This lecture will be recorded!

Welcome to

DS595 Reinforcement Learning

Prof. Yanhua Li

Time: 6:00pm –8:50pm W
Zoom Lecture
Fall 2022

Quiz 4 today in Week 9 (3/16 W)

- Linear Value Function Approximation（30 mins）
  - Stochastic Gradient Decent
  - VFA for policy evaluation
  - VFA for control

Quiz 5 in Week 12 (4/6 W)

- policy gradient (PG) RL（30 mins）
  - Basic PG,
  - REINFORCE PG,
  - and Vanilla PG)

Project 3 is Due 3/23 Wed, Week #10
Top three on the leader board get 10 bonus points

- https://users.wpi.edu/~yli15/courses/DS595Spring22/Assignments.html

- https://github.com/yingxue-zhang/DS595-RL-Projects/tree/master/Project3

Project 4 is available
Starts 3/23 Wed Week 10
Due 4/25 Monday Week 15

- https://users.wpi.edu/~yli15/courses/DS595Spring22/Assignments.html

- https://github.com/yingxue-zhang/DS595-RL-Projects/tree/master/Project4

A Project 4 self-intro session Wed in Week 9 (3/16)

We will have a

- Self Introduction Session on Wed in Week 9

- Who are you? Your expertise, such as programming experience, background knowledge of data mining, management, analytics.

- Experience on RL, Deep Learning, Data analytics

- Any initial idea for the open project 4?

Last Lecture

- Advanced DQN methods
  - Double-DQN
  - Prioritized DQN
  - Dueling DQN

- Project 3 (by Yingxue) starting from around 8:20PM
  - Project 3 description
  - Pytorch configuration and Google cloud environment

This Lecture

- Advanced DQN methods
  - Double-DQN
  - Dueling DQN
  - Prioritized DQN
  - Multi-step
  - Noisy net
  - Distributional Q-learning
  - Rainbow
  - Continuous actions
- Self-Introduction
- Imitation Learning / Inverse Reinforcement Learning
  - Introduction
  - Behavioral Cloning
  - Inverse reinforcement learning
    - Model-Based, Linear Reward Functions (this time)

	Reinforcement Learning	Inverse Reinforcement Learning
Single Agent	**Tabular representation of reward** Model-based control Model-free control (MC, SARSA, Q-Learning)	Linear reward function learning Imitation learning Apprenticeship learning Inverse reinforcement learning MaxEnt IRL MaxCausalEnt IRL MaxRelEnt IRL
	Function representation of reward *1. Linear value function approx* (MC, SARSA, Q-Learning) *2. Value function approximation* (Deep Q-Learning, Double DQN, prioritized DQN, Dueling DQN) *3. Policy function approximation* (Policy gradient, PPO, TRPO) *4. Actor-Critic methods* (A2C, A3C)	**Non-linear reward function learning** Generative adversarial imitation learning (GAIL) Adversarial inverse reinforcement learning (AIRL)
	Review of Deep Learning *As bases for non-linear function approximation (used in 2-4).*	**Review of Generative Adversarial nets** As bases for non-linear IRL
Multiple Agents	**Multi-Agent Reinforcement Learning** Multi-agent Actor-Critic etc.	**Multi-Agent Inverse Reinforcement Learning** MA-GAIL MA-AIRL AMA-GAIL

Applications

This Lecture

- Advanced DQN methods
  - Double-DQN
  - Dueling DQN
  - Prioritized DQN
  - Multi-step
  - Noisy net
  - Distributional Q-learning
  - Rainbow
  - Continuous actions
- Self-Introduction
- Imitation Learning / Inverse Reinforcement Learning
  - Introduction
  - Behavioral Cloning
  - Inverse reinforcement learning
    - Model-Based, Linear Reward Functions (this time)

Model-Free Deep Q-Learning

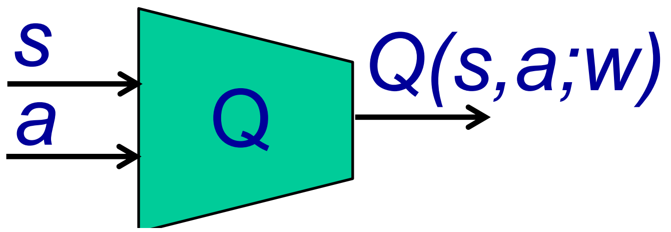
```
1: Initialize $\mathbf{w} = \mathbf{0}$, $k = 1$
2: loop
3:     Sample tuple $(s_k, a_k, r_k, s_{k+1})$ given $\pi$
4:     Update weights:
           $\Delta w = -\alpha(\underline{r_k + \gamma \max_{a_{k+1}} \hat{Q}(s_{k+1}, a_{k+1}; w)} - \underline{\hat{Q}(s_k, a_k; w)})\nabla_w \hat{Q}(s_k, a_k; w)$
           $w = w - \Delta w$
           $\pi(s_k) = \arg\max_{a_k} \hat{Q}(s_k, a_k)$, with prob $1 - \epsilon$, else random.
5:     $k = k + 1$
6: end loop
```

+ experience replay

    reduce correlations between samples

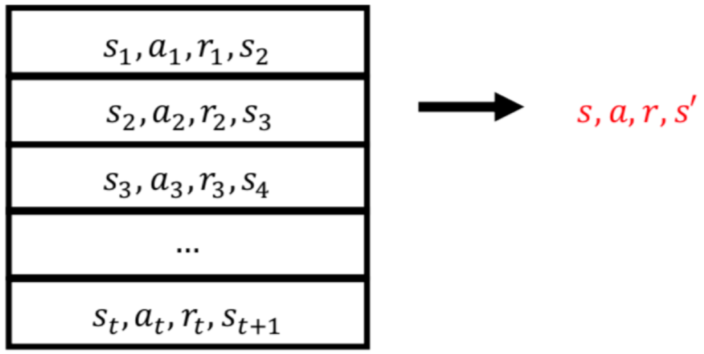
+ fixed target

    improve target stability



DQNs: Experience Replay

- To help remove correlations, store dataset (called a **replay buffer**) $\mathcal{D}$ from prior experience



- To perform experience replay, repeat the following:
  - $(s, a, r, s') \sim \mathcal{D}$: sample an experience tuple from the dataset
  - Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w})$
  - Use stochastic gradient descent to update the network weights

$$\Delta \mathbf{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w}) - \hat{Q}(s, a; \mathbf{w}))\nabla_{\mathbf{w}} \hat{Q}(s, a; \mathbf{w})$$

DQNs: Fixed Q-Targets

- To help improve stability, fix the **target weights** used in the target calculation for multiple updates
- Use a different set of weights to compute target than is being updated
- Let parameters $\boldsymbol{w}^-$ be the set of weights used in the target, and $\boldsymbol{w}$ be the weights that are being updated
- Slight change to computation of target value:
    - $(s, a, r, s') \sim \mathcal{D}$: sample an experience tuple from the dataset
    - Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}^-)$
    - Use stochastic gradient descent to update the network weights

$$\Delta \boldsymbol{w} = \text{-}\alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}^-) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})$$

Periodically, update the fixed Q-target -network by the current Q-network.

Q-Learning Algorithm with two tricks

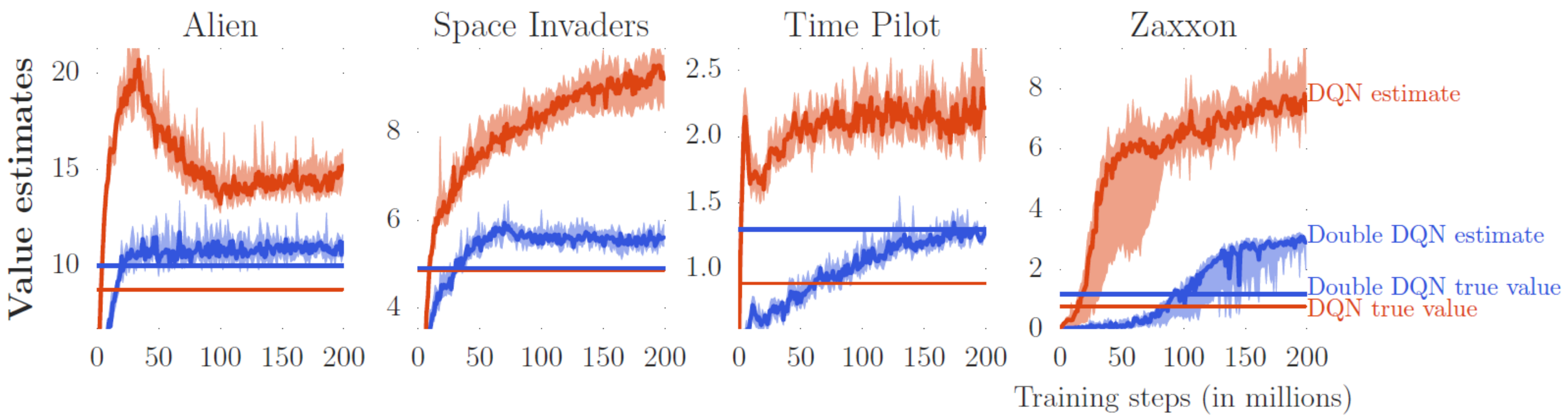
- Initialize Q-function Q, target Q-function $\hat{Q} = Q$
- In each episode
  - For each time step t
    - Given state s_t, take action a_t based on Q (epsilon greedy)
    - Obtain reward r_t, and reach new state s_{t+1}
    - Store (s_t, a_t, r_t, s_{t+1}) into buffer
    - Sample (s_i, a_i, r_i, s_{i+1}) from buffer (usually a batch)
    - Target $y = r_i + \max_a \hat{Q}(s_{i+1}, a)$
    - Update the parameters of Q to make $Q(s_i, a_i)$ close to y (regression)
    - Every C steps reset $\hat{Q} = Q$

This Lecture

- Advanced DQN methods
  - Double-DQN
  - Dueling DQN
  - Prioritized DQN
  - Multi-step
  - Noisy net
  - Distributional Q-learning
  - Rainbow
  - Continuous actions
- Self-Introduction
- Imitation Learning / Inverse Reinforcement Learning
  - Introduction
  - Behavioral Cloning
  - Inverse reinforcement learning
    - Model-Based, Linear Reward Functions (this time)

Double DQN

- Q value is usually over-estimated



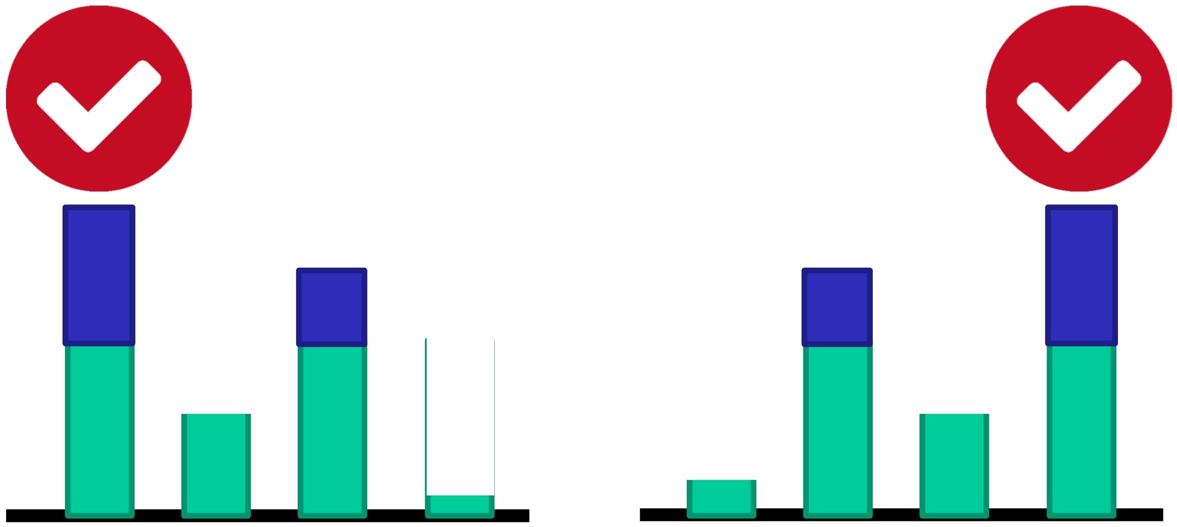
Double DQN

- Q value is usually over estimate

$$Q(s_t, a_t) \longleftrightarrow r_t + \max_a Q(s_{t+1}, a)$$

Tend to select the action that is over-estimated

$Q(s_{t+1}, a)$



Double DQN

- Q value is usually over estimate

$$Q(s_t, a_t) \longleftrightarrow r_t + \max_a Q(s_{t+1}, a)$$

- Double DQN: two functions Q Target Network

$$Q(s_t, a_t) \longleftrightarrow r_t + Q'\left(s_{t+1}, arg\max_a Q(s_{t+1}, a)\right)$$

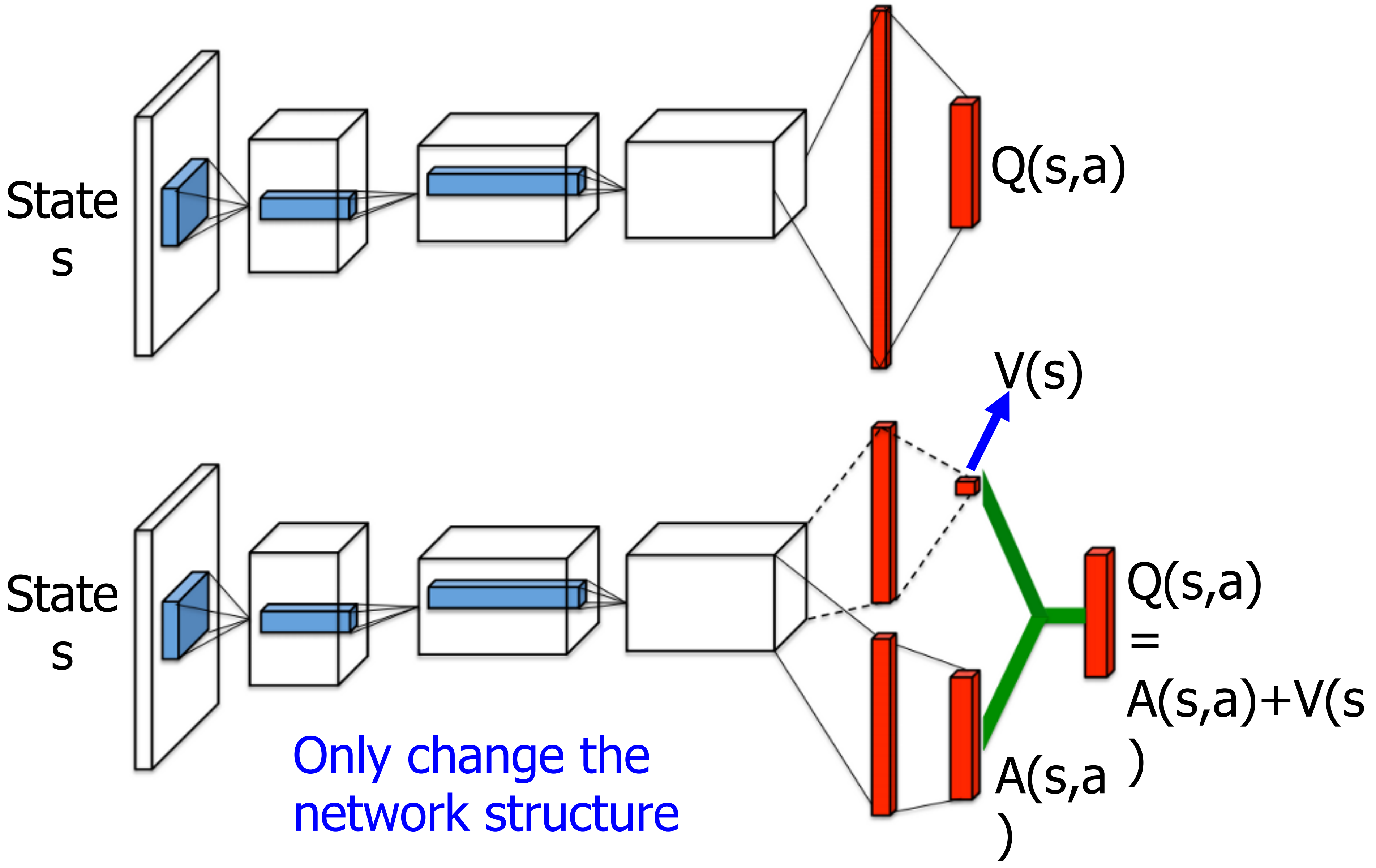
If Q over-estimate a, so it is selected. Q′ would give it proper value.

How about Q′ overestimate? The action will not be selected by Q.

Hado V. Hasselt, "Double Q-learning", NIPS 2010
Hado van Hasselt, Arthur Guez, David Silver, "Deep Reinforcement Learning with Double Q-learning", AAAI 2016

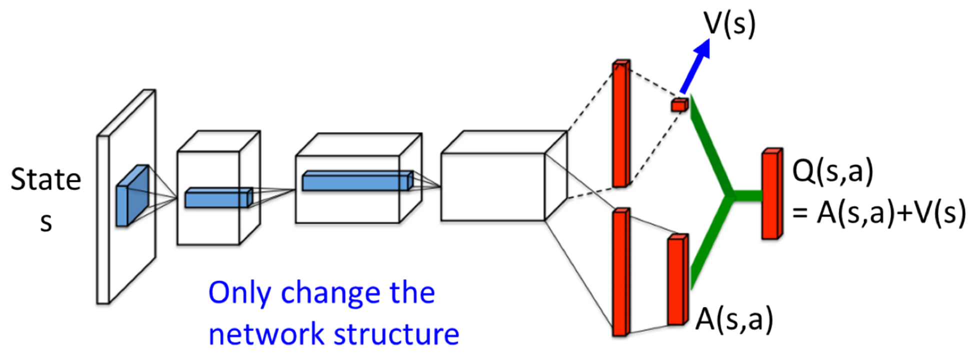
Dueling DQN

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, Nando de Freitas, "Dueling Network Architectures for Deep Reinforcement Learning", arXiv preprint, 2015



Only change the network structure

Dueling DQN



Q(s,a)

state

action	3	~~3~~ **4**	3	1
	1	~~-1~~ **0**	6	1
	2	~~-2~~ **-1**	3	1

=

V(s) Average of column

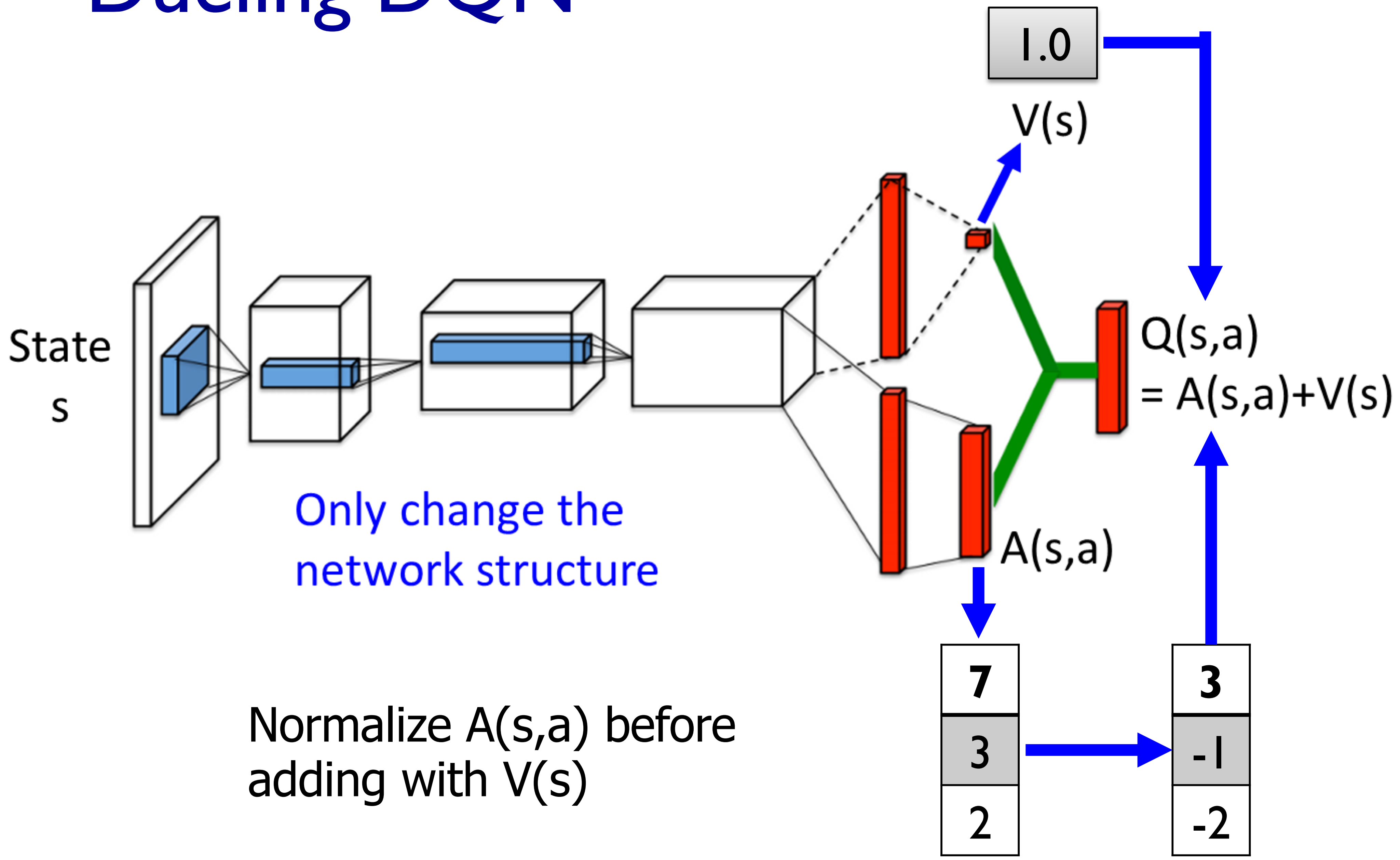
2	~~0~~ **1**	4	1

+

A(s,a) sum of column = 0

1	3	-1	0
-1	-1	2	0
0	-2	-1	0

Dueling DQN



Only change the network structure

Normalize A(s,a) before adding with V(s)

Dueling DQN - Visualization

(from the link of the original paper)

Dueling DQN - Visualization

Value Advantage

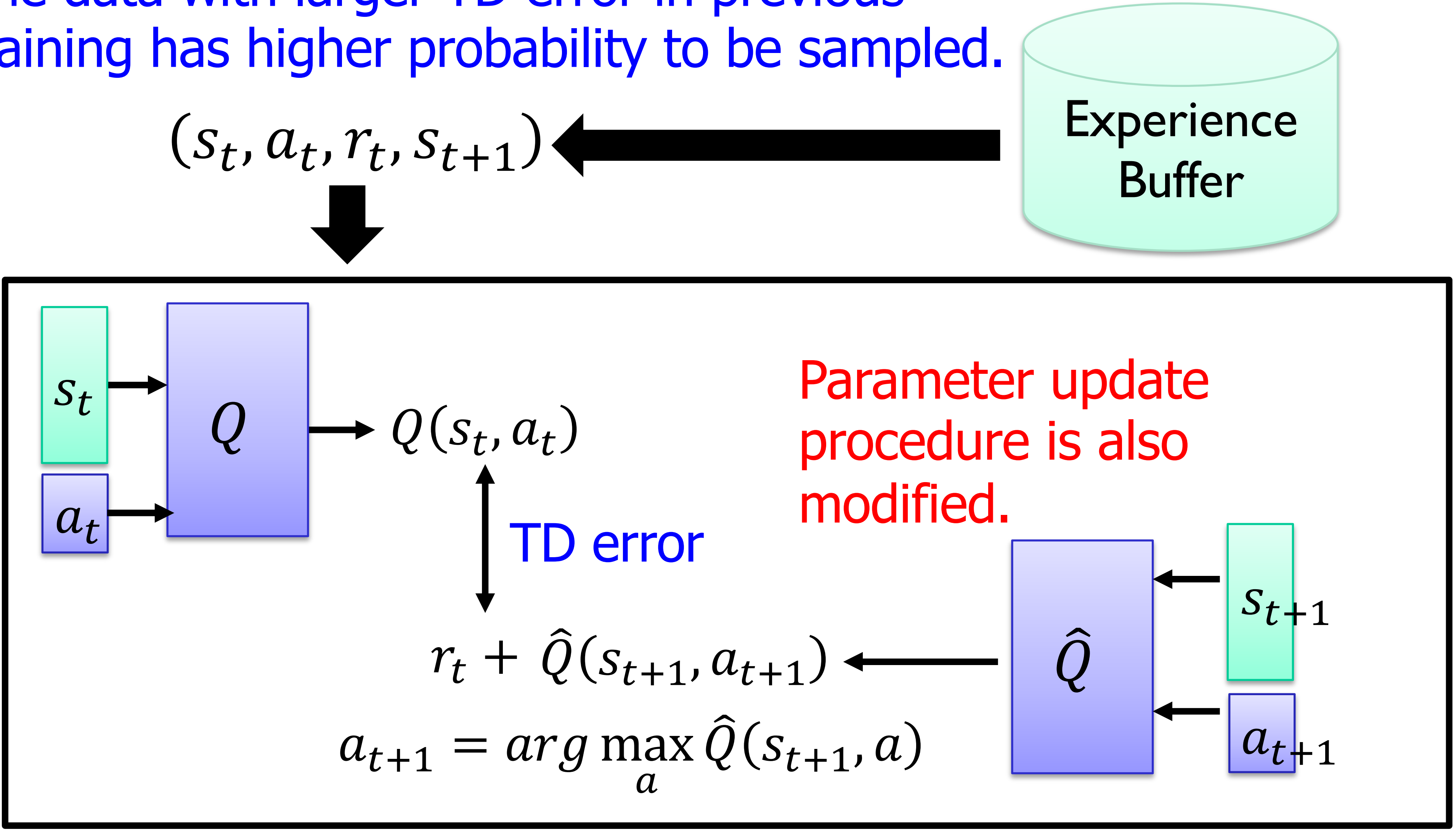
(from the link of the original paper)

https://arxiv.org/abs/1511.05952?context=cs

Prioritized Reply

The data with larger TD error in previous training has higher probability to be sampled.

Parameter update procedure is also modified.

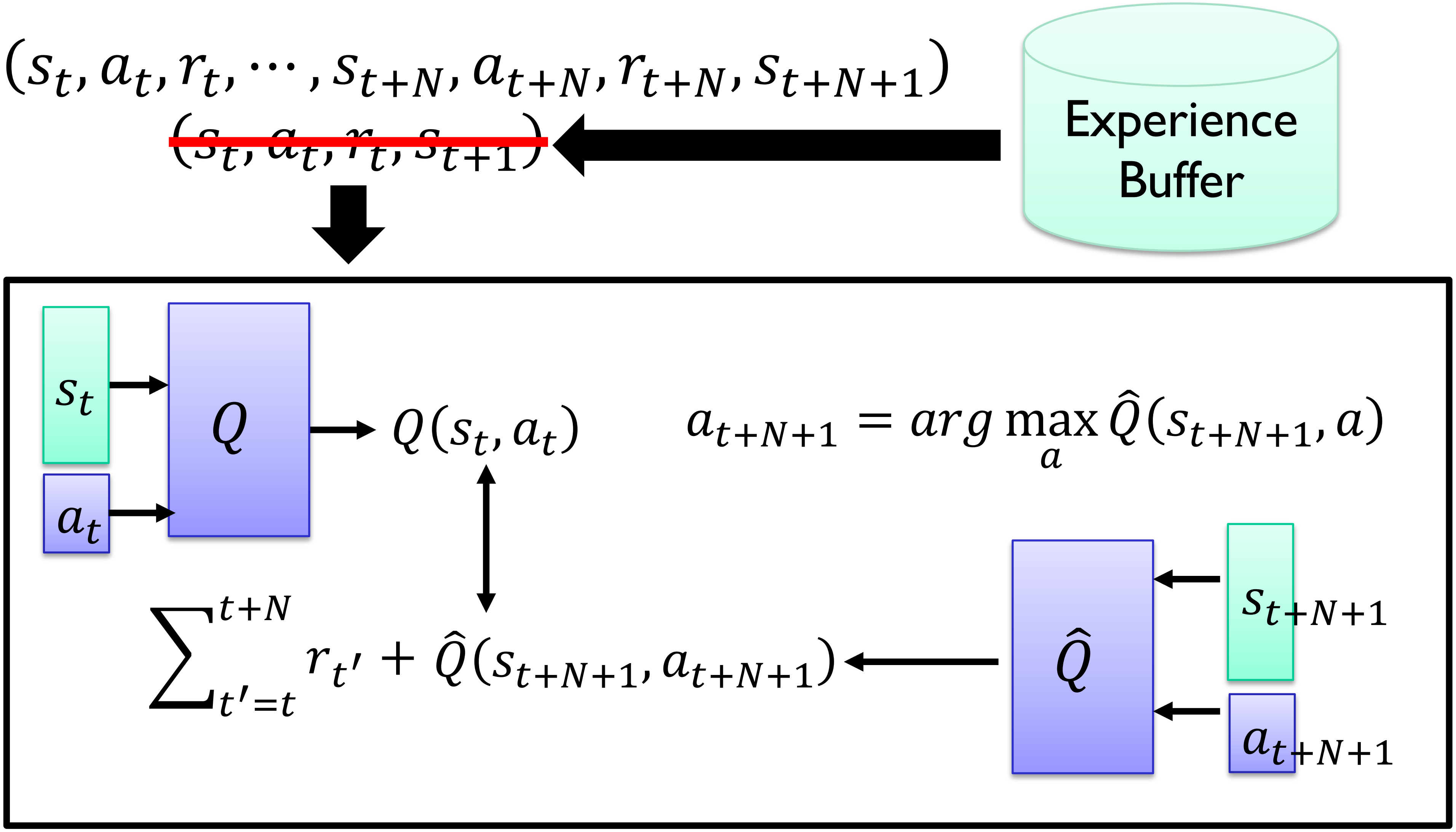

Experience Buffer
(s_t, a_t, r_t, s_{t+1})
s_t
a_t
Q
$Q(s_t, a_t)$
TD error
$r_t + \hat{Q}(s_{t+1}, a_{t+1})$
$a_{t+1} = arg\max_a \hat{Q}(s_{t+1}, a)$
$\hat{Q}$
s_{t+1}
a_{t+1}

Multi-step

Balance between MC and TD

$(s_t, a_t, r_t, \cdots, s_{t+N}, a_{t+N}, r_{t+N}, s_{t+N+1})$

~~(s_t, a_t, r_t, s_{t+1})~~


Experience Buffer
s_t
a_t
Q
$Q(s_t, a_t)$
$a_{t+N+1} = arg \max_a \hat{Q}(s_{t+N+1}, a)$
$\sum_{t'=t}^{t+N} r_{t'} + \hat{Q}(s_{t+N+1}, a_{t+N+1})$
$\hat{Q}$
s_{t+N+1}
a_{t+N+1}

Noisy Net

https://arxiv.org/abs/1706.01905
https://arxiv.org/abs/1706.10295

- Noise on Action (Epsilon Greedy)

$$a = \begin{cases} arg \max_a Q(s,a), & with\ probability\ 1 - \varepsilon \\ random, & otherwise \end{cases}$$

- Noise on Parameters

Inject noise into the parameters of Q-function **at the beginning of each episode**

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

$$Q(s,a) \xrightarrow[\text{Add noise}]{} \tilde{Q}(s,a)$$

The noise would **NOT** change in an episode.

Noisy Net

- Noise on Action
  - Given the same state, the agent may takes different actions.
  - No real policy works in this way

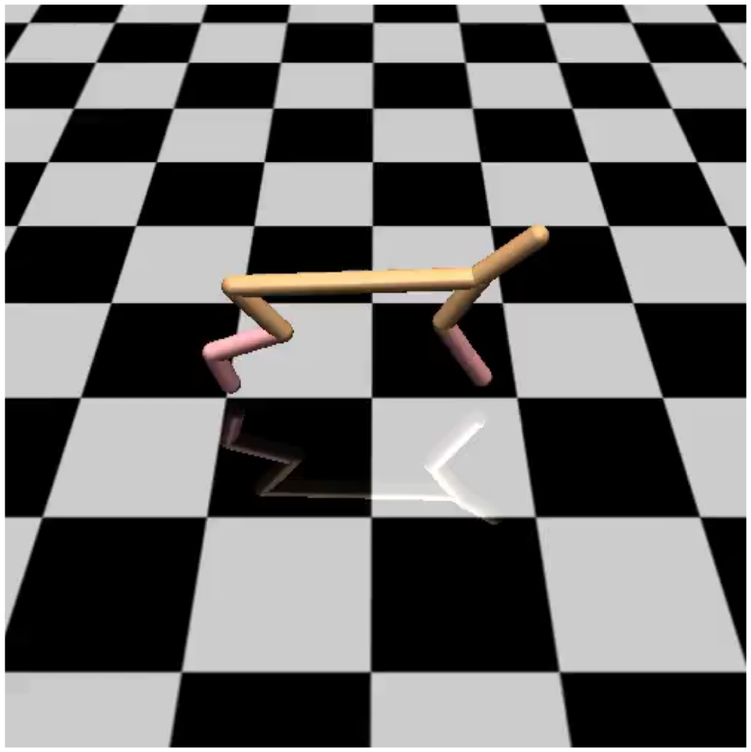
Random

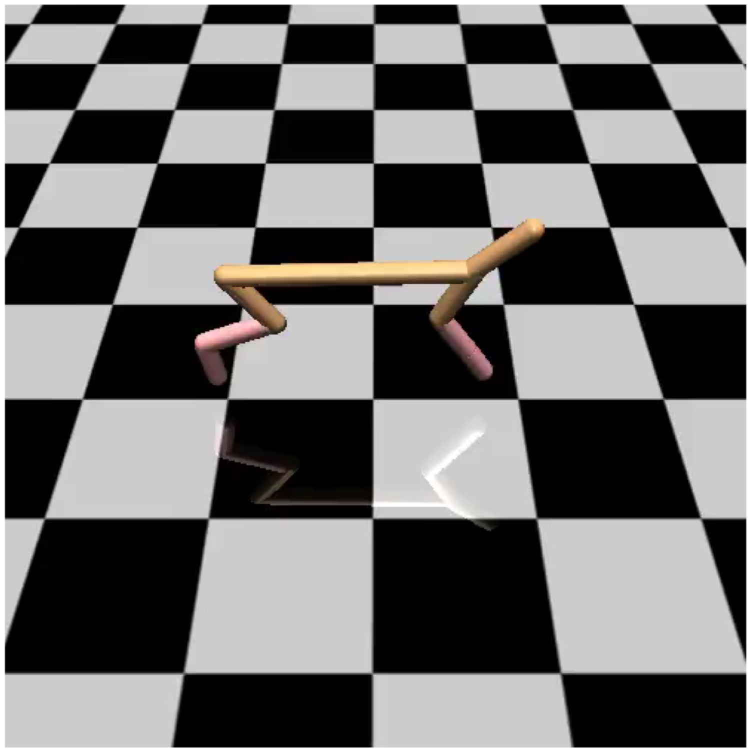
- Noise on Parameters
  - Given the same (similar) state, the agent takes the same action.
    - $\longrightarrow$ State-dependent Exploration
  - Explore in a *consistent* way

Systematic

Demo

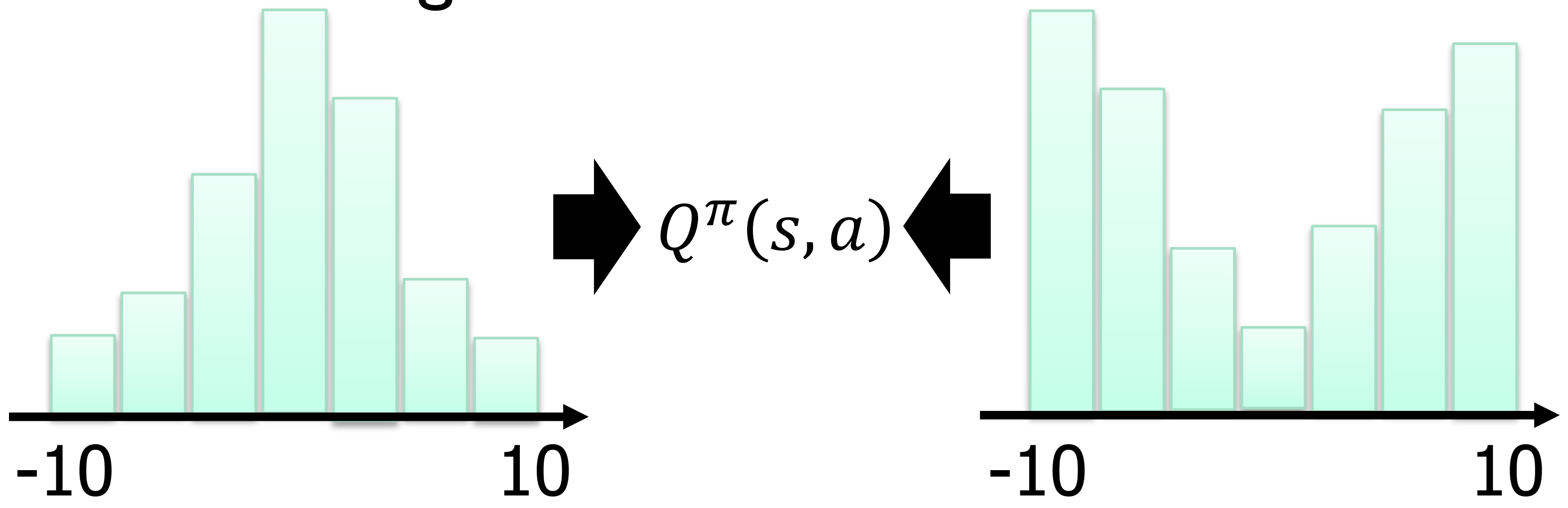
https://blog.openai.com/better-exploration-with-parameter-noise/





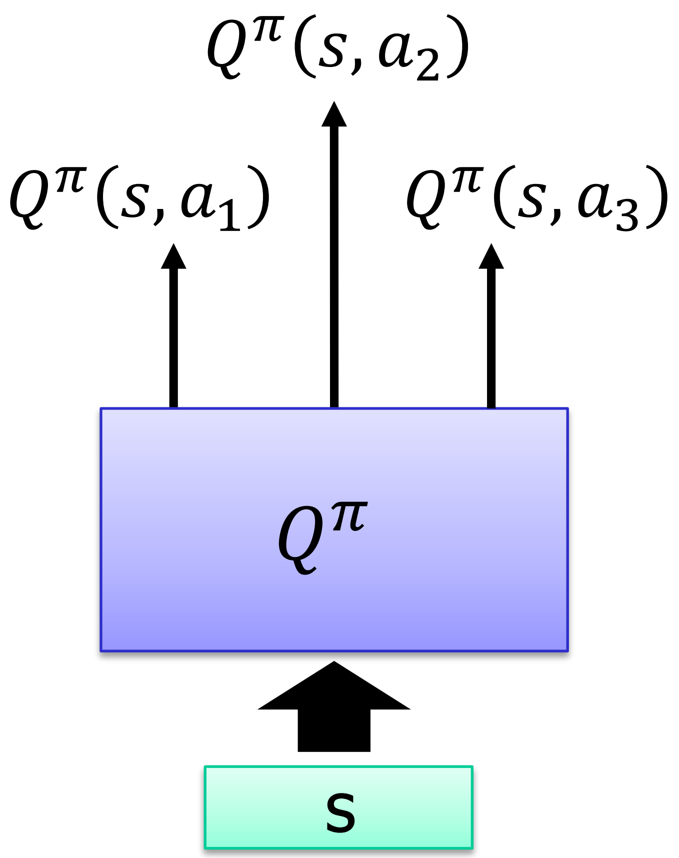
Distributional Q-function

- State-action value function $Q^{\pi}(s,a)$
  - When using actor π, the *cumulated* reward expects to be obtained after seeing observation s and taking a

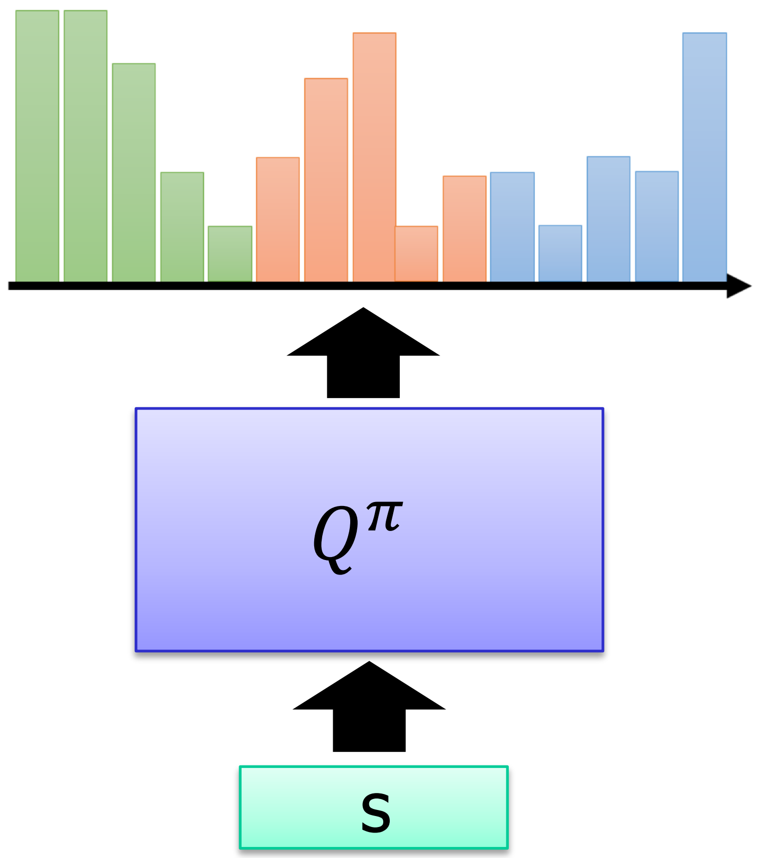


Different distributions can have the same values.

Distributional Q-function

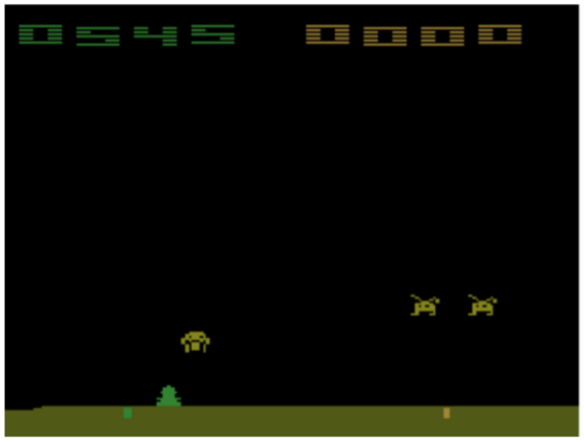


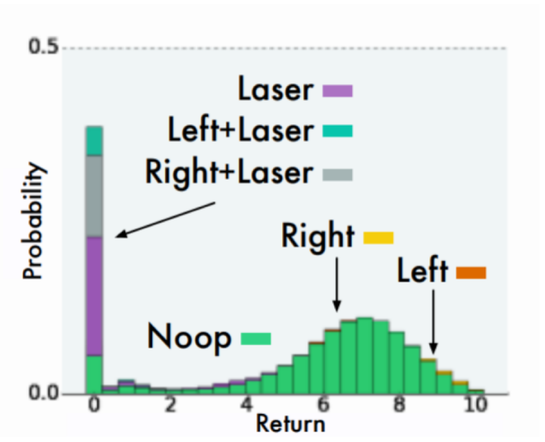
A network with 3 outputs

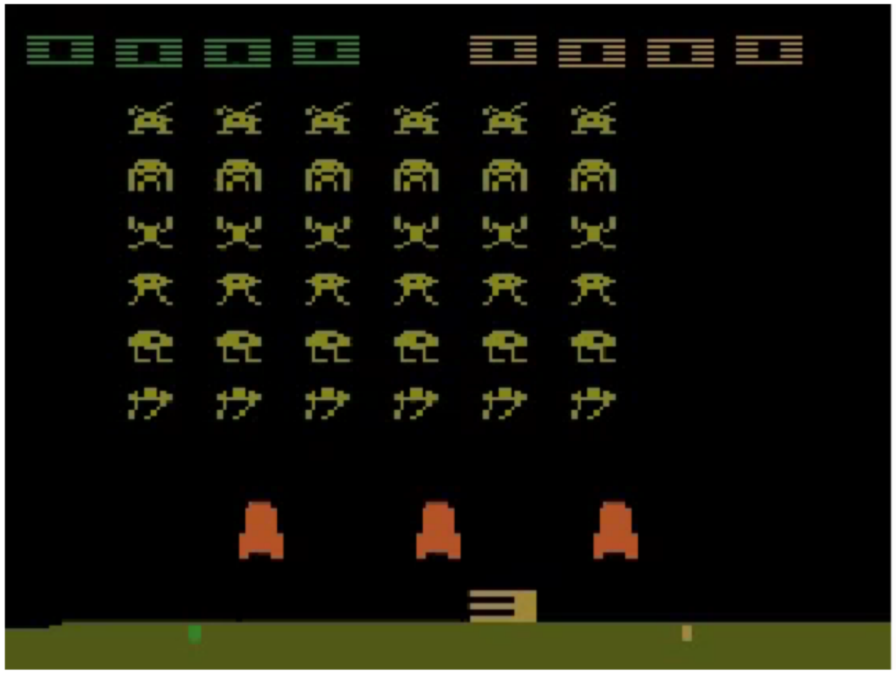


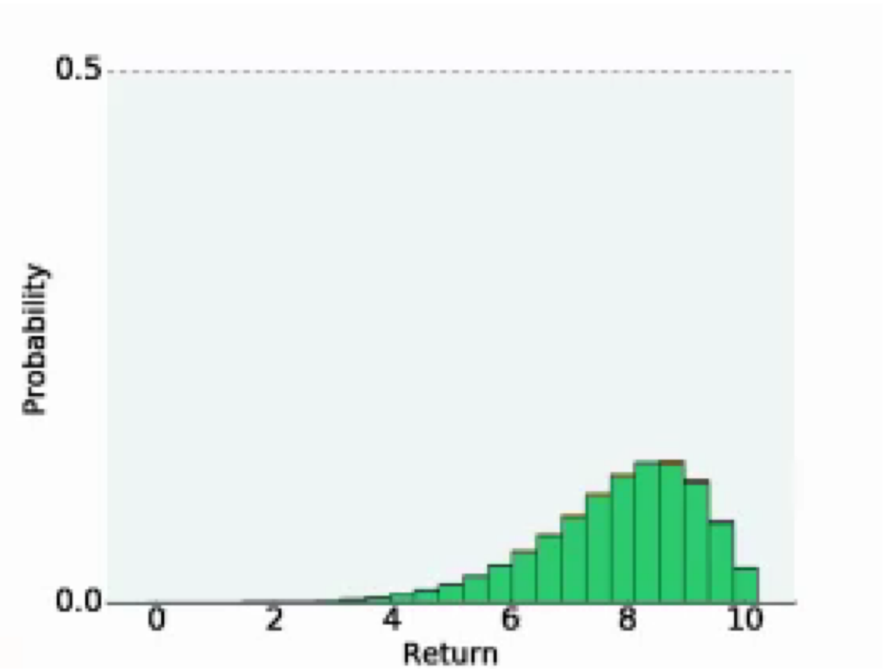
A network with 15 outputs
(each action has 5 bins)

Demo



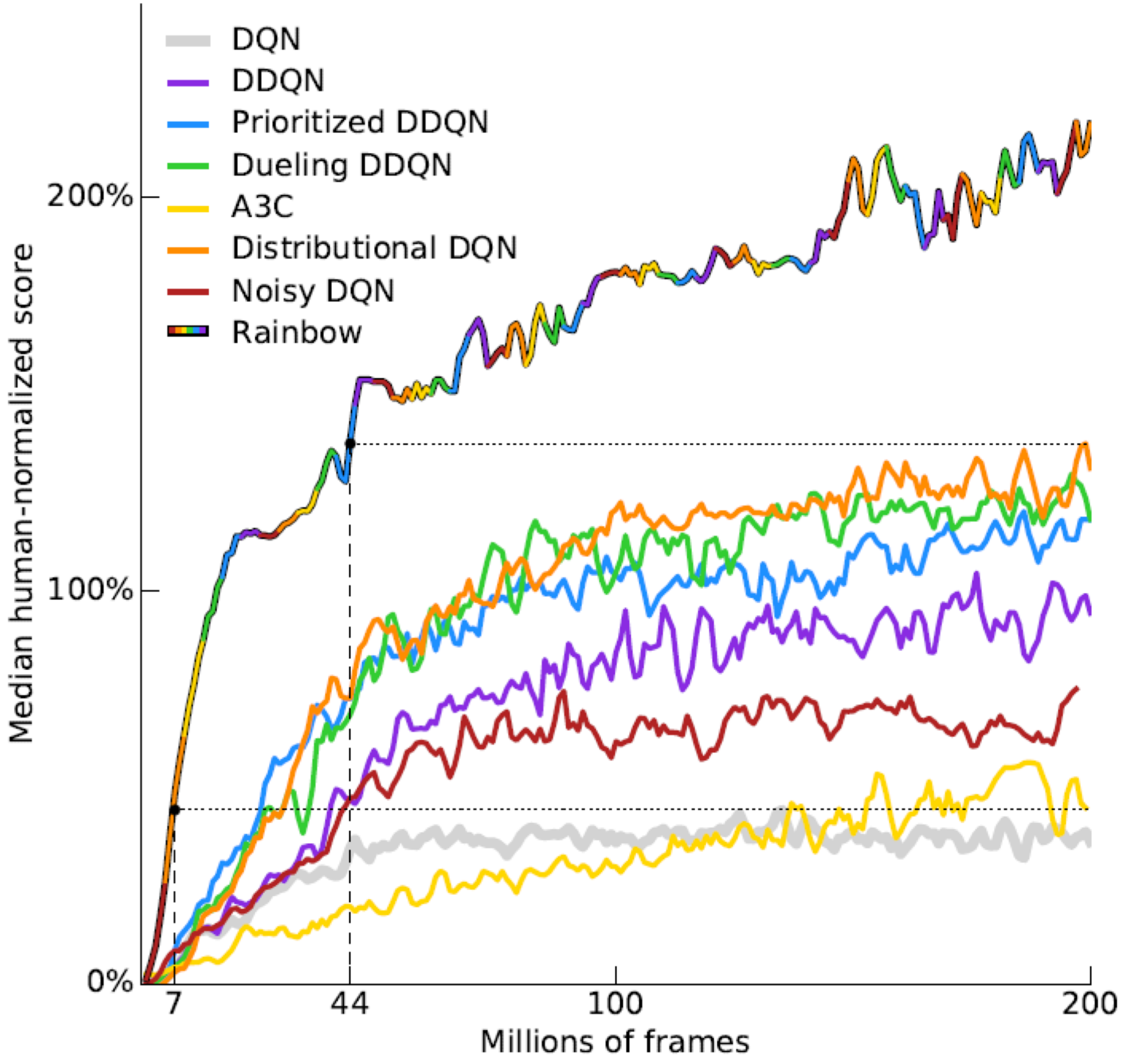






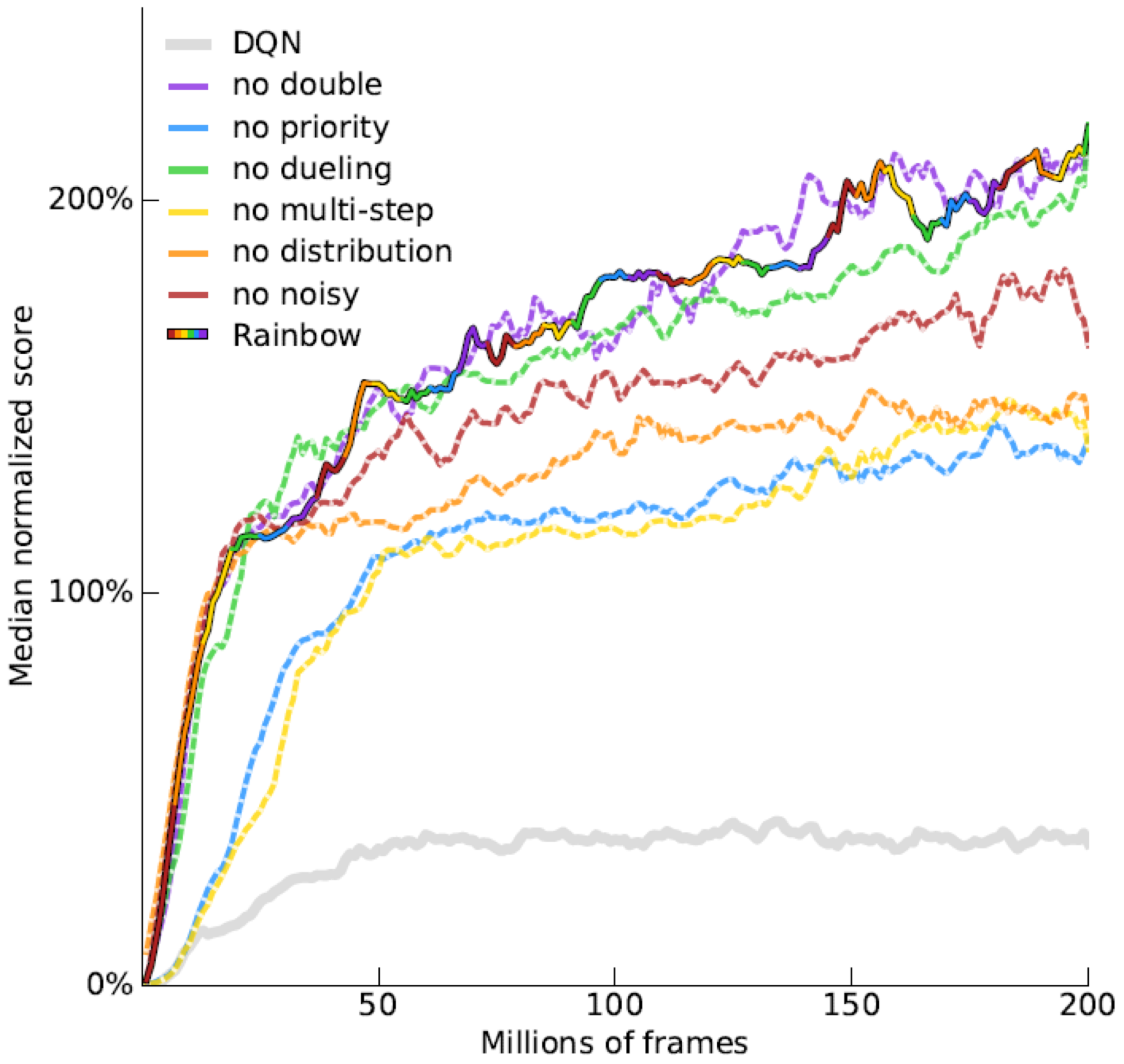
https://youtu.be/yFBwyPuO2Vg

Rainbow



https://arxiv.org/abs/1710.02298

Rainbow



https://arxiv.org/abs/1710.02298

Continuous Actions

- Action a is a *continuous vector*

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

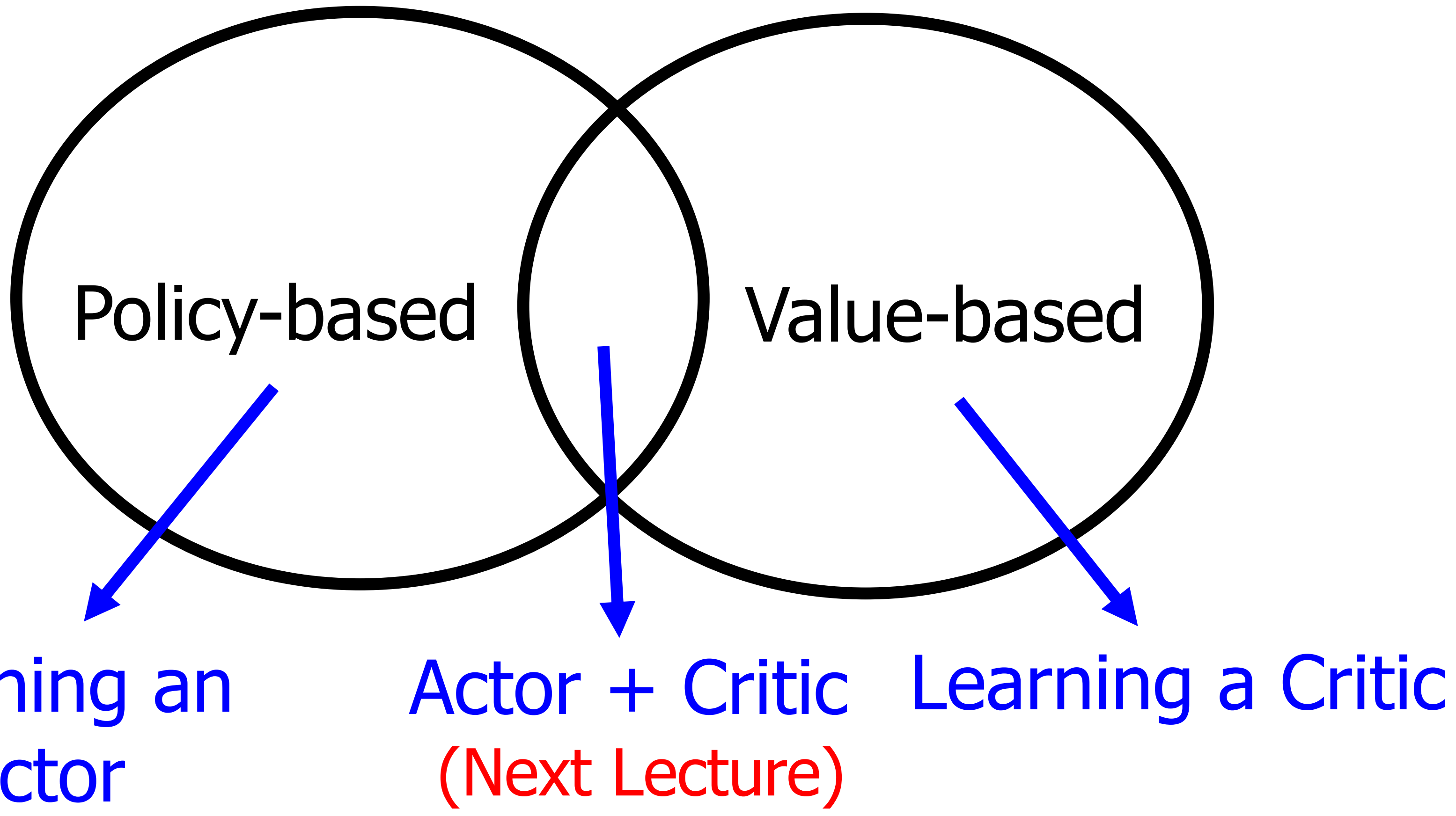
Solution 1

Sample a set of actions: $\{a_1, a_2, \cdots, a_N\}$

See which action can obtain the largest Q value

Continuous Actions

Solution 2 Don't use Q-learning



https://www.youtube.com/watch?v=ZhsEKTo7V04

This Lecture

- Advanced DQN methods
  - Double-DQN
  - Dueling DQN
  - Prioritized DQN
  - Multi-step
  - Noisy net
  - Distributional Q-learning
  - Rainbow
  - Continuous actions
- Self-Introduction
- Imitation Learning / Inverse Reinforcement Learning
  - Introduction
  - Behavioral Cloning
  - Inverse reinforcement learning
    - Model-Based, Linear Reward Functions (this time)

A Project 4 self-intro session Wed in Week 9 (3/16)

We will have a

- Self Introduction Session on Wed in Week 9

- Who are you? Your expertise, such as programming experience, background knowledge of data mining, management, analytics.

- Experience on RL, Deep Learning, Data analytics

- Any initial idea for the open project 4?

This Lecture

- Advanced DQN methods
  - Double-DQN
  - Dueling DQN
  - Prioritized DQN
  - Multi-step
  - Noisy net
  - Distributional Q-learning
  - Rainbow
  - Continuous actions
- Self-Introduction
- Imitation Learning / Inverse Reinforcement Learning
  - Introduction
  - Behavioral Cloning
  - Inverse reinforcement learning
    - Model-Based, Linear Reward Functions (this time)

Problems with many RL scenarios

- Reinforcement Learning:
  - Learning policies guided by (often sparse) rewards (e.g. win the game or not)
  - Pros: simple, cheap form of supervision
  - Cons: High sample complexity

Problems with many RL scenarios

- Where is it successful?
  - In simulation where data is cheap and parallelization is easy

- Not when:
  - Execution of actions is slow
  - Very expensive or not tolerable to fail
  - Want to be safe

Learning from Demonstrations (LfD)

- Expert provides a set of **demonstration trajectories**: sequences of states and actions
- Imitation learning is useful when is easier for the expert to demonstrate the desired behavior rather than:
  - come up with a reward that would generate such behavior,
  - coding up the desired policy directly

- Learning two things from imitation learning:
  - Policy
  - Reward function (why?)

Learning from Demonstrations (LfD)

- Expert provides a set of **demonstration trajectories**: sequences of states and actions
- Imitation learning is useful when is easier for the expert to demonstrate the desired behavior rather than:
  - come up with a reward that would generate such behavior,
  - coding up the desired policy directly

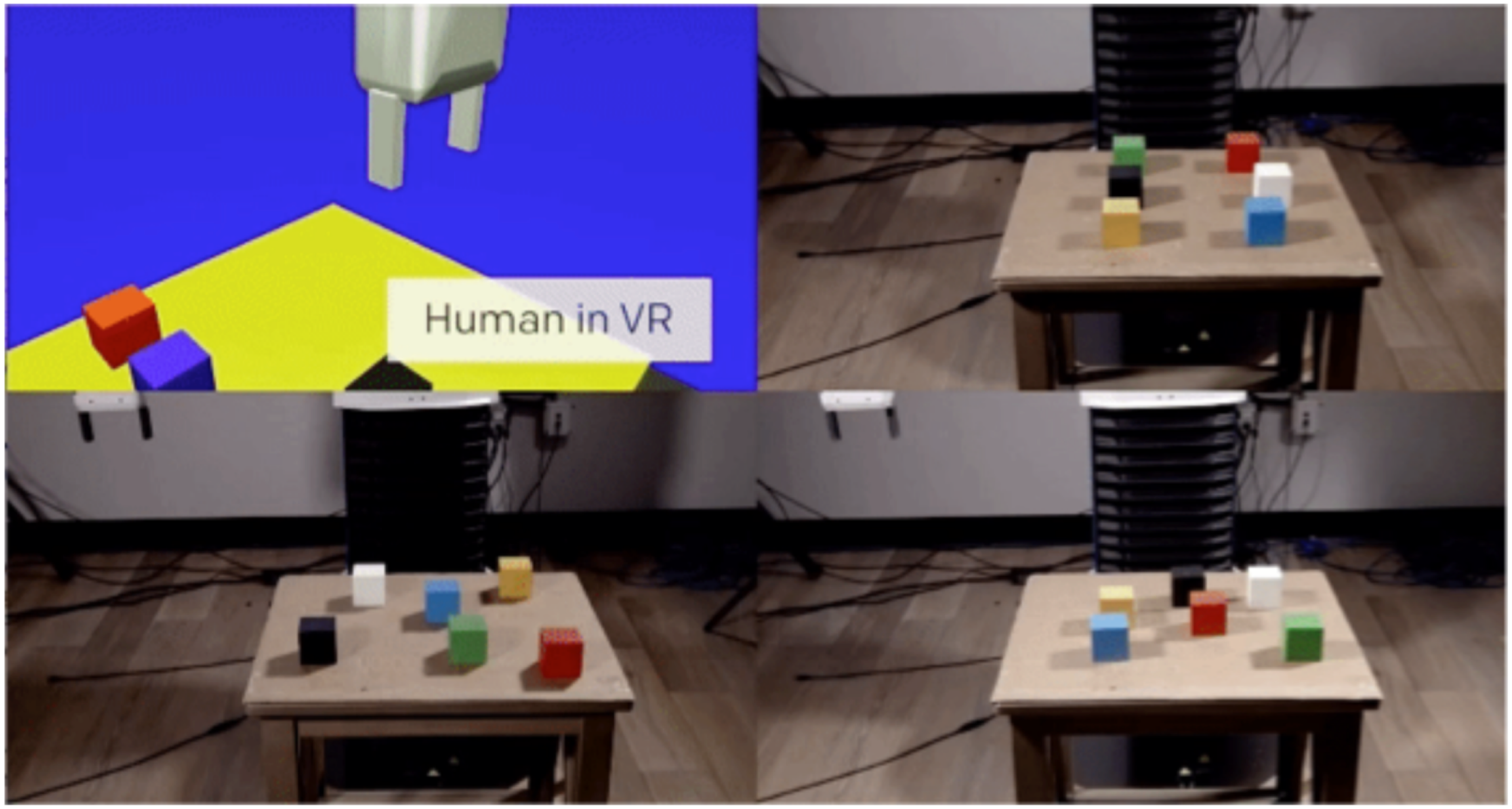
❖ Learning two things from imitation learning:

- Policy
- Reward function (why?)
  - Understand/reason how demonstrator makes decisions
  - Predict future behaviors
  - Good initial reward function for training RL agents

One Shot Imitation Learning

https://www.youtube.com/watch?v=oMZwkIjZzCM

Duan et al., NIPS ‘17



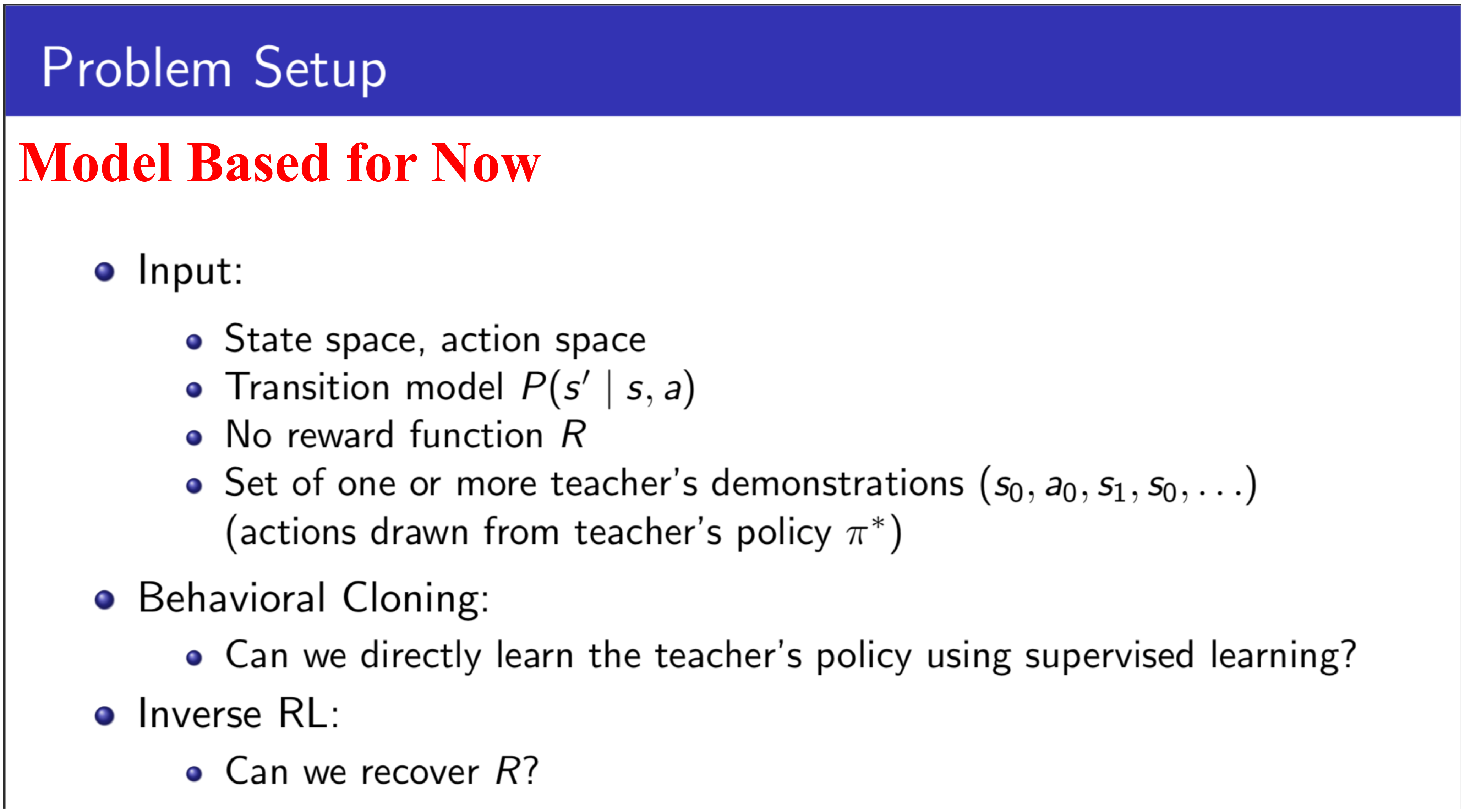
https://www.youtube.com/watch?v=oMZwkIjZzCM

The task that needs to be achieved is to stack blocks into 4 towers: "ab," "cde," "fg," and "hij," where the blocks are ordered from top to bottom within each group.

A Deep Learning Approach for Generalized Speech Animation
Sarah Taylor, Taehwan Kim, Yisong Yue et al., SIGGRAPH 2017

https://www.youtube.com/watch?v=9zL7qejW9fE

https://www.youtube.com/watch?v=9zL7qejW9fE



Problem Setup

Model Based for Now

- Input:
  - State space, action space
  - Transition model $P(s' \mid s, a)$
  - No reward function R
  - Set of one or more teacher's demonstrations $(s_0, a_0, s_1, s_0, \ldots)$ (actions drawn from teacher's policy π^*)
- Behavioral Cloning:
  - Can we directly learn the teacher's policy using supervised learning?
- Inverse RL:
  - Can we recover R?

We will discuss model-free (i.e., unknown P) in future lectures.

This Lecture

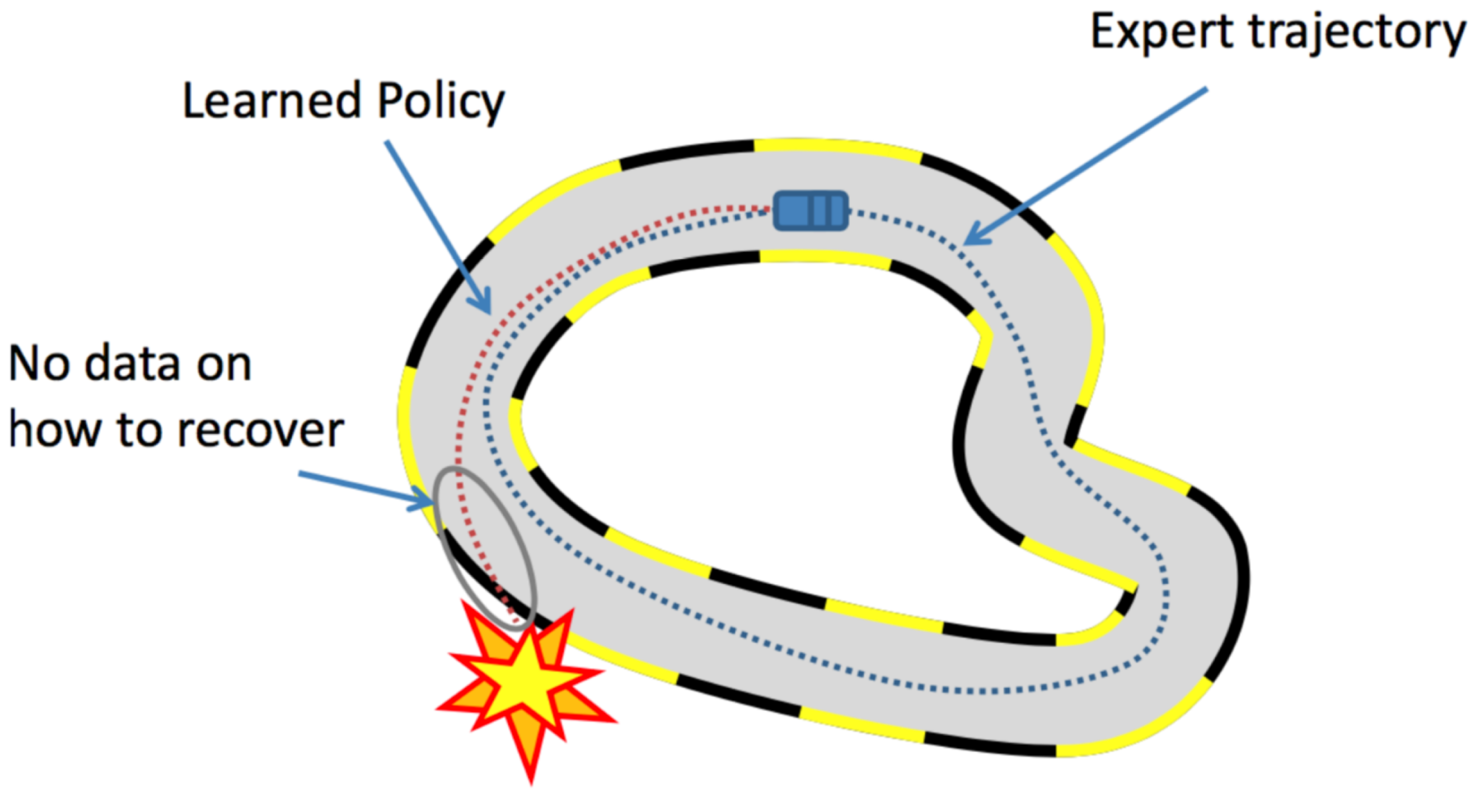
- Advanced DQN methods
  - Double-DQN
  - Dueling DQN
  - Prioritized DQN
  - Multi-step
  - Noisy net
  - Distributional Q-learning
  - Rainbow
  - Continuous actions
- Self-Introduction
- Imitation Learning / Inverse Reinforcement Learning
  - Introduction
  - Behavioral Cloning
  - Inverse reinforcement learning
    - Model-Based, Linear Reward Functions (this time)

Behavioral Cloning

- Formulate problem as a standard machine learning problem:
    - Fix a policy class (e.g. neural network, decision tree, etc.)
    - Estimate a policy from training examples $(s_0, a_0), (s_1, a_1), (s_2, a_2), \ldots$

Problem with the BC approach?

Problem: Compounding Errors



Data distribution mismatch!

In supervised learning, $(x, y) \sim D$ during train **and** test. In MDPs:

- Train: $s_t \sim D_{\pi^*}$
- Test: $s_t \sim D_{\pi_\theta}$

This Lecture

- Advanced DQN methods
  - Double-DQN
  - Dueling DQN
  - Prioritized DQN
  - Multi-step
  - Noisy net
  - Distributional Q-learning
  - Rainbow
  - Continuous actions
- Self-Introduction
- Imitation Learning / Inverse Reinforcement Learning
  - Introduction
  - Behavioral Cloning
  - Inverse reinforcement learning
    - Model-Based, Linear Reward Functions (this time)

Linear Feature Reward Inverse RL

- Recall linear value function approximation
- Similarly, here consider when reward is linear over features
  - $R(s) = \boldsymbol{w}^T x(s)$ where $w \in \mathbb{R}^n, x : S \to \mathbb{R}^n$
- Goal: identify the weight vector $\boldsymbol{w}$ given a set of demonstrations
- The resulting value function for a policy π can be expressed as

$$V^\pi = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi]$$

Linear Feature Reward Inverse RL

- Recall linear value function approximation
- Similarly, here consider when reward is linear over features
  - $R(s) = \boldsymbol{w}^T x(s)$ where $w \in \mathbb{R}^n, x : S \rightarrow \mathbb{R}^n$
- Goal: identify the weight vector $\boldsymbol{w}$ given a set of demonstrations
- The resulting value function for a policy π can be expressed as

$$
\begin{aligned}
V^\pi = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi] \quad &= \mathbb{E}[\textstyle\sum_{t=0}^{\infty} \gamma^t \boldsymbol{w}^T x(s_t) \mid \pi] \\
&= w^T \mathbb{E}[\textstyle\sum_{t=0}^{\infty} \gamma^t x(s_t) \mid \pi] \\
&= \boldsymbol{w}^T \mu(\pi)
\end{aligned}
$$

where $\mu(\pi)(s)$ is defined as the discounted weighted frequency of state features under policy π.

Inverse Reinforcement Learning

To find the reward function R used by the expert:

- Note $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R^*(s_t) \mid \pi^*] = V^* \geq V^\pi = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R^*(s_t) \mid \pi] \quad \forall \pi,$
- Therefore if the expert's demonstrations are from the optimal policy, to identify ***w*** it is sufficient to find w^* such that

$$w^{*T} \mu(\pi^*) \geq w^{*T} \mu(\pi), \forall \pi \neq \pi^*$$

Inverse reinforcement learning

- **Goal:** Learn a policy function and a reward function that are as good as the demonstration expert
- Linear reward function assumption: $R(s) = w^T x(s)$

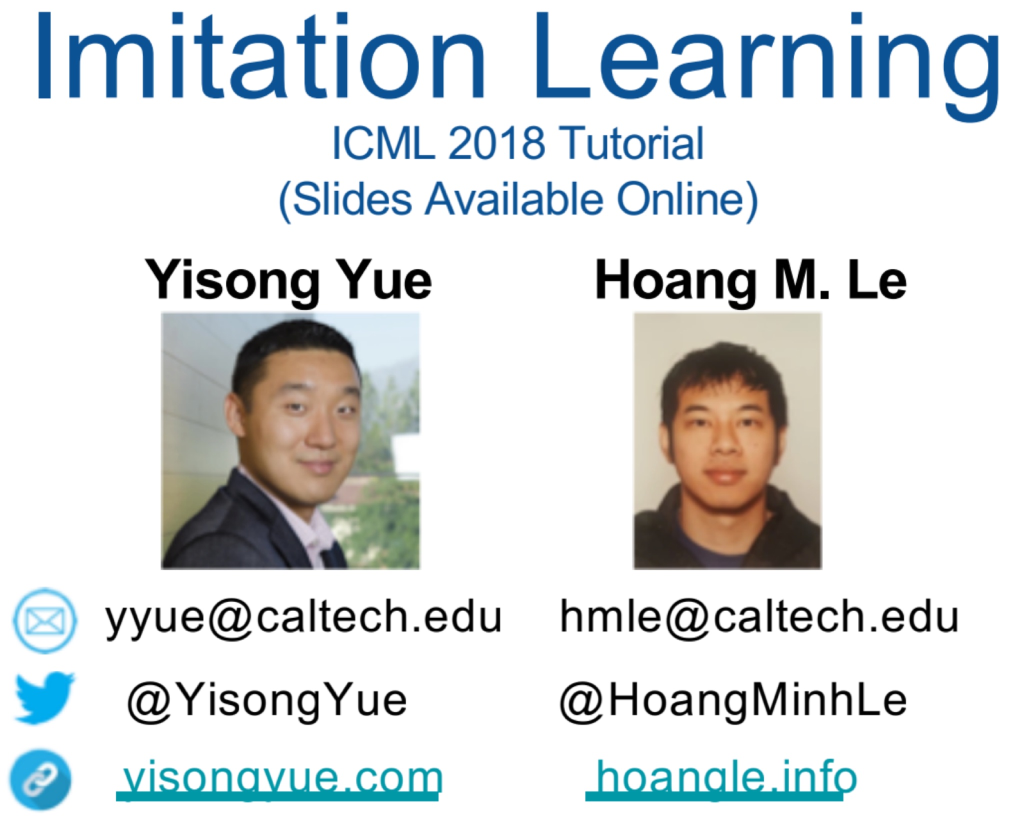
  - Initialize $\pi=\pi_0$, stopping criteria $\varepsilon=10^{-3}$ (for example)
  - For i=1,2,… …
    - Find a reward function that the expert maximally outperforms previous policies: (Any quadratic programming solver)

      $$\arg\max_{\mathrm{w}} (\mathrm{w}^\top \mu(\pi^*) - \mathrm{w}^\top \mu(\pi)), \text{ s.t., } \|\mathrm{w}\|_2 \le 1$$
    - Find the optimal π_i with the current w (dynamic programming)
    - Exit if $\mathrm{w}^\top \mu(\pi^*) - \mathrm{w}^\top \mu(\pi) \le \epsilon/2$
    - $\pi = \pi^*$

$\downarrow$ Suppose it is model-based, i.e., environment dynamics is known.

More on Imitation Learning

- Slides: https://drive.google.com/file/d/12QdNmMII-bGISWnm8pmD_TawuRN7xagX/view
- Video: https://www.youtube.com/watch?v=WjFdD7PDGw0



Imitation Learning

ICML 2018 Tutorial

(Slides Available Online)

Yisong Yue — yyue@caltech.edu — @YisongYue — yisongyue.com

Hoang M. Le — hmle@caltech.edu — @HoangMinhLe — hoangle.info

Next Lecture

- Other deep reinforcement learning approaches

  - Value based DRL (DQN),

  - Policy based DRL
    - Policy Gradient
    - Proximal Policy Optimization, PPO, -> PPO2
    - TRPO (Trust Region Policy Optimization, TRPO

  - (Asynchronous) Advantage Actor Critic:
    - A2C
    - A3C

Questions?