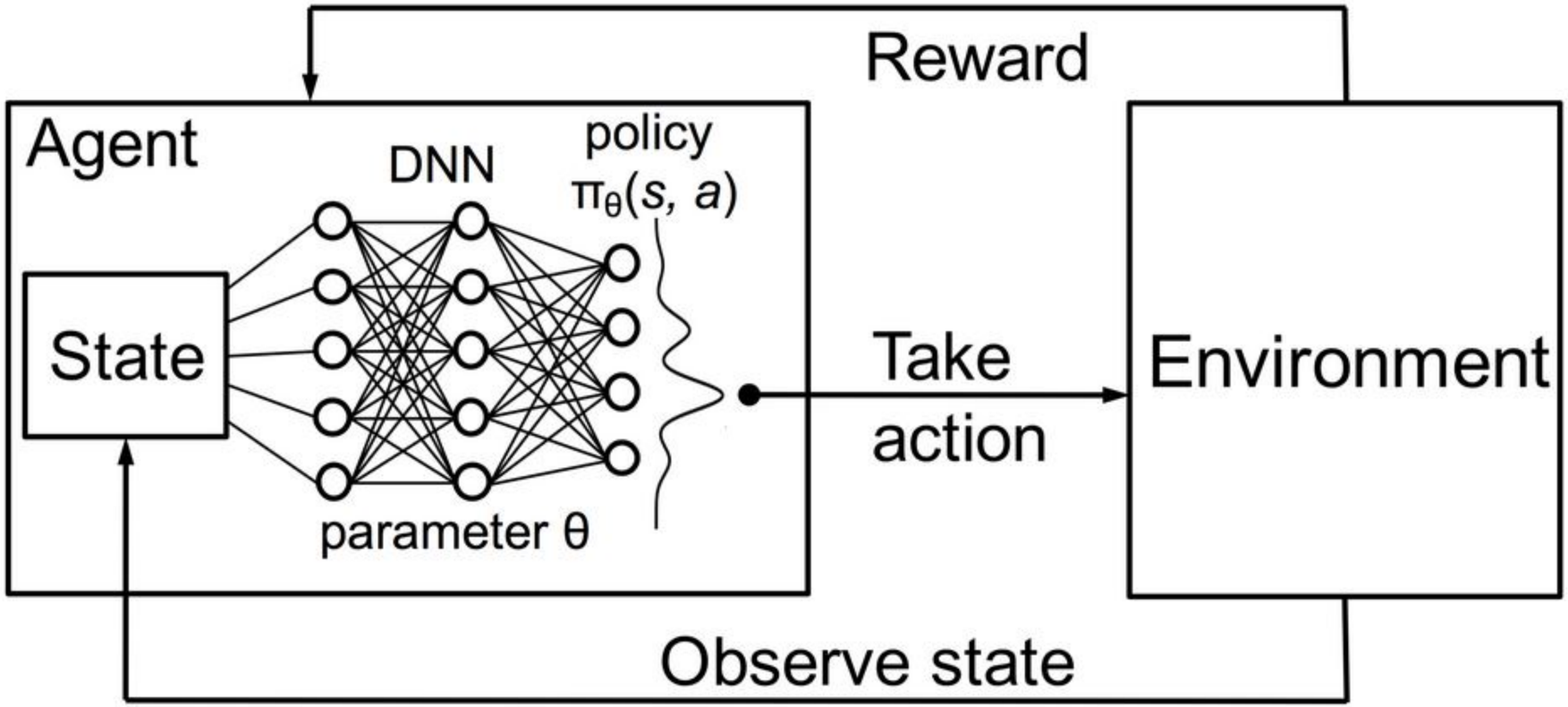
Deep Q-Network (DQN)

Contributions made by DQN

- Parameterize the Q-function with a DNN
- Enhance data-efficiency by a **experience replay buffer**
- Enhance the performance by introducing **two Q-functions**
  - Q_θ is the learning Q-function
  - Q_{θ^-} is the target Q-function
- Modification of the input state representations

Deep Q-Network (DQN)

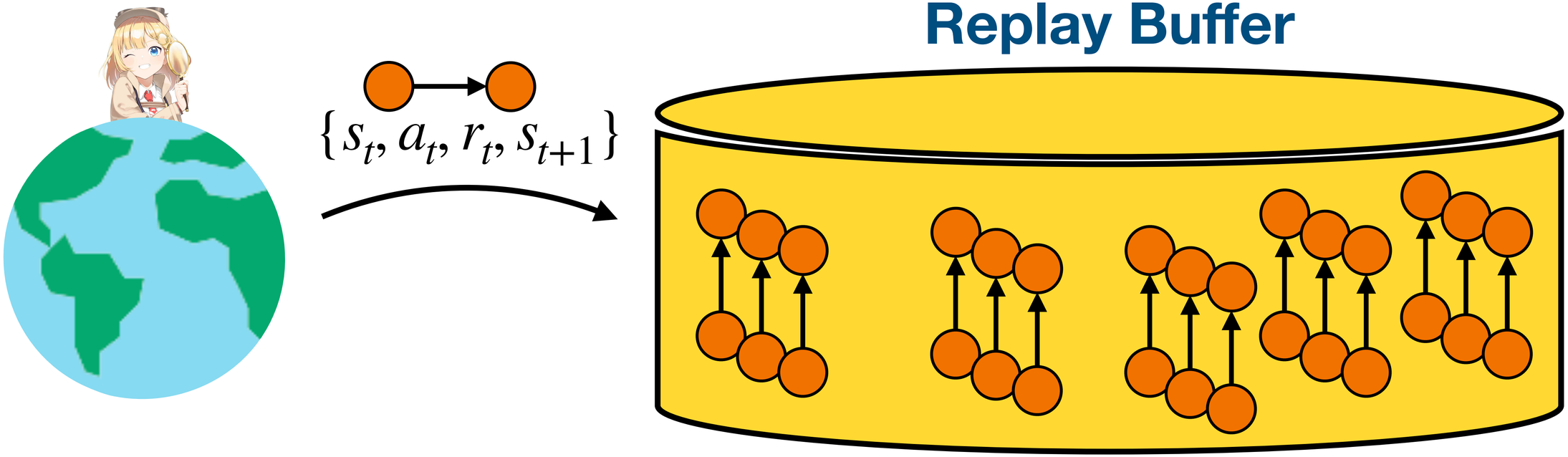
The overall framework of DQN



Deep Q-Network (DQN)

The concept of the experience replay buffer

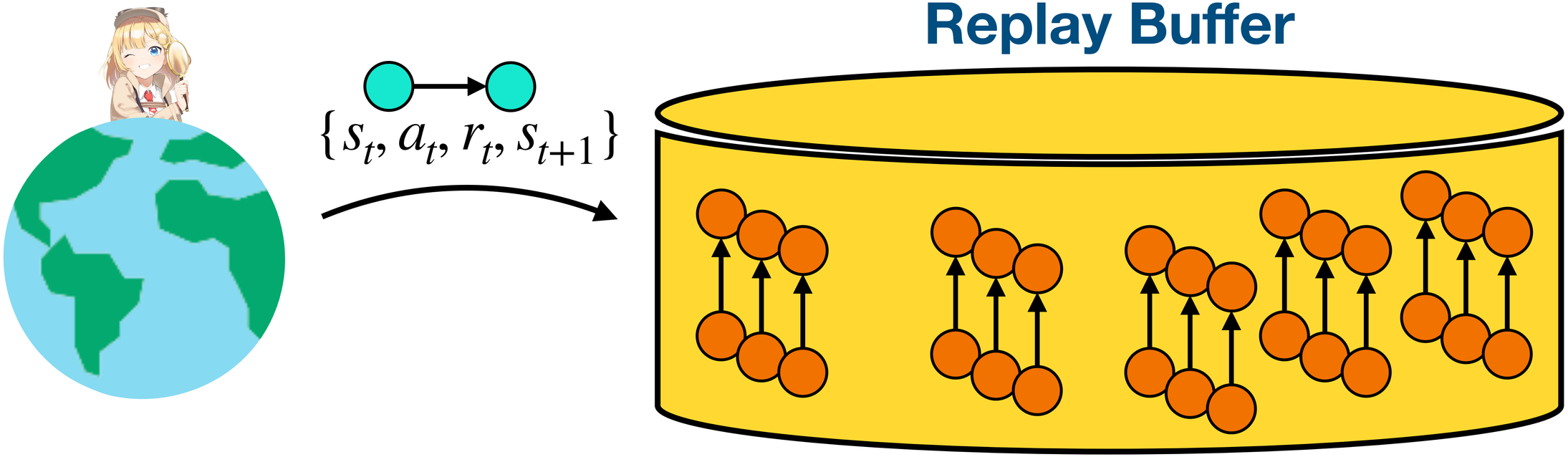
- Regularly storing data into a buffer



Deep Q-Network (DQN)

Update the contents of the experience replay buffer

- Regularly storing data into a buffer



Deep Q-Network (DQN)

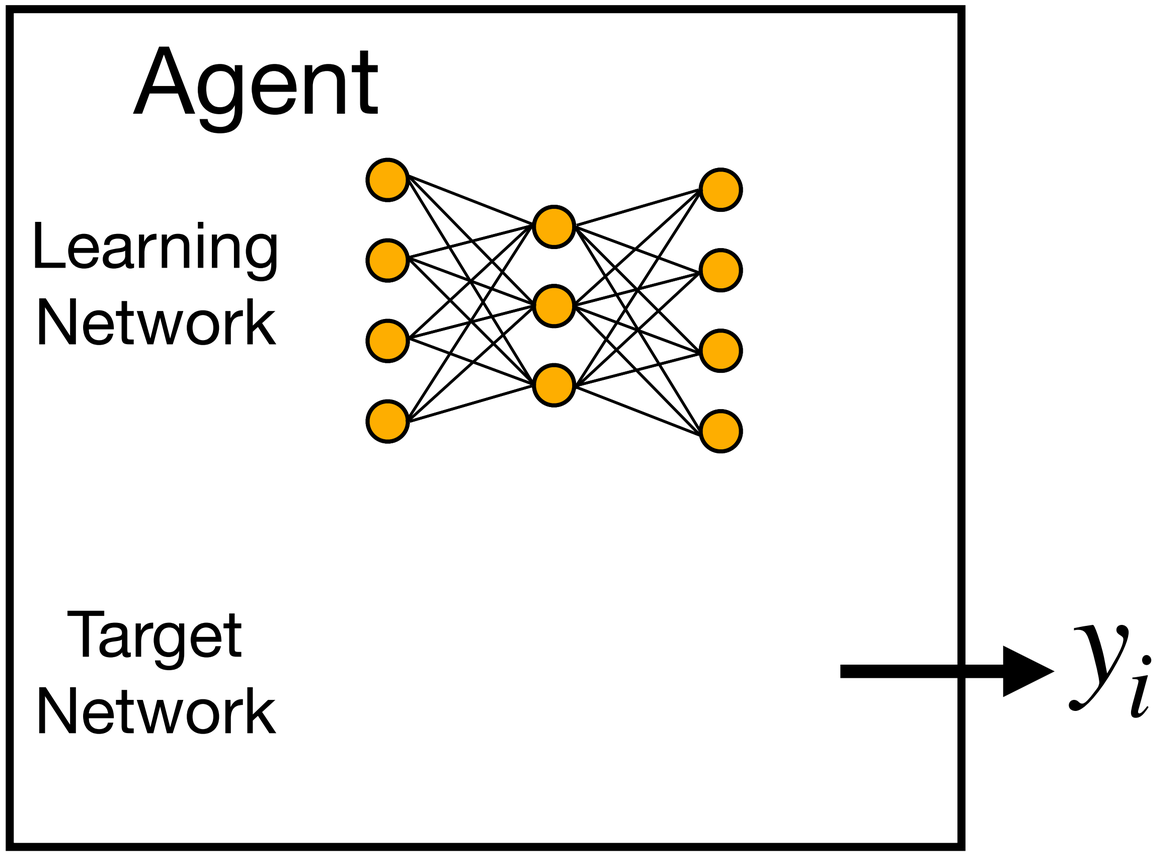
Benefits of the experience replay buffer

* For each timestep, sample multiple datas from the replay buffer
* Reduce the chances of **correlated data** (i.e., the data samples are not sequential)
* Data samples can be used more than one times (high data-efficiency)

Deep Q-Network (DQN)

The concept of target network

- Training with **two Q-functions**



Modify the training target y_i

as $r_t + \gamma \max_a Q_{\theta^-}(s_{t+1}, a)$

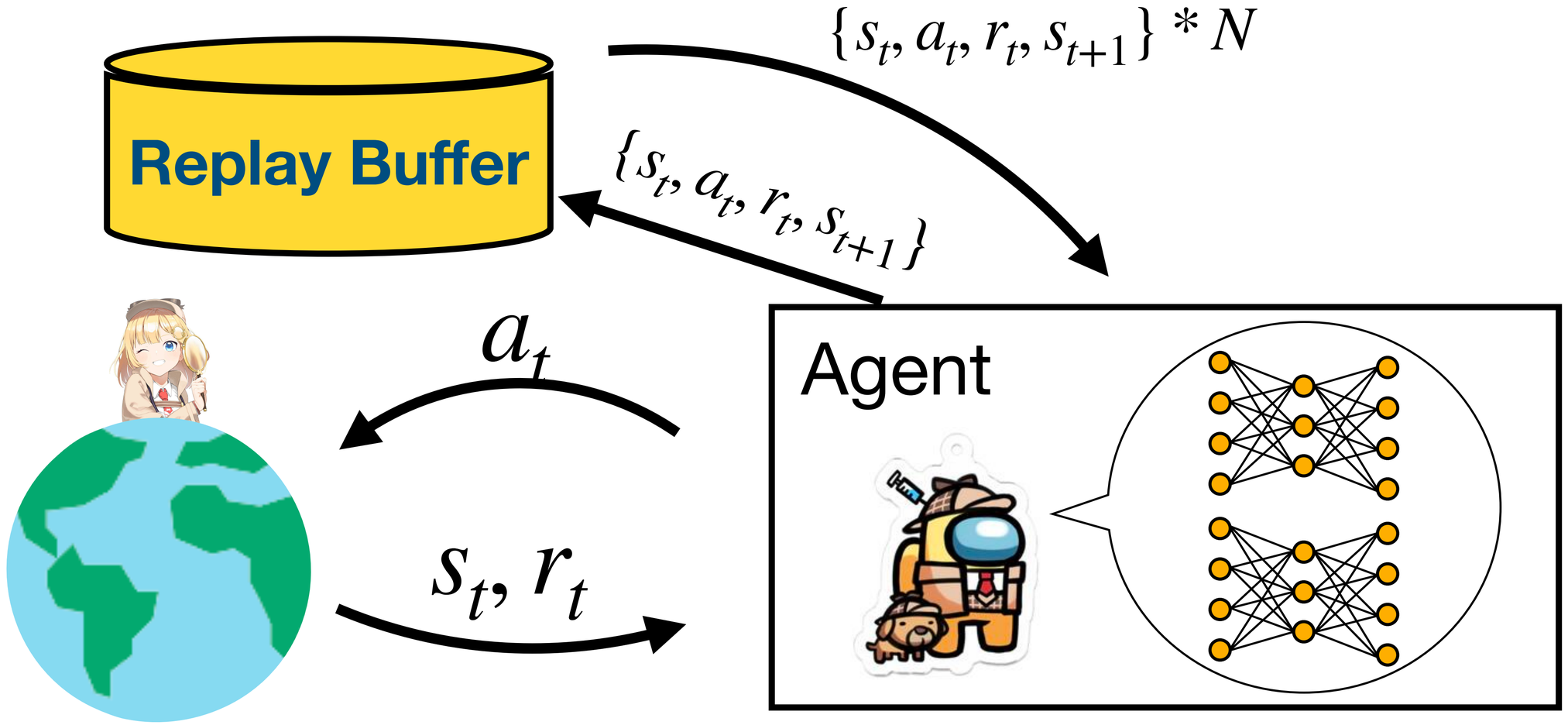
Deep Q-Network (DQN)

Update mechanism of the target network

* **Periodically** update the target network by directly copying the parameters from those of the training network.
    * **Hard approach** : Copy the parameter directly $\theta^- \leftarrow \theta$
    * **Soft approach** : Copy gradually $\theta^- \leftarrow \tau\theta^- + (1-\tau)\theta$
* **Benefit:** Stabilize the training target (target does note always change)
* **Objective:** Stabilize the training process
* **Other enhancements:**
    * Frame skipping
    * Stacked input frames

Deep Q-Network (DQN)

The entire workflow of DQN



Deep Q-Network (DQN)

The pseudo-code of DQN

for ***update the target network*** $\theta^{-} \leftarrow \theta$

for ***store data samples*** $\{s_t, a_t, r_t, s_{t+1}\}$ ***collected by the policy*** π ***into the experience replay buffer*** **Z**

Sample **N** data entries from the experience replay buffer **Z**

Derive the update target and update the parameters

$$y_i = r_t + \gamma \max_a Q_{\theta^-}(s_{t+1}, a)$$

$$\theta = \theta - \alpha \sum_i^N \nabla Q_\theta(s_i, a_i)(y_i - Q_\theta(s_i, a_i))$$

Outline

- Deep Q-Learning
- **The variants of DQN**

DQN Variants

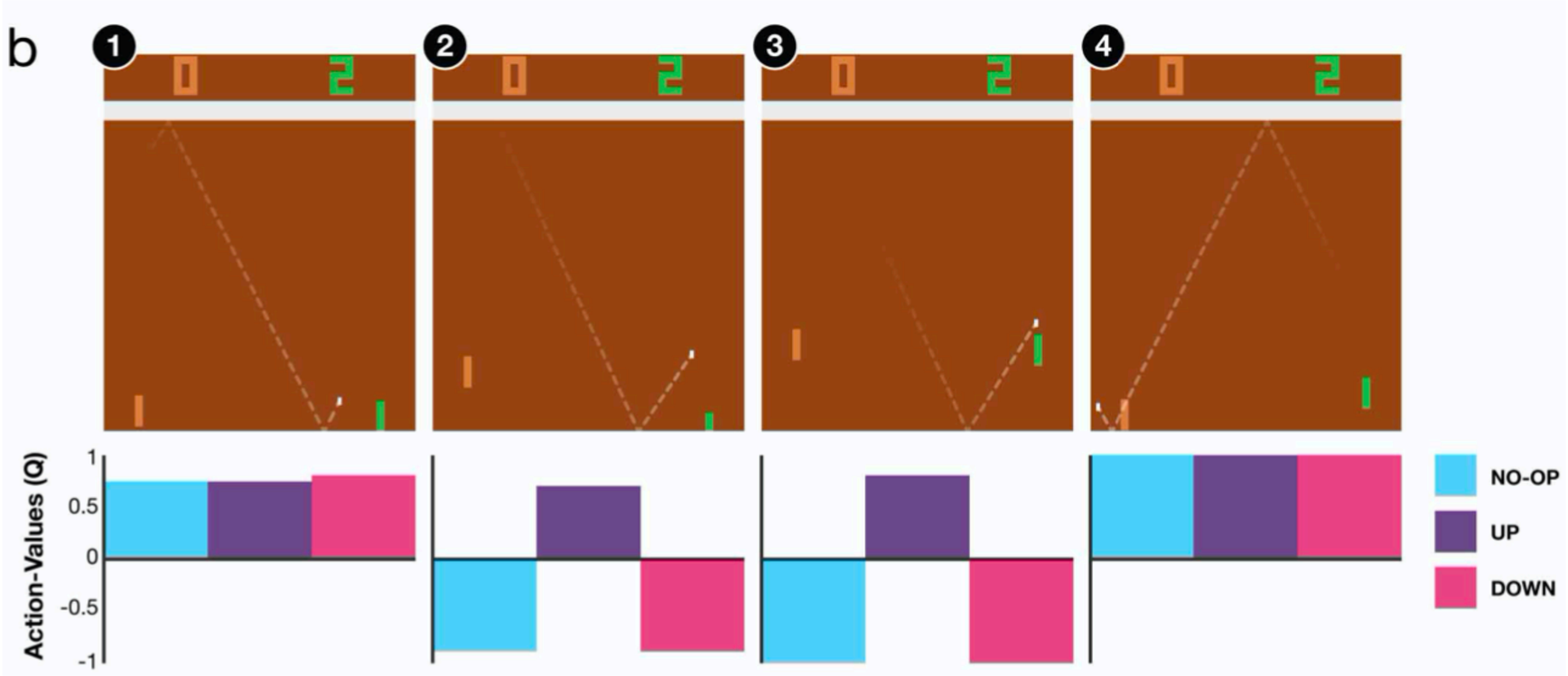
There exists multiple variants of DQN for improving it

- **Double DQN**
- **Dueling DQN**
- **Prioritized Experience Replay for DQN**
- **Deep Recurrent Q-Network (DRQN)**
- **Rainbow DQN**

Double DQN

Improving the DQN algorithm

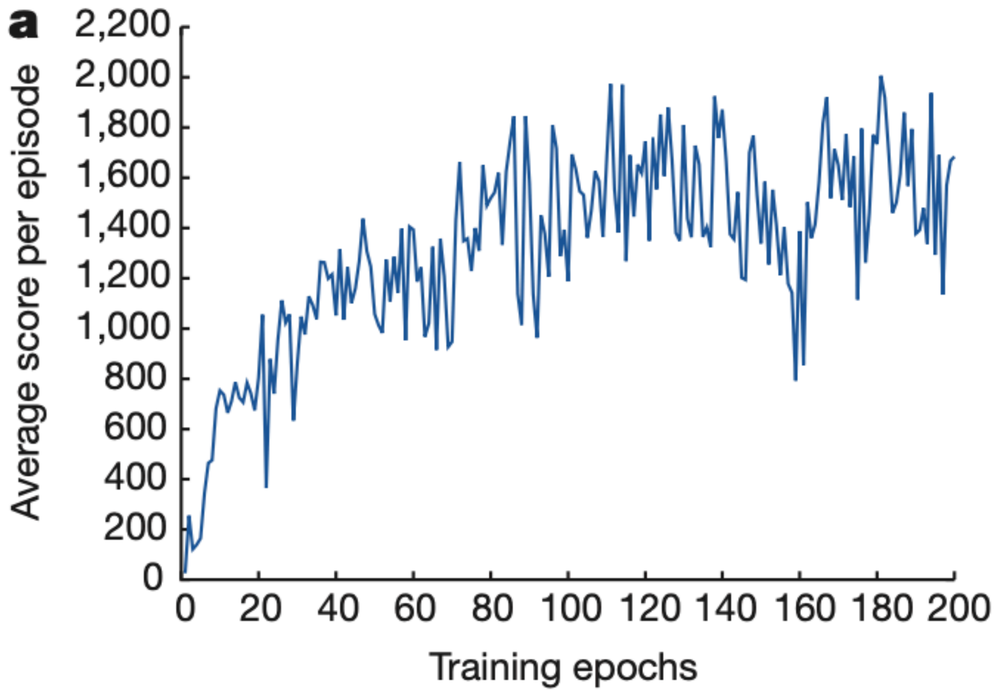
- The Q-function is approximated by DNN

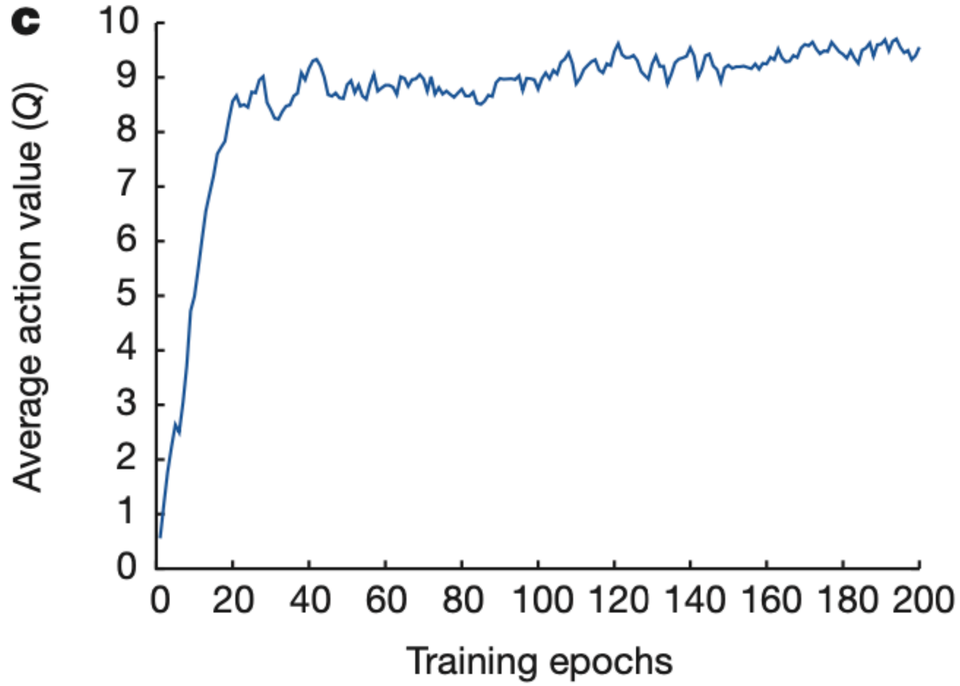


Double DQN

An observation of the estimated Q-value derived by DQN

- As the return become higher, the Q-value become higher

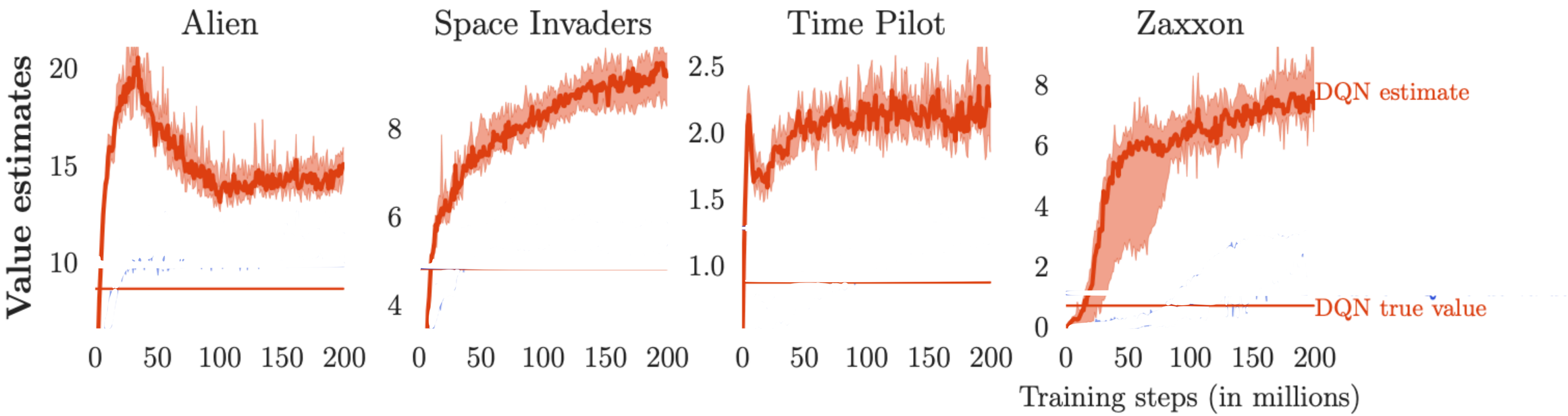




Is the estimated value Q(s,a) really accurate?

Double DQN

Inaccurate estimation of the true values in DQN



- DQN estimate is sometimes higher than the true value

Double DQN

DQN Recap: Decomposition of the Q-function in the update target

- The original equation of the update target is formulated as:
  - $y_i = r_t + \gamma Q_{\theta^-}(s_{t+1}, \arg\max_{a'} Q_{\theta^-}(s_{t+1}, a'))$

$$\downarrow$$

$$\arg\max_{a'} Q_{\theta_A}(s_{t+1}, a')$$

- **Concept:** Avoid the usage of the same Q-function in the update target
- **Objective:** Decompose the update target so that the Q-function for value evaluation and action selection are **different**

Overestimation

The overestimation issue in DQN

- $$y_i = r_t + \gamma \max_a Q_{\theta^-}(s_{t+1}, a)$$
$$= r_t + \gamma Q_{\theta^-}(s_{t+1}, \arg\max_{a'} Q_{\theta^-}(s_{t+1}, a'))$$

- If Q_{θ^-} is not perfect (some noise), the ‘max’ operation would always choose the largest one. Thus, some Q-values would always be updated based on the same actions with increasing trends.
- This condition causes an issue in DQN, called **overestimation**

Double DQN

Original Double Q-learning with NN

- $y_A = r_t + \gamma Q_{\theta_A}(s_{t+1}, \arg\max_{a'} Q_{\theta_B}(s_{t+1}, a'))$
- $y_B = r_t + \gamma Q_{\theta_B}(s_{t+1}, \arg\max_{a'} Q_{\theta_A}(s_{t+1}, a'))$

- **Double Q-learning** uses two networks to update the target values, each network selects actions for the other
- **Double Q-learning** is a conventional way for avoiding the overestimation problem

Double DQN

How Double DQN actually deals with the overestimation issue

* Since **DQN** already contains two networks (the target and learning networks), it does not need to introduce additional networks
  * $y_i = r_t + \gamma Q_{\theta^-}(s_{t+1}, \underset{a'}{\arg\max}\, Q_\theta(s_{t+1}, a'))$

The target network for value estimation

The learning network for action selection

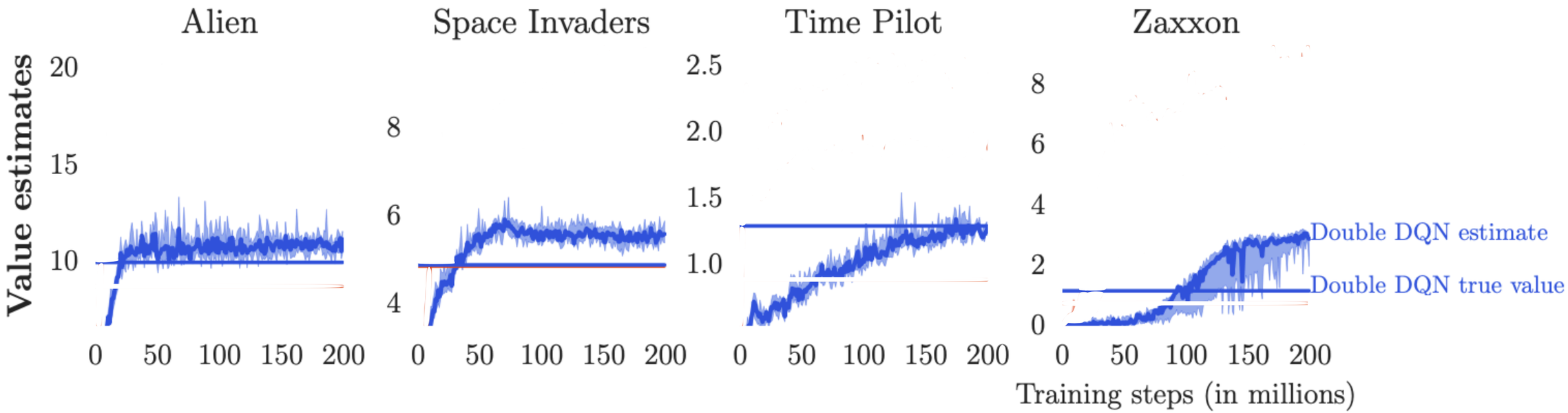
Double DQN

Comparison of DQN and Double DQN

* Modification of the target value as follows:
    * **DQN :** $y_i = r_t + \gamma \max_a Q_{\theta^-}(s_{t+1}, a)$
    * **Double DQN :** $y_i = r_t + \gamma Q_{\theta^-}(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a'))$
* The modification allows the target values to be estimated more accurately, alleviating the impacts of overestimation in DQN
* Double DQN embraces the advantages from both **double Q-learning** and **DQN**

Double DQN

Improving the DQN algorithm



* The experimental results show that the values estimated by double DQN is much closer to true value than those of DQN

DQN Variants

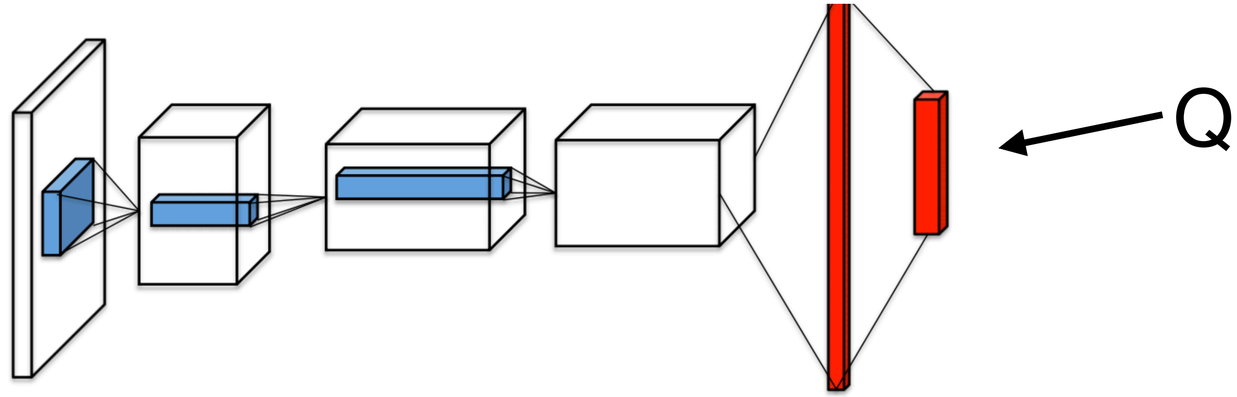
There exists multiple variants of DQN for improving it

- **Double DQN**
- **Dueling DQN**
- **Prioritized Experience Replay for DQN**
- **Deep Recurrent Q-Network (DRQN)**
- **Rainbow DQN**

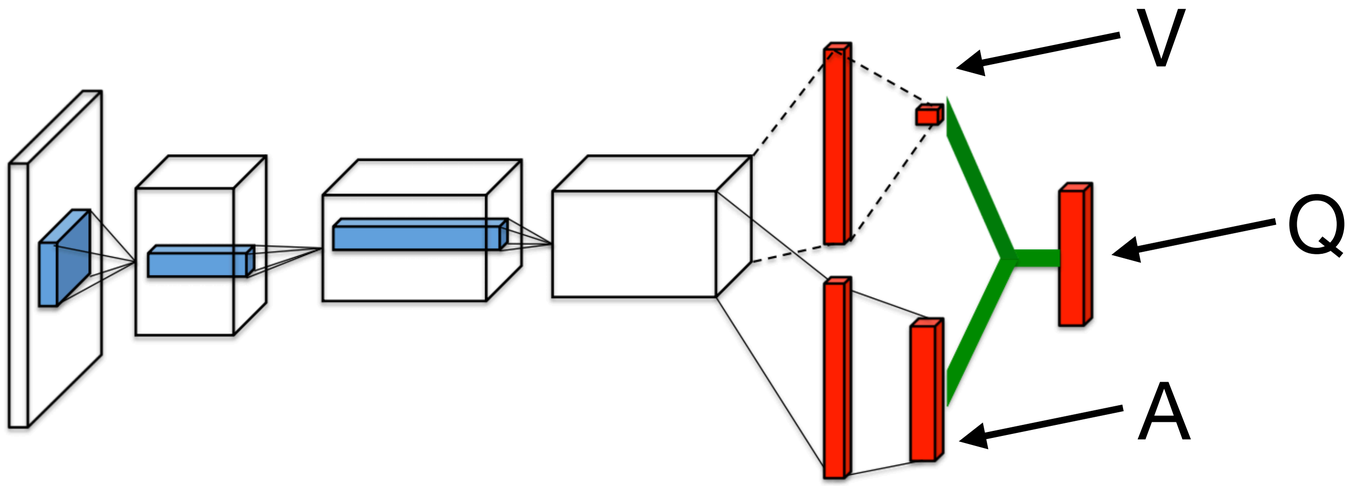
Dueling DQN

Architecture comparison of dueling DQN and the vanilla DQN

- **DQN**



- **Dueling DQN**



Dueling DQN

The concept of the advantage function

- Recall the concept of **Advantage Function**

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$$

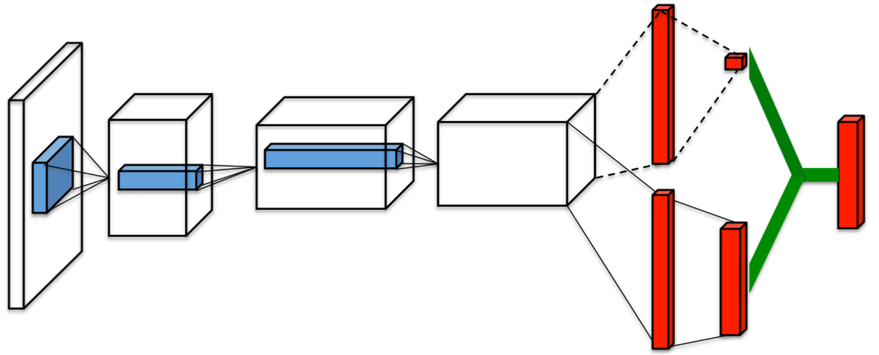
- The **advantage function** describes how much better (in terms of the value) an action is than all of the other possible actions at that state.

Dueling DQN

Decomposition of the Q-function

- **Dueling DQN**

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \rightarrow Q_\pi(s,a) = V_\pi(s) + A_\pi(s,a)$$



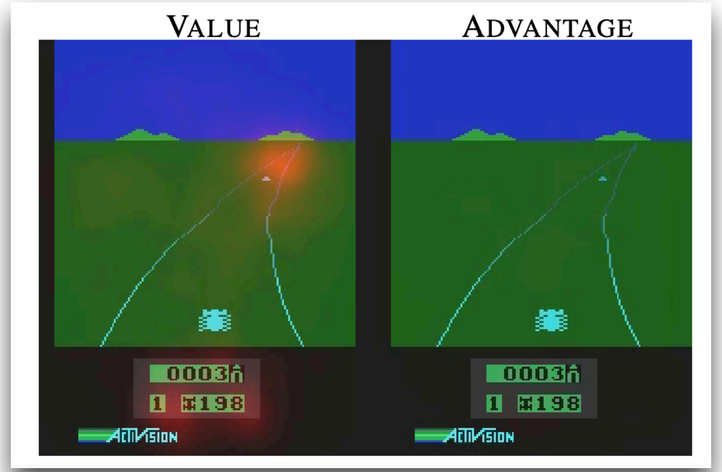
- Dueling DQN decomposes the Q-function estimation into two parts: **value estimation** and **advantage estimation**.
- By decoupling the estimation, intuitively Dueling DQN can learn which states are **more (or less) valuable** **without** having to learn the value of each action at each state (since it's also derivable from $V(s)$).

Dueling DQN

An example from *Enduro* of the Atari 2600 Benchmark

Dueling DQN focuses on **2** things:

- The **scoreboard**
- The **horizon** where new cars may appear



When no car in front of the agent

The **advantage function** focuses on nothing, since no action can apparently bring advantages.

When some cars are in front of the agent

The **advantage function** pays attention to the cars. In this case, action selection is very crucial.

Dueling DQN

Improving the DQN algorithm

- In practice, dueling DQN normalizes the **Advantage Function** as follows

$$Q(s,a) = V(s) + (A(s,a) - \frac{1}{|A|}\sum_{a' \in A} A(s,a'))$$

- It does not change the relative rank of the Advantage function
- This formulation may lose the original semantics. However, it increases the stability of the optimization process (according to the original paper)

DQN Variants

There exists multiple variants of DQN for improving it

- **Double DQN**
- **Dueling DQN**
- **Prioritized Experience Replay for DQN**
- **Deep Recurrent Q-Network (DRQN)**
- **Rainbow DQN**

Prioritized Experience Replay for DQN

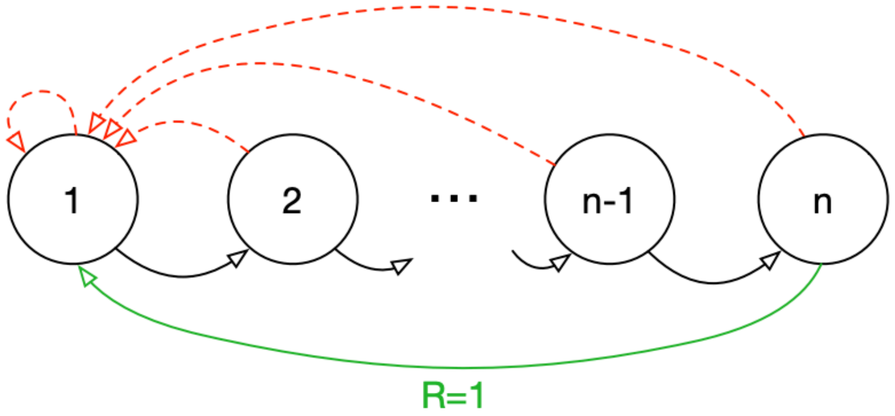
Problems of the experience replay buffer scheme in DQN

- **DQN** randomly samples data from an **Experience Replay Buffer**
  - There exist several potential issues in this scheme
    - It may obtain old data samples
    - Some data samples are not effective
- It needs a better strategy for sampling data
    - However, **which are good data samples**?

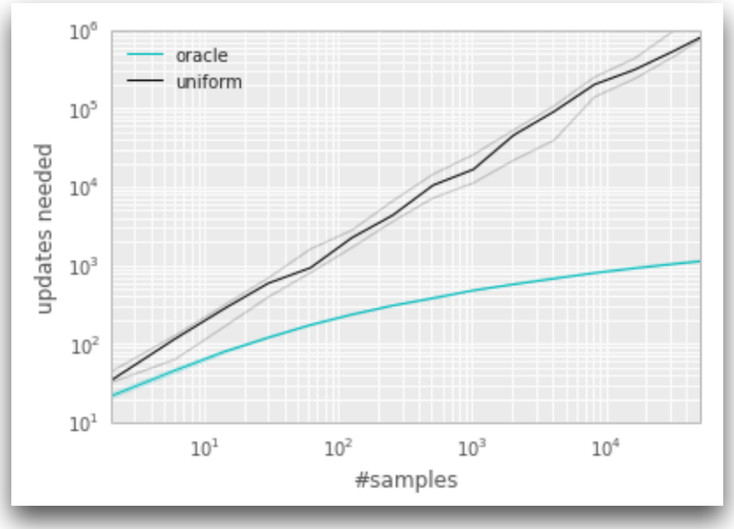
Prioritized Experience Replay for DQN

Why a good data sampling strategy is important

- An example for demonstrating the difference between the uniform sampling strategy and the oracle (exactly knowing the relative importance of the data samples)



A simple sparse reward MDP. The agents can choose to follow the solid or the dashed edges. It only obtains a reward signal R=1 when it moves along the green edge.



For most of the updates, the agent uses samples from the edges other than the green one, making the uniform sampling strategy less effective than the oracle strategy.

Prioritized Experience Replay for DQN

Relative importance of the data samples in the replay buffer

- **Concept**: Prioritizing the data samples in the replay buffer according to their TD-error δ_t:

$$\delta_t = r_t + \gamma \max_a Q_\theta(s_{t+1}, a) - Q_\theta(s_t, a)$$

$$\text{Transition priority}: \quad p_t \leftarrow |\delta_t|$$

- **Methodology**: Higher chances to sample the data in the experience replay buffer with higher values of $|\delta_t|$
    - Higher $|\delta_t|$ indicates that the estimated value is less accurate.

Prioritized Experience Replay for DQN

The data sampling strategy used in a prioritized experience replay

- The data sampling strategy adopted in a **prioritized experience replay buffer** is something between a uniform random strategy and a greedy strategy
  - Each data sample ***i*** in a **prioritized experience replay buffer** is associated with a probability ***P(i)*** of being sampled
  - ***P(i)*** is represented as a non-zero probability for each data sample, formulated as:

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

  - α : a hyper-parameter that defines the extent of prioritization
  - p_i: prioritization of data sample ***i***

Prioritized Experience Replay for DQN

Formulation of prioritization

- There are two approaches for assigning the value of p_i
  - **Proportional prioritization**

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

  - **Rank-based prioritization**

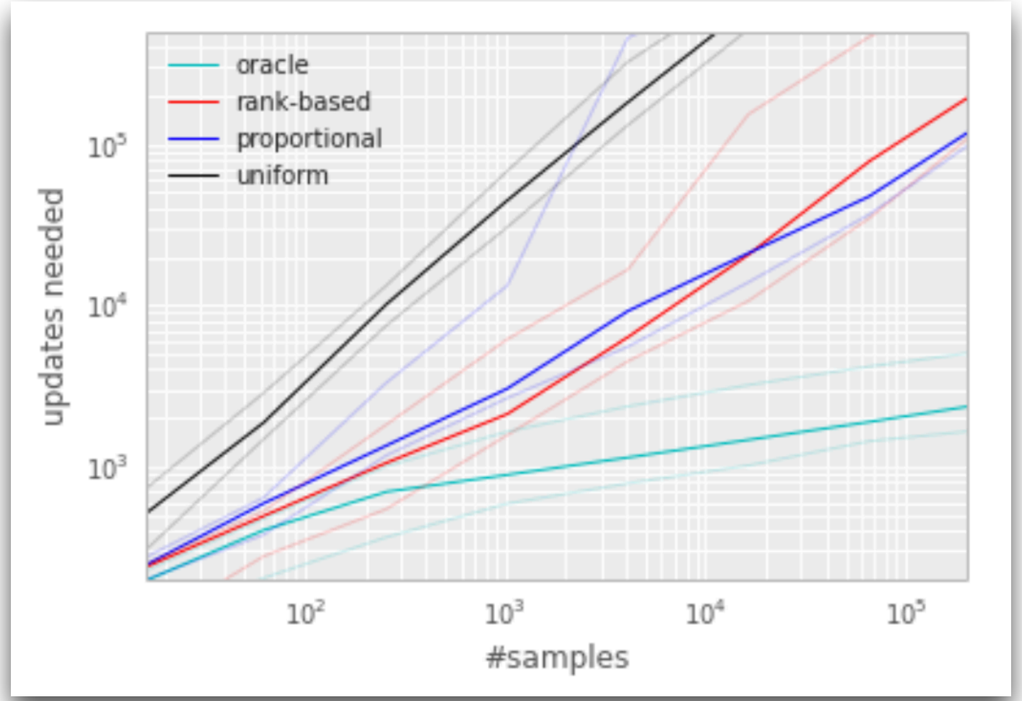
$$p_i = \frac{1}{rank(i)},$$

where $rank(i)$ is the rank of data sample i when the **experience replay buffer** is sorted according to $|\delta_i|$.

Prioritized Experience Replay for DQN

Effectiveness of the prioritized experience replay method

- The experimental results indicates that both approaches are able to enhance the data efficiency (i.e., less number of updates are required)



Prioritized Experience Replay for DQN

Adjustment by importance sampling

- **Prioritized experience replay buffer** introduces bias because it changes the data distribution in an uncontrolled fashion
- Correct this bias by using importance-sampling (IS) weights w_i

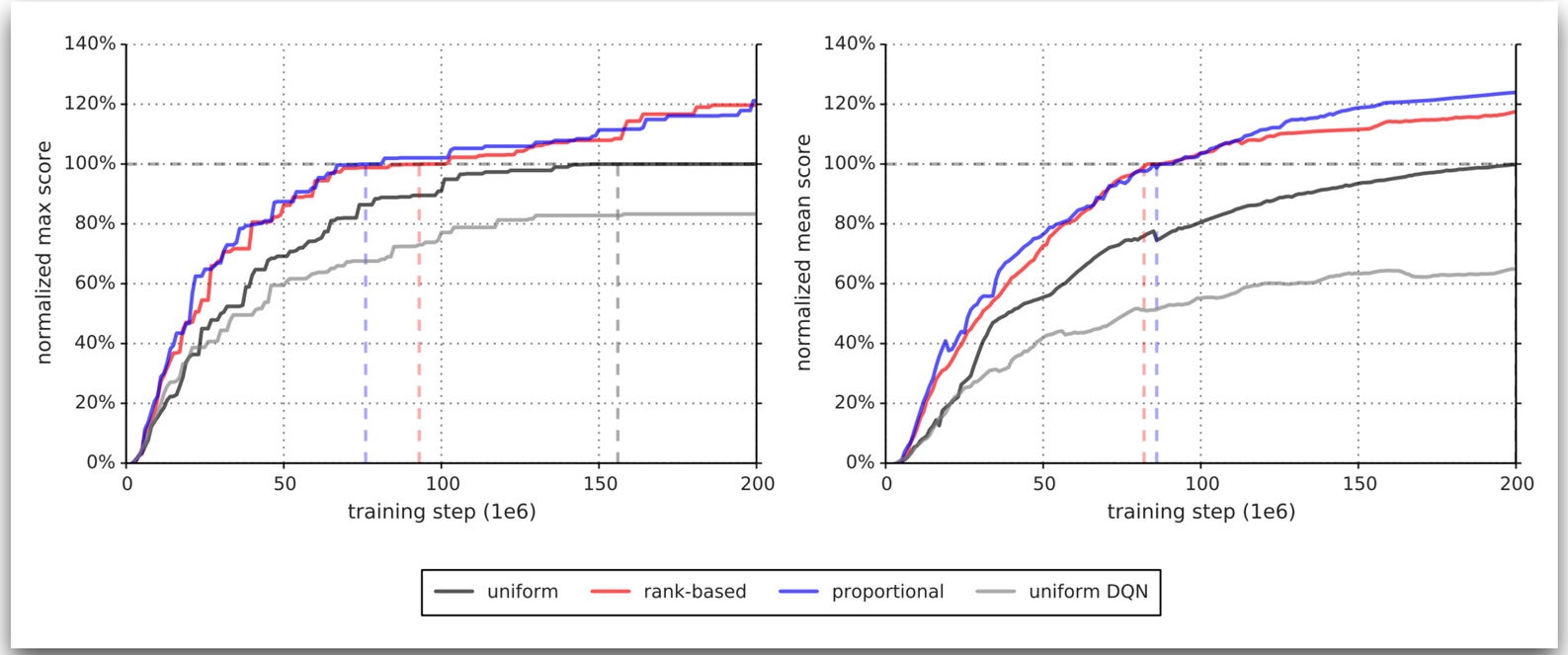
$$w_i = \left(\frac{1}{N} \frac{1}{P(i)} \right)^{\beta},$$

where β is a hyper-parameter, and N is the buffer size

Prioritized Experience Replay for DQN

Evaluation results on the Atari 2600 environments

- The following figures show the averaged evaluation results of ***uniform (double DQN), rank-based (double DQN), proportional (double DQN), and uniform DQN (vanilla DQN)*** on the Atari 2600 environments



DQN Variants

There exists multiple variants of DQN for improving it

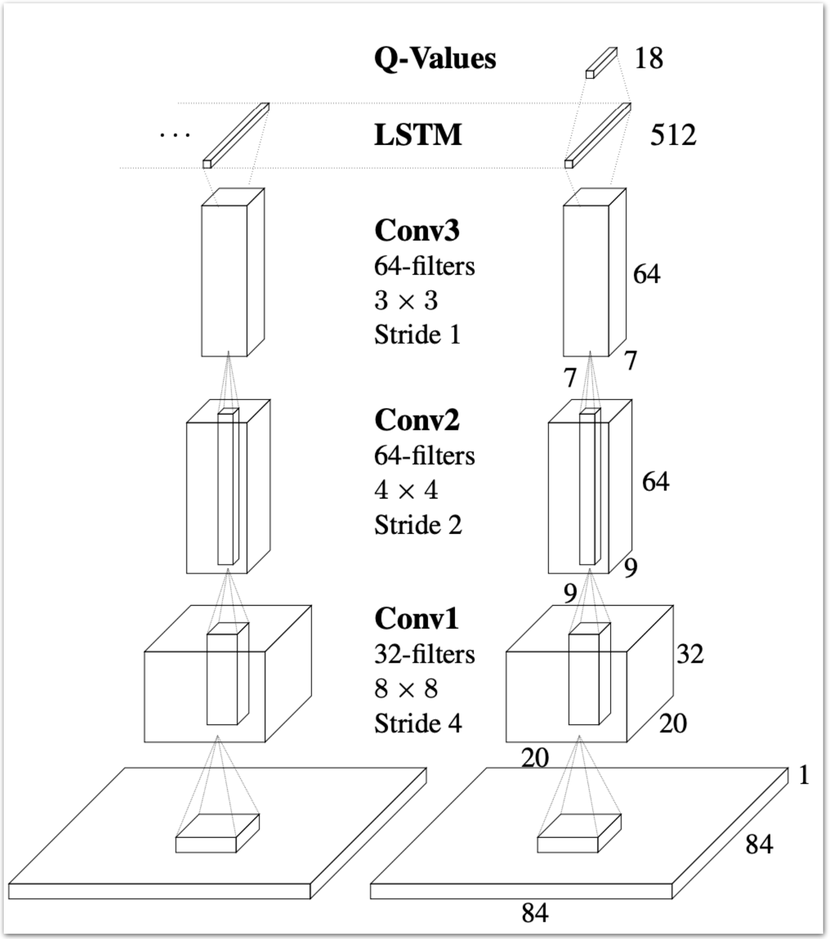
- **Double DQN**
- **Dueling DQN**
- **Prioritized Replay DQN**
- **Deep Recurrent Q-Network (DRQN)**
- **Rainbow DQN**

Deep Recurrent Q-Network (DRQN)

An approach for dealing with POMDP

- **DRQN** is proposed to deal with POMDP
  - **DRQN** modifies the last fully-connected (FC) layer of DQN by LSTM
  - Instead of taking four stacked input frames as DQN, **DRQN** only takes a single input frame

Input size : 84x84x1



Deep Recurrent Q-Network (DRQN)

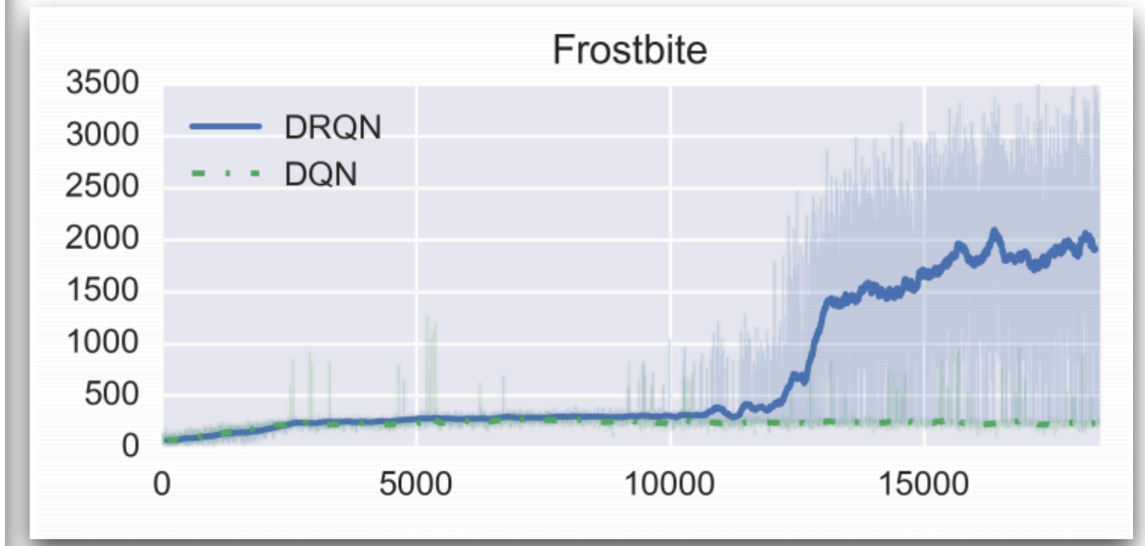
Data sampling strategies of DRQN

- Since LSTM needs sequential data to update, DRQN requires a different data sampling strategy. Two strategies can be selected:
    - **Bootstrapped Sequential Updates**
        - Randomly selects the episodes from the replay buffer
    - **Bootstrapped Random Updates**
        - Randomly selects the episodes from the replay buffer, and begins at random points

Deep Recurrent Q-Network (DRQN)

Experimental results: A comparison of DRQN and DQN for Atari

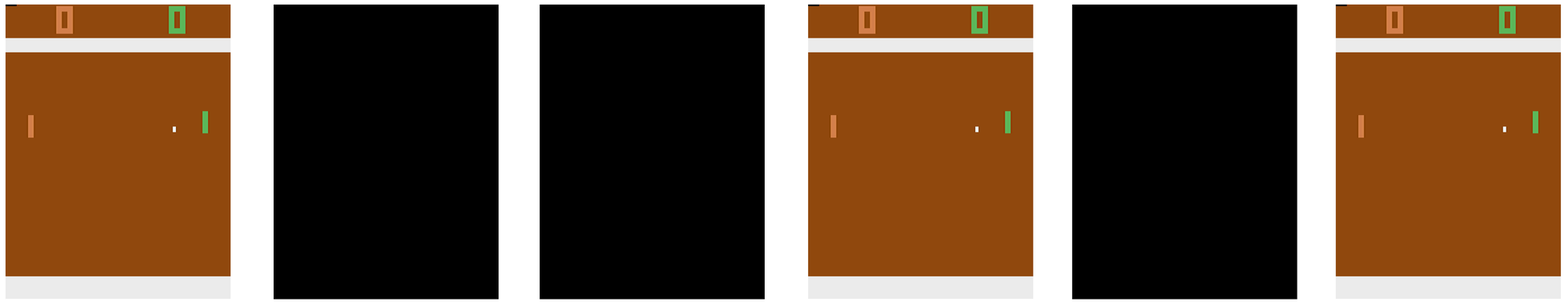
	DRQN $\pm std$	DQN $\pm std$	
Game		Ours	Mnih et al.
Asteroids	1020 ($\pm$312)	1070 ($\pm$345)	1629 ($\pm$542)
Beam Rider	3269 ($\pm$1167)	**6923** ($\pm$1027)	6846 ($\pm$1619)
Bowling	62 ($\pm$5.9)	72 ($\pm$11)	42 ($\pm$88)
Centipede	3534 ($\pm$1601)	3653 ($\pm$1903)	8309 ($\pm$5237)
Chopper Cmd	2070 ($\pm$875)	1460 ($\pm$976)	6687 ($\pm$2916)
Double Dunk	**-2** ($\pm$7.8)	-10 ($\pm$3.5)	-18.1 ($\pm$2.6)
Frostbite	**2875** ($\pm$535)	519 ($\pm$363)	328.3 ($\pm$250.5)
Ice Hockey	-4.4 ($\pm$1.6)	-3.5 ($\pm$3.5)	-1.6 ($\pm$2.5)
Ms. Pacman	2048 ($\pm$653)	2363 ($\pm$735)	2311 ($\pm$525)



Deep Recurrent Q-Network (DRQN)

An example of a POMDP environment: Flickering environments

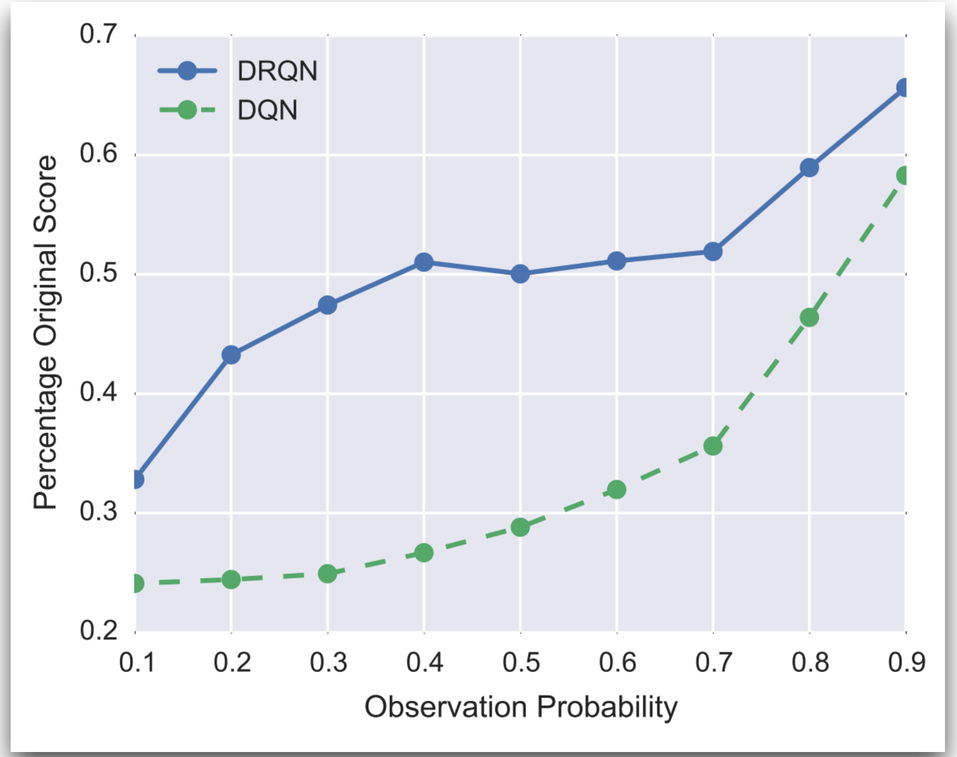
- A flickering environment is a typical POMDP problem
  - The input frames are flickering, meaning that they are not available for every time steps
- E.g., *Flickering Pong* — fully revealed or fully obscured with probability ***p***



Deep Recurrent Q-Network (DRQN)

Evaluation results on the flickering environments

- DRQN performs better than DQN in **SOME** environments



Flickering	DRQN $\pm std$	DQN $\pm std$
Asteroids	1032 ($\pm$410)	1010 ($\pm$535)
Beam Rider	618 ($\pm$115)	**1685.6** ($\pm$875)
Bowling	65.5 ($\pm$13)	57.3 ($\pm$8)
Centipede	4319.2 ($\pm$4378)	5268.1 ($\pm$2052)
Chopper Cmd	1330 ($\pm$294)	1450 ($\pm$787.8)
Double Dunk	-14 ($\pm$2.5)	-16.2 ($\pm$2.6)
Frostbite	414 ($\pm$494)	436 ($\pm$462.5)
Ice Hockey	-5.4 ($\pm$2.7)	-4.2 ($\pm$1.5)
Ms. Pacman	1739 ($\pm$942)	1824 ($\pm$490)
Pong	**12.1** ($\pm$2.2)	-9.9 ($\pm$3.3)

DQN Variants

There exists multiple variants of DQN for improving it

- **Double DQN**
- **Dueling DQN**
- **Prioritized Replay DQN**
- **Deep Recurrent Q-Network (DRQN)**
- **Rainbow DQN**

Rainbow-DQN

A hybrid technique that leverages the benefits from all DQN variants

---

- **Ranbow-DQN** combines different variants of DQN, including
    - Double **Q-learning**
    - Prioritized experience replay
    - Dueling networks
    - Multi-step return
    - Distributional RL (future topic)
    - Noisy nets (further topic)

Rainbow-DQN

The concept of C51: A distributional RL technique

- **Distributional RL based on C51**
  - Modify V and Q to distributions with ***N*** categories:

$$V(s), Q(s,a) \rightarrow Z(s), Q(s,a)$$

  - Parametric Distribution : set $[N, V_{max}, V_{min}]$,

$$z_i = V_{min} + (i-1)\frac{V_{max} - V_{min}}{N-1} : 0 \leq i \leq N$$

$$Z_{\theta_i}(s,a) = z_i, \qquad p_i(s,a) = \frac{e^{\theta_i(s,a)}}{\sum_j e^{\theta_j(s,a)}}$$

Rainbow-DQN

The concept of C51: A distributional RL technique

- **Distributional RL based on C51**

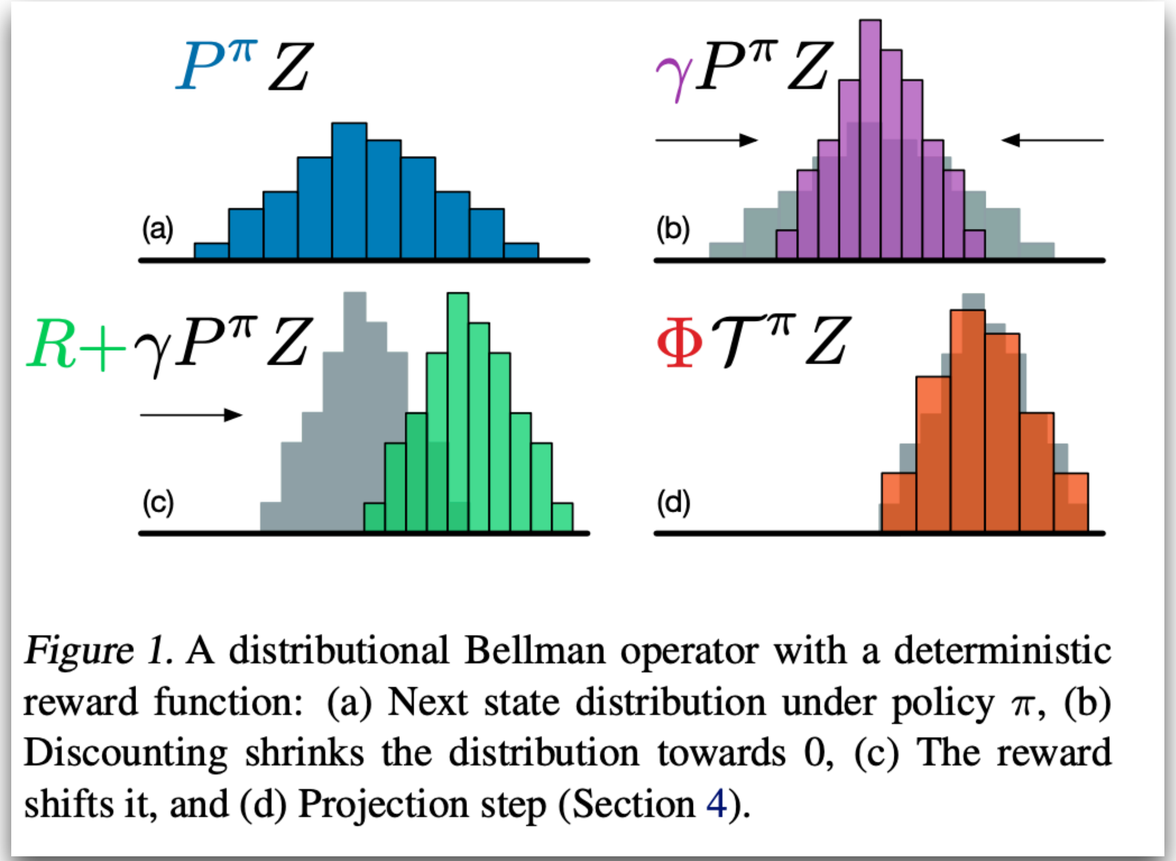


Figure 1. A distributional Bellman operator with a deterministic reward function: (a) Next state distribution under policy π, (b) Discounting shrinks the distribution towards 0, (c) The reward shifts it, and (d) Projection step (Section 4).

Rainbow-DQN

Improving the DQN algorithm

- **Distributional RL based on C51**

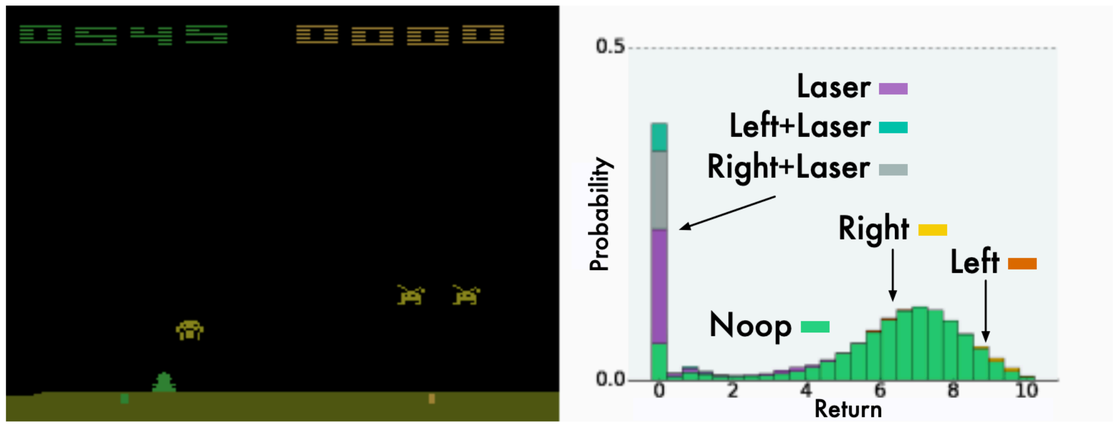
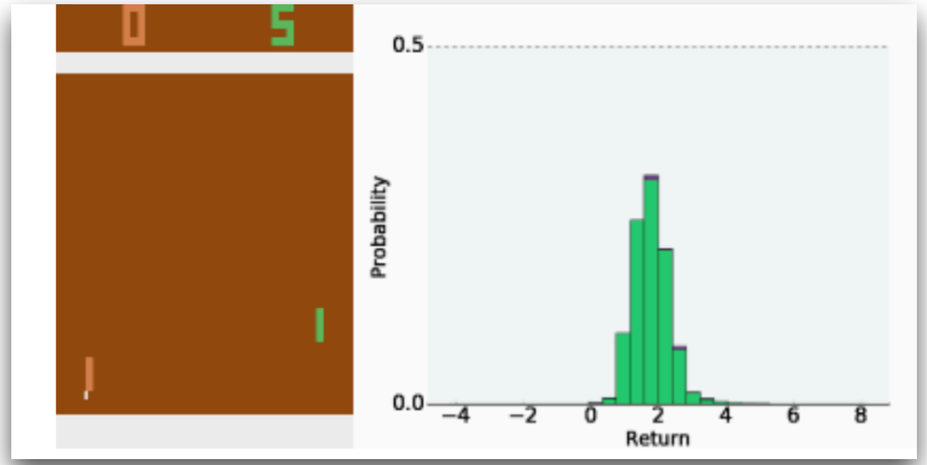
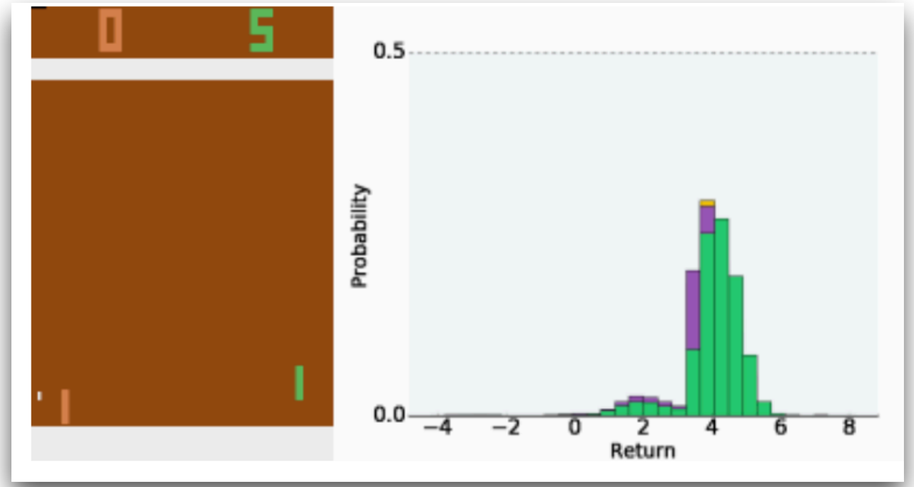


Figure 4. Learned value distribution during an episode of SPACE INVADERS. Different actions are shaded different colours. Returns below 0 (which do not occur in SPACE INVADERS) are not shown here as the agent assigns virtually no probability to them.

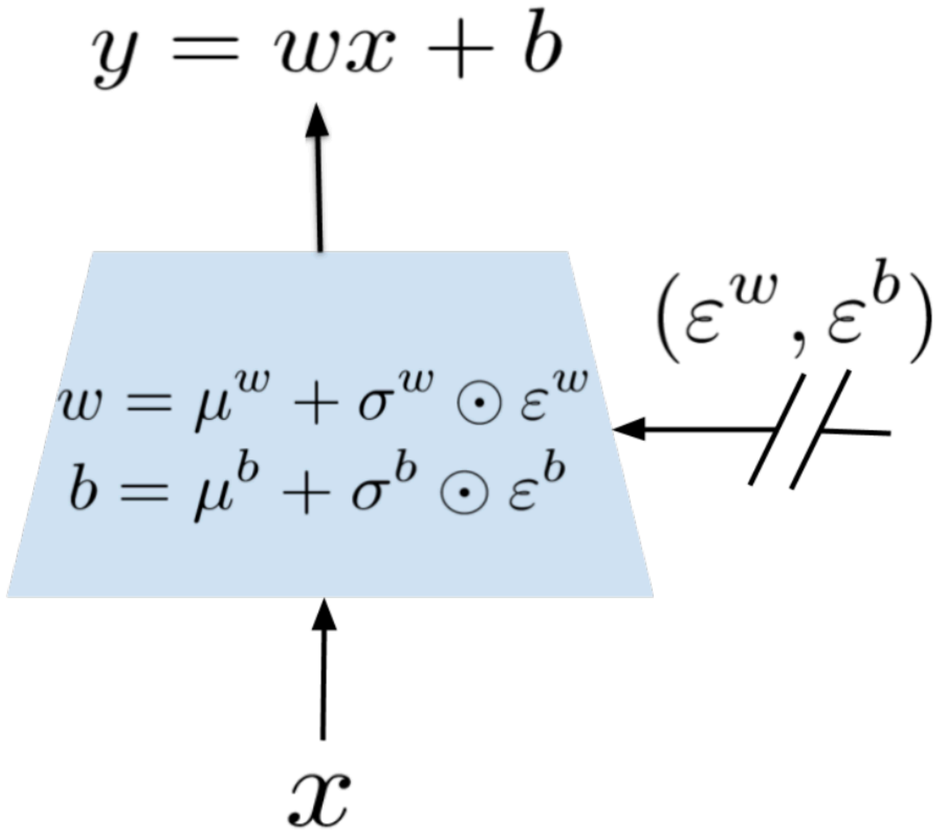




Rainbow-DQN

Noisy nets: An exploration technique

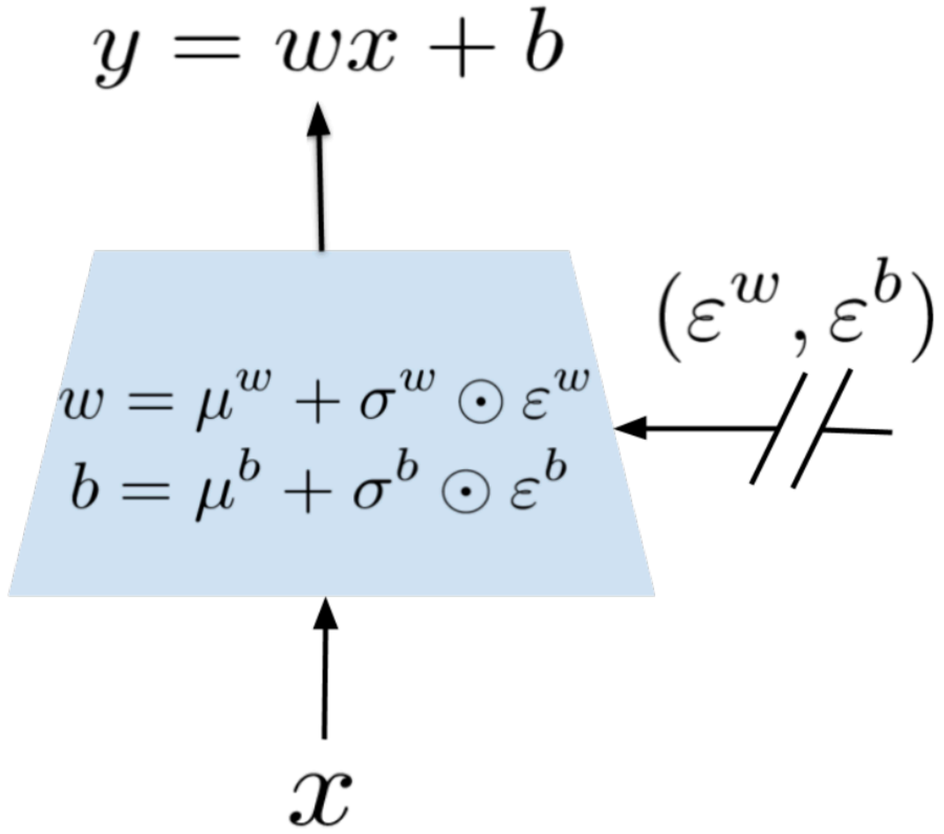
- **Noisy nets**
  - Add a noise layer to introduce stochastic behaviors for the policy
    - ϵ^w, ϵ^b are random variable
    - $\mu^w, \mu^b, \sigma^w, \sigma^b$ are training Variables



Rainbow-DQN

Noisy nets: An exploration technique

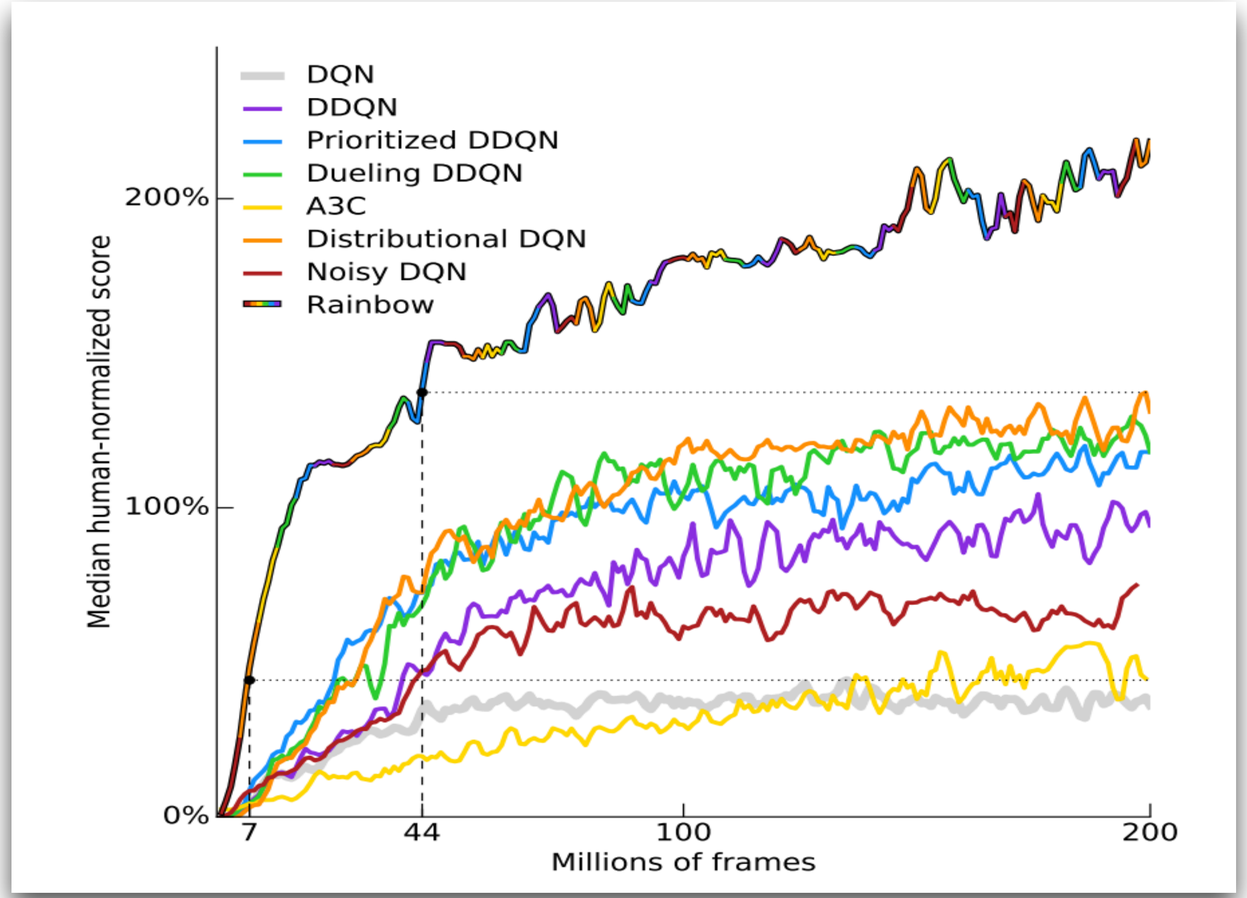
- **Noisy Nets**
  - Does not require the ϵ-greedy exploration strategy
  - Can be applied to other RL algorithms



Rainbow-DQN

Comparing Rainbow-DQN with the other DQN variants

- **Rainbow-DQN** outperforms the other DQN variants by a significant margin



ありがとうございます