535514: Reinforcement Learning

Lecture 21 — Double Q-Learning, DQN, DDQN, and Distributional RL

Ping-Chun Hsieh

May 6, 2024

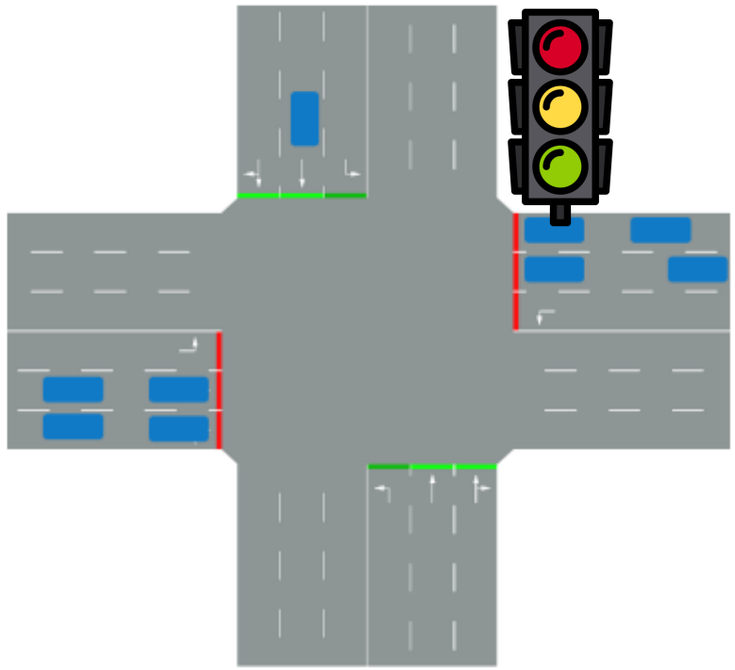
A Quick Mention of Multi-Objective RL

Robotics

- Speed vs energy cost

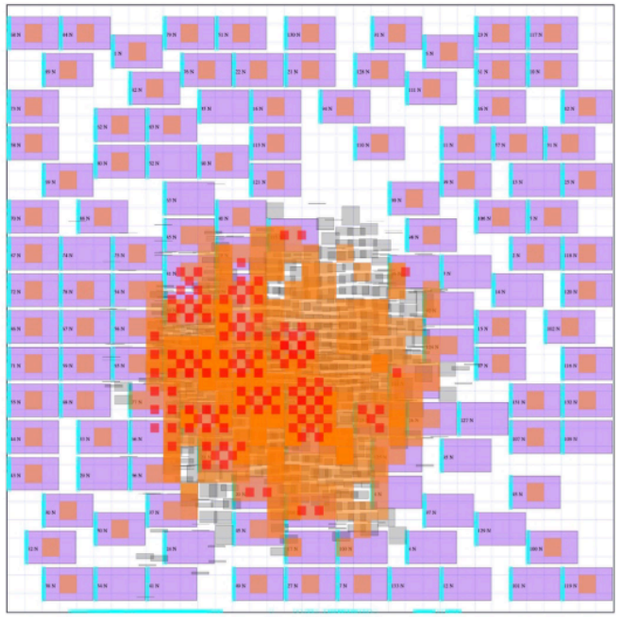
  (Normal & battery-saving mode)

Traffic Light Control



- Throughput, waiting time, and # of phase switches

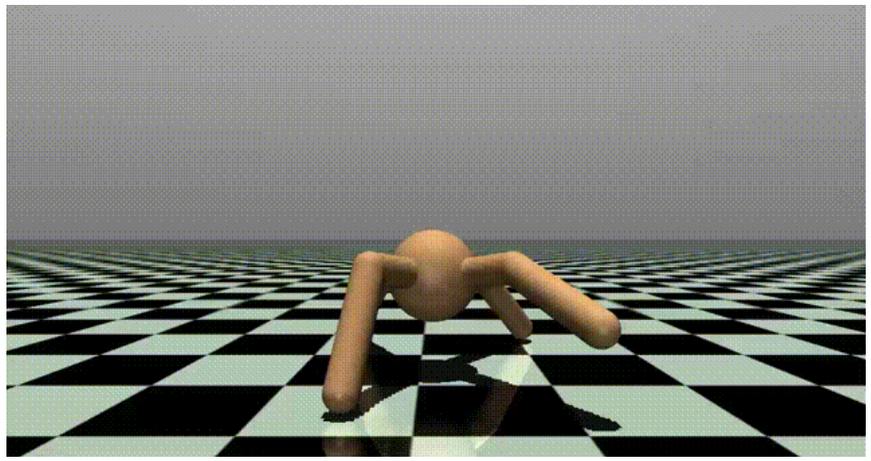
Chip Design



- Wirelength, routing congestion, and density

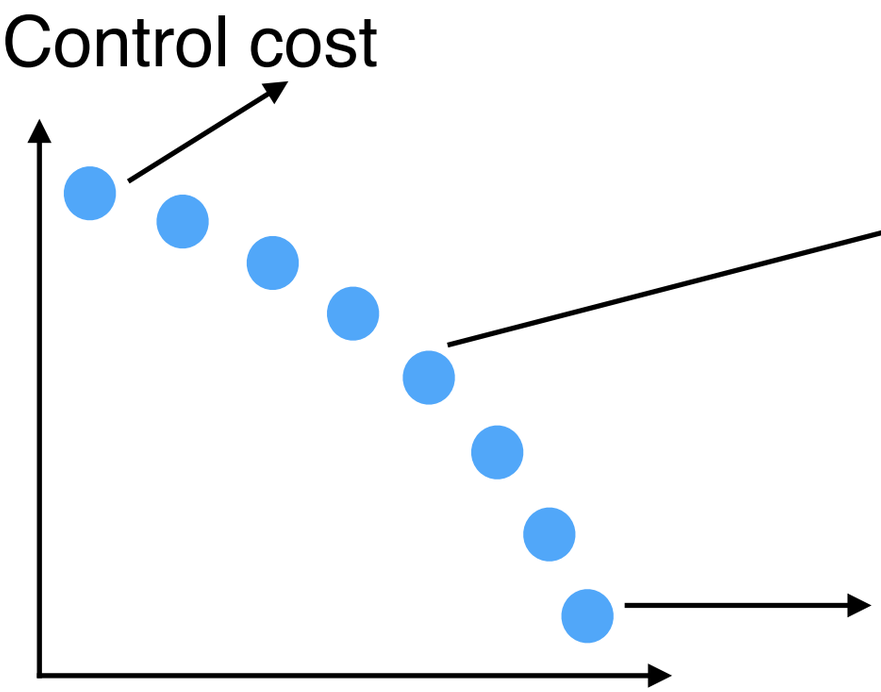
"Preference-dependent" optimal strategies!

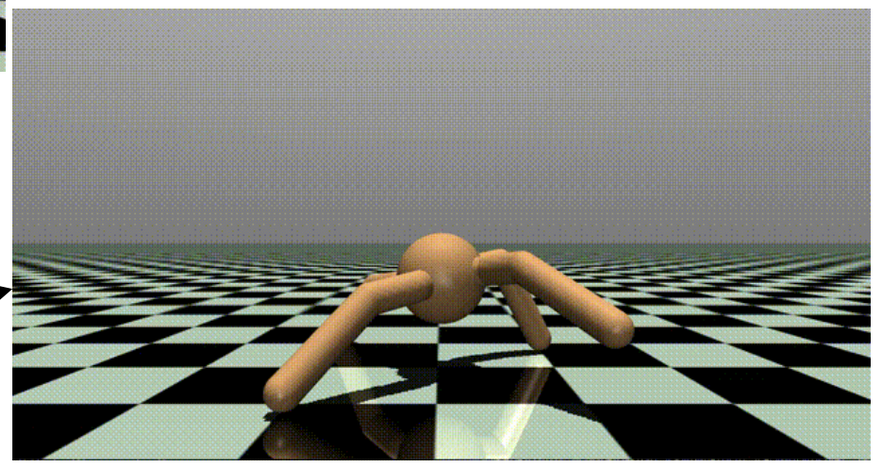
MORL Learns Diverse Behaviors



$\mathbf{w} = [0.1,0.9]$ (favor control cost)

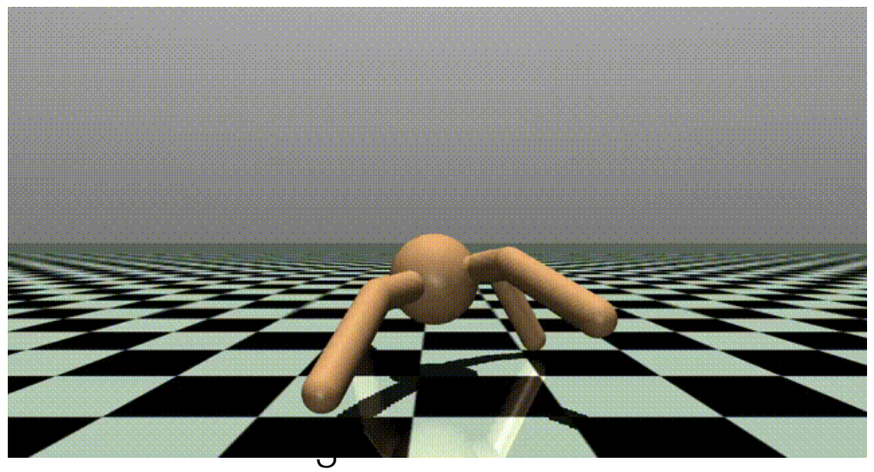
Balance with minimal control cost





$\mathbf{w} = [0.5,0.5]$

Walk gracefully

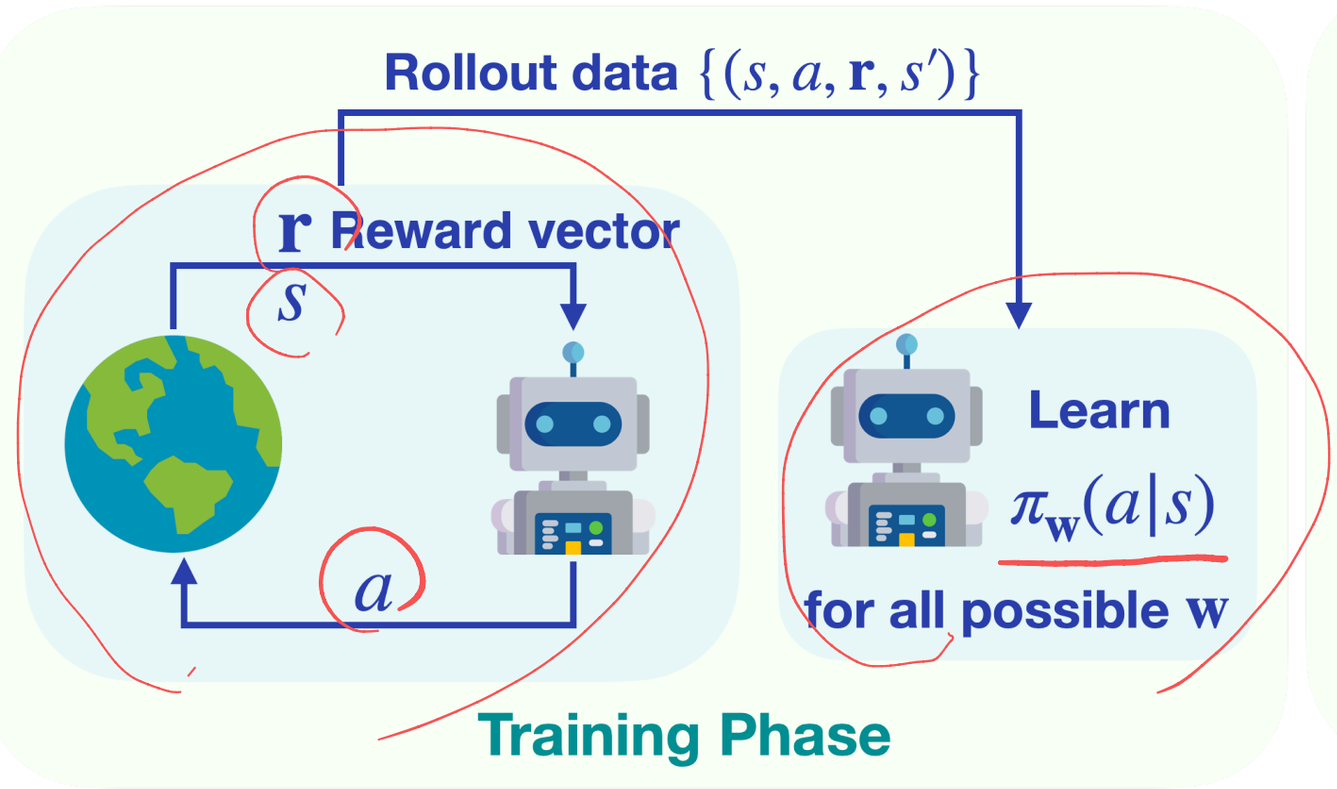


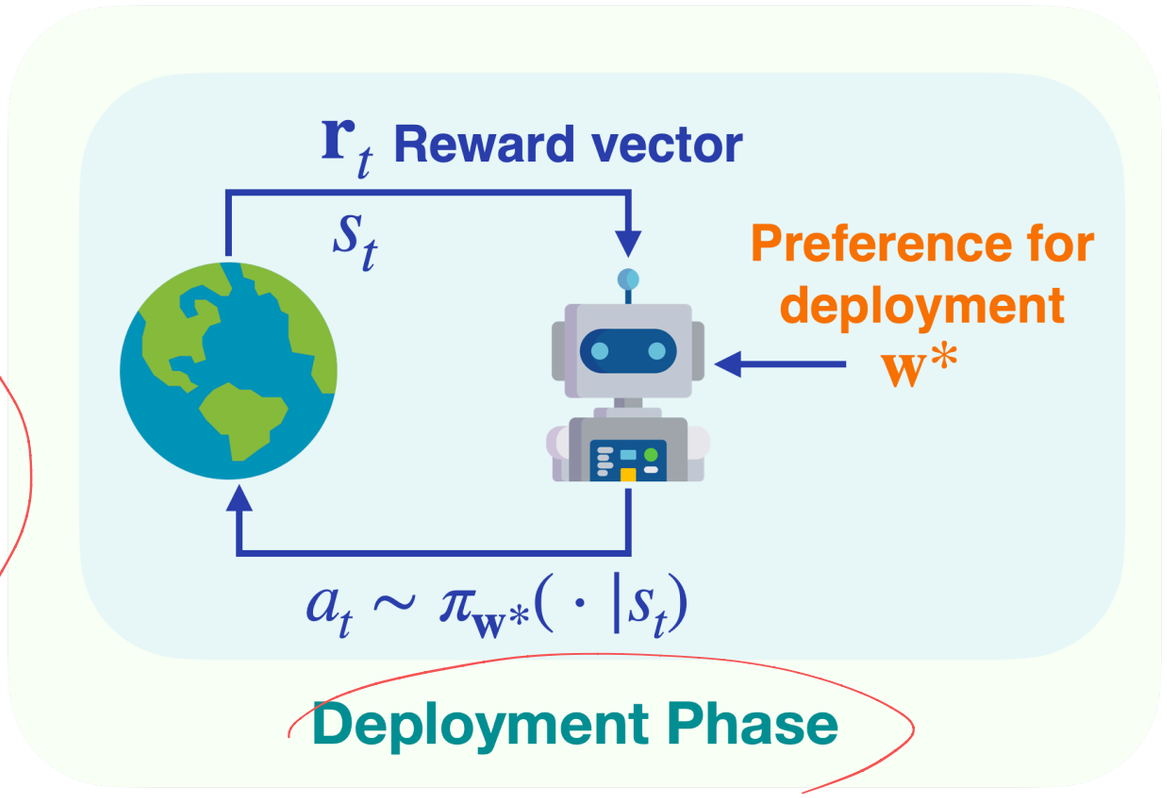
$\mathbf{w} = [0.9,0.1]$

(favor speed)

Sprint at all cost!

MORL Formulation in One Page





$$\mathbf{Q}^{\pi_{\mathbf{w}}}(s, a) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}_t \,\middle|\, s_0 = s, a_0 = a; \pi_{\mathbf{w}}\right] \quad \text{(Q-Function in MORL)}$$

$$\mathbf{J}(\pi_{\mathbf{w}}) := \mathbb{E}_{s_0, a_0 \sim \pi_{\mathbf{w}}}\left[\mathbf{Q}^{\pi_{\mathbf{w}}}(s_0, a_0)\right] \quad \text{(Vector-Valued Total Return)}$$

Goal: Learn $\pi_{\mathbf{w}}$ that maximizes *utility* $\mathbf{w}^\top \mathbf{J}(\pi_{\mathbf{w}})$, for all possible $\mathbf{w}$

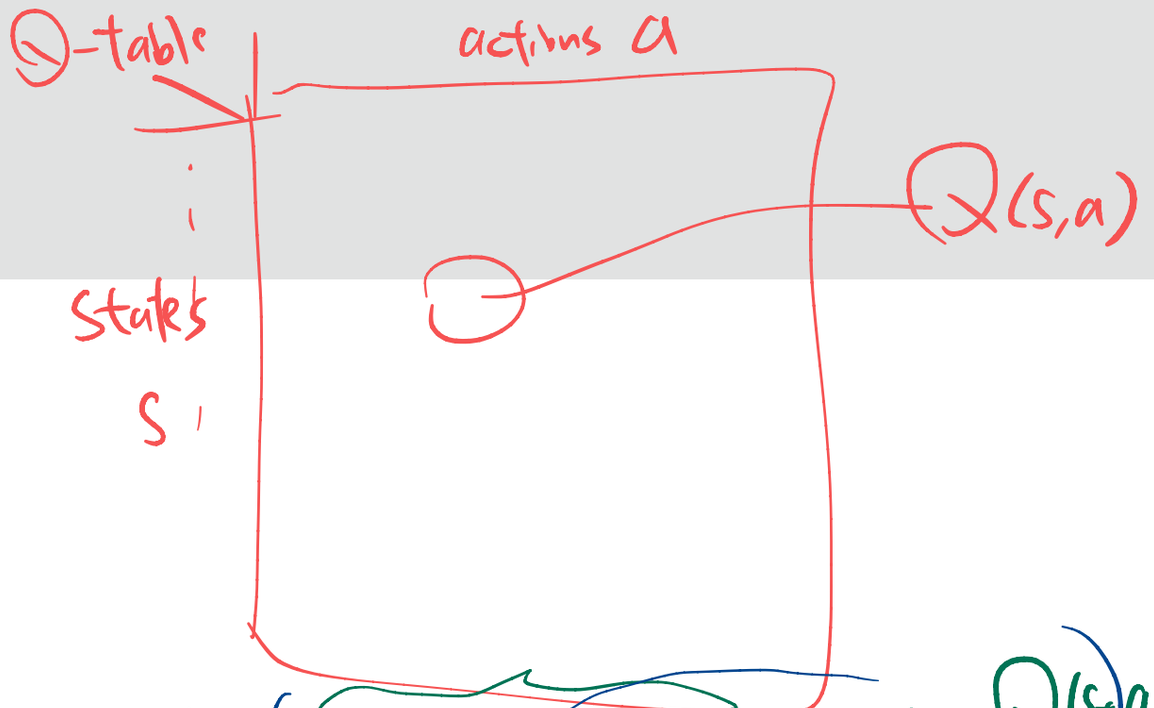
MORL is multi-task RL (but with *infinitely* many tasks!)

On-Policy vs Off-Policy Methods

	Policy Optimization	Value-Based	Model-Based	Imitation-Based
On-Policy	**Exact PG** **REINFORCE (w/i baseline)** **A2C** **On-policy DAC** **TRPO** **Natural PG (NPG)** **PPO-KL & PPO-Clip** **RLHF by PPO-KL**	**Epsilon-Greedy MC** **Sarsa** **Expected Sarsa**	**Model-Predictive Control (MPC)** **PETS**	**IRL** **GAIL** **IQ-Learn**
Off-Policy	**Off-policy DPG & DDPG** **Twin Delayed DDPG (TD3)**	**Q-learning** **Double Q-learning** **DQN & DDQN** **C51 / QR-DQN / IQN** **Rainbow** **Soft Actor-Critic (SAC)**		

Quick Review

- Q-learning?



At each time t,

$$Q(S_t,a_t) \leftarrow \underline{Q(S_t,a_t)} + \underline{\left(\alpha_t(S_t,a_t)\right)}\underline{\left(r_t + \gamma \cdot \max_{a' \in A} Q(S_{t+1},a') - Q(S_t,a_t)\right)}$$

- Overestimation bias?

$$E\left[\max_{a' \in A} Q(s,a')\right] \geq \max_{a' \in A} E\left[Q(s,a')\right]$$

Double Q-Learning

Hado van Hasselt et al., Double Q Learning, NIPS 2010

How to Mitigate Overestimation Bias: Double Estimators

- Observation: Overestimation bias can occur under a greedy policy w.r.t the estimated Q function
- Idea: Avoid using "max of estimates" as "estimate of max"

  - Create 2 independent unbiased estimates $\hat{Q}_1(s,a), \hat{Q}_2(s,a)$
    - Use one estimate to select action: $a^* = \arg\max_a Q_1(s,a)$
    - Use the other estimate for evaluate a^*: $\hat{Q}_2(s,a^*)$
    - Obtain an unbiased estimate: $\mathbb{E}[\hat{Q}_2(s,a^*)] = Q(s,a^*)$

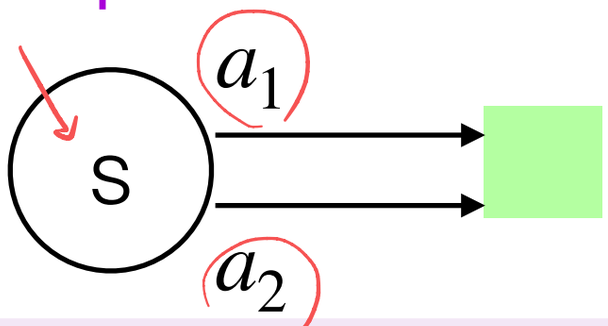
(Unfortunately, unbiased only for $Q(s,a^*)$, not for $\max_a Q(s,a)$)

Hado van Hasselt et al., Double Q Learning, NIPS 2010

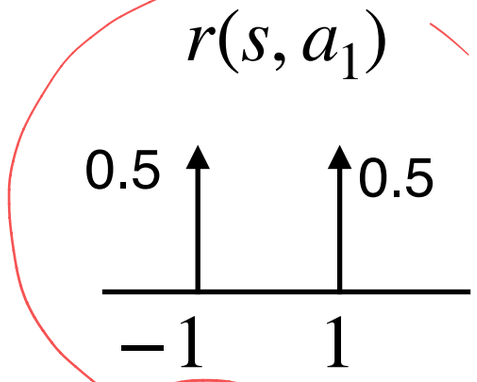
Estimation Bias of Double Estimators

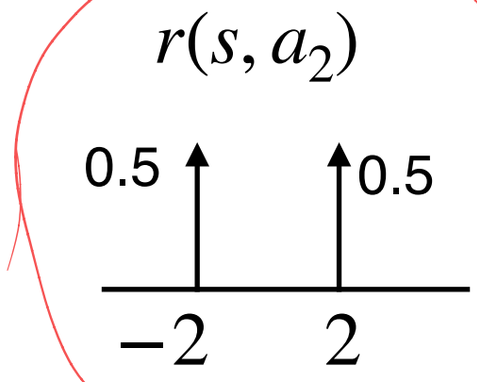
$Q^{\pi}(s,a_1)=0$, $Q^{\pi}(s,a_2)=0$

- Example: 1-state MDP with 2 actions



terminal state





- Create 2 independent unbiased estimates $\hat{Q}_A(s,a), \hat{Q}_B(s,a)$
  - Use one estimate to select action: $\bar{a} = \arg\max_a \hat{Q}_A(s,a)$
  - Use the other estimate for evaluate $\bar{a}$: $\hat{Q}_B(s,\bar{a})$
  - Obtain an unbiased estimate: $\mathbb{E}[\hat{Q}_B(s,\bar{a})] = Q(s,\bar{a})$

$$\max_a Q(s,a) = \max\left\{\mathbb{E}[\hat{Q}_{A/B}(s,a_1)], \mathbb{E}[\hat{Q}_{A/B}(s,a_2)]\right\} = 0$$

$$\mathbb{E}[\hat{Q}_B(s,\bar{a})] = P(\bar{a}=a_1)\cdot \mathbb{E}\left[\hat{Q}_B(s,\bar{a}) \mid \bar{a}=a_1\right] + P(\bar{a}=a_2)\cdot \mathbb{E}\left[\hat{Q}_B(s,\bar{a}) \mid \bar{a}=a_2\right] = 0$$

$P(\bar{a}=a_1)=\frac{1}{2}$, $\mathbb{E}[\hat{Q}_B(s,\bar{a}) \mid \bar{a}=a_1]=0$, $P(\bar{a}=a_2)=\frac{1}{2}$, $\mathbb{E}[\hat{Q}_B(s,\bar{a}) \mid \bar{a}=a_2]=0$

$$\bar{a} = \begin{cases} a_1, & \text{w.p. } \frac{1}{2} \\ a_2, & \text{w.p. } \frac{1}{2} \end{cases}$$

$\hat{Q}_A(s,a_1)$ \ $\hat{Q}_A(s,a_2)$	1	−1
2	$\bar{a}=a_2$	$\bar{a}=a_2$
−2	$\bar{a}=a_1$	$\bar{a}=a_1$

(overestimation bias is mitigated)

How to extend such ideas to general MDPs?

Double Q-Learning Algorithm (Formally)

- Double Q-Learning:

Step 1: Initialize $Q^A(s,a), Q^B(s,a)$ for all (s,a), and initial state s_0

Step 2: For each step $t = 0,1,2,\cdots$

Select a_t using ε-greedy w.r.t $Q^A(s_t,a) + Q^B(s_t,a)$

Observe (r_{t+1}, s_{t+1})

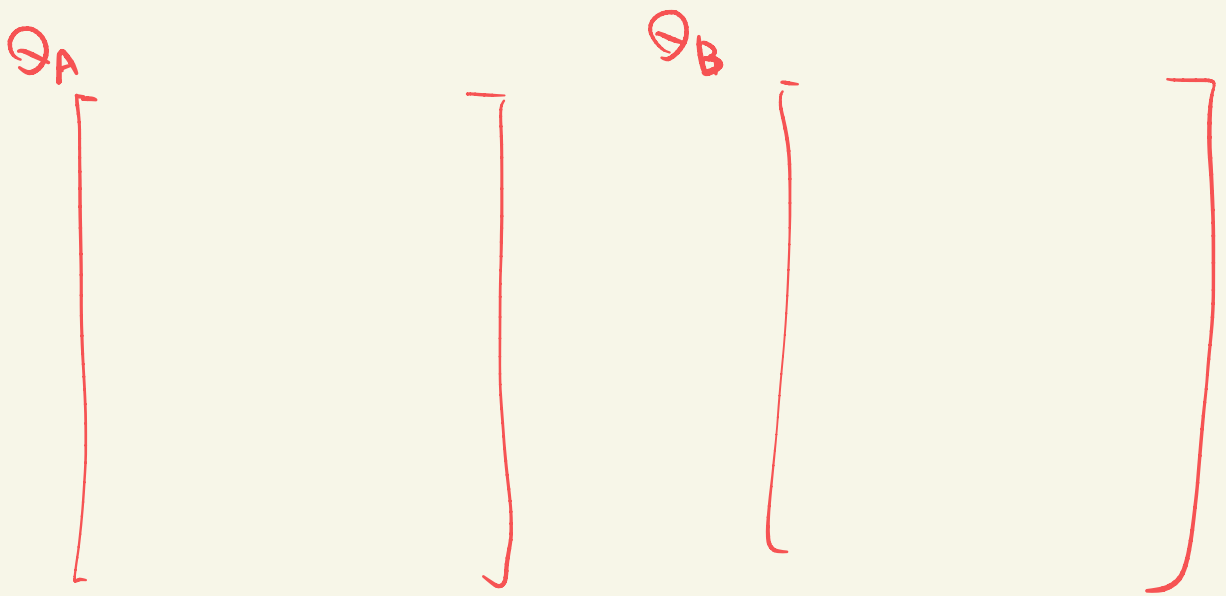
Choose one of the following updates uniformly at random

$$Q^A(s_t,a_t) \leftarrow Q^A(s_t,a_t) + \alpha\left(r_{t+1} + \gamma Q^B(s_{t+1}, \arg\max_a Q^A(s_{t+1},a)) - Q^A(s_t,a_t)\right)$$

$$Q^B(s_t,a_t) \leftarrow Q^B(s_t,a_t) + \alpha\left(r_{t+1} + \gamma Q^A(s_{t+1}, \arg\max_a Q^B(s_{t+1},a)) - Q^B(s_t,a_t)\right)$$

- Question: *Memory & computation per step (compared to Q-learning)*?

van Hasselt et al., Double Q Learning, NIPS 2010



Objective function: $\bar{a} := \underset{a'}{\text{argmax}}\ Q_A(s_{t+1}, a')$

$Q_B(s_{t+1}, \bar{a})$

$r_t + \gamma\, Q_A(s_t, \quad) = [-1, 0, 1]$

Convergence of Double Q-Learning

- Technical assumptions:

(A1) Both Q^A, Q^B receive infinite number of updates

(A2) Q^A, Q^B are stored in a lookup table, respectively

(A3) Each state-action pair is visited for an infinite number of times

(A4) Step sizes: $\sum_k \alpha_k = \infty, \sum_k \alpha_k^2 < \infty$ (Robbins-Monro conditions)

- **Convergence Result**:

  Under the assumptions (A1)-(A4), both Q^A and Q^B converge to the optimal Q function, with probability one.

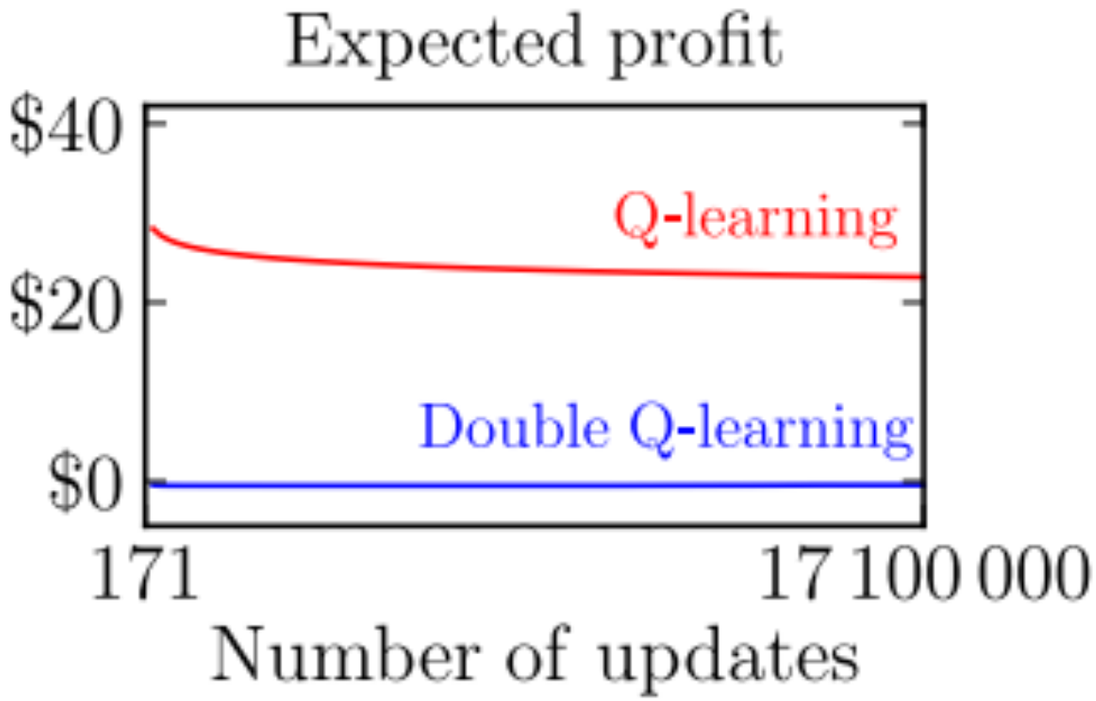
- Proof: Stochastic approximation (similar to Q-learning)

van Hasselt et al., Double Q Learning, NIPS 2010

Evaluation: Q-Learning and Double Q-Learning

OpenAI gym

- Example: Roulette with 1 state and 171 actions (assume $1 for each bet)

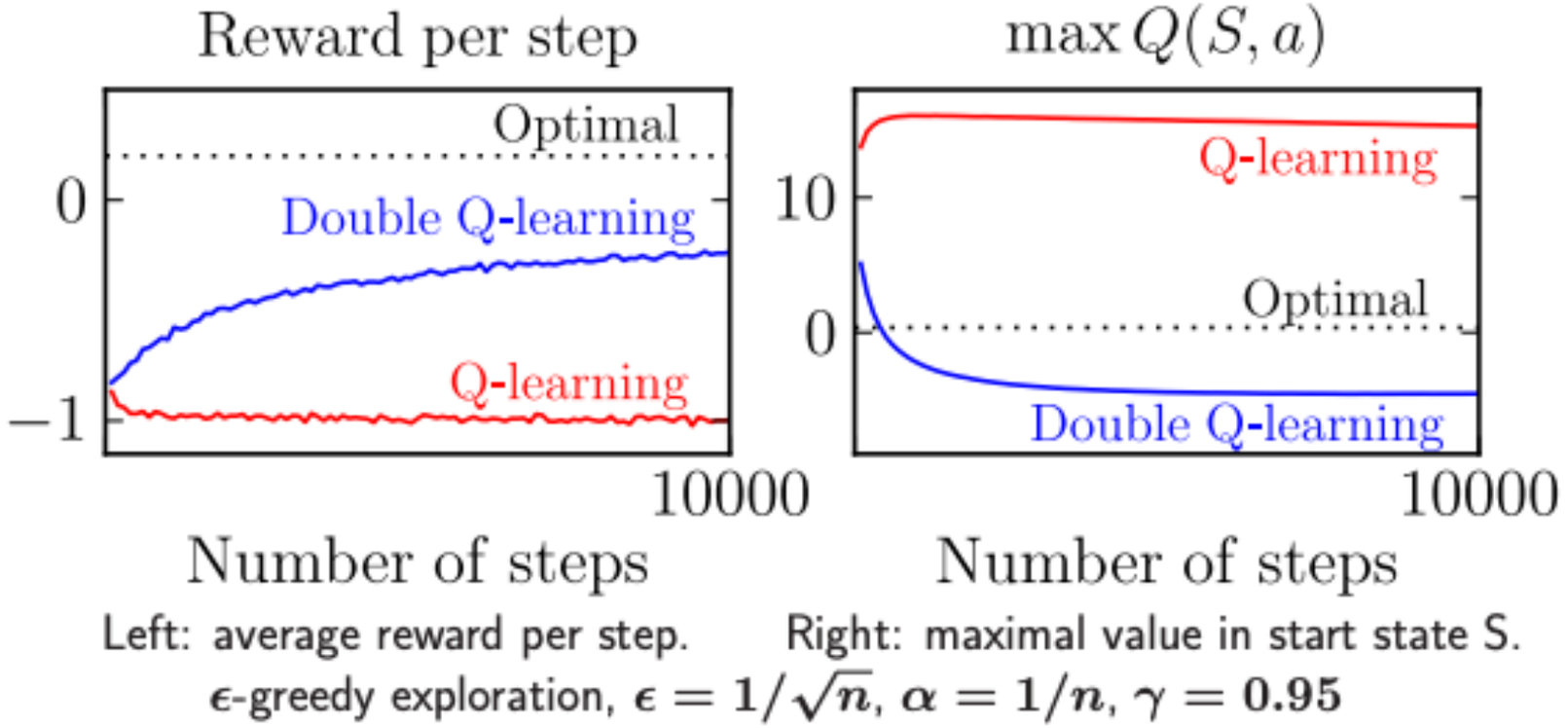


van Hasselt et al., Double Q Learning, NIPS 2010

Evaluation: Q-Learning & Double Q-Learning (Cont.)

- Example: Grid World with 9 states and 4 actions
  - Reward at the terminal state G: +5
  - Reward at any non-terminal state: -12 or +10 (random)
  - Under an optimal policy, an episode ends after 5 actions

		G
S		



Left: average reward per step. Right: maximal value in start state S.
ϵ-greedy exploration, $\epsilon = 1/\sqrt{n}$, $\alpha = 1/n$, $\gamma = 0.95$

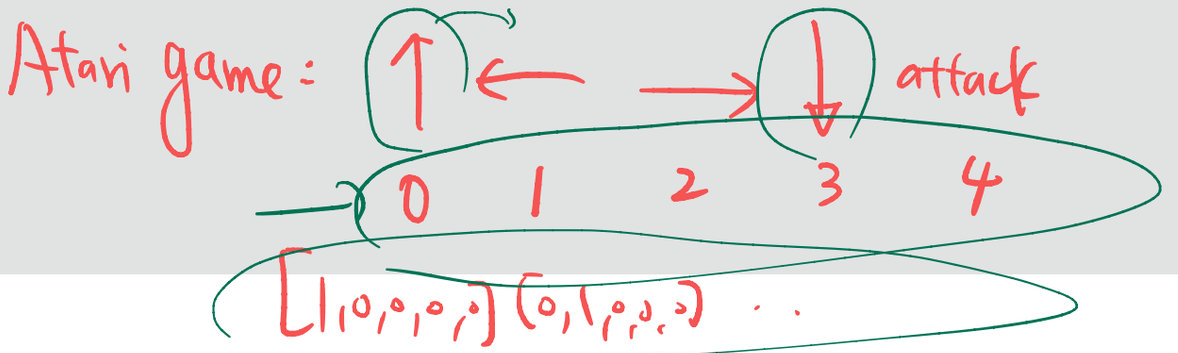
Maxmin Q-learning (ICLR 2020)

Double Q-learning may lead to "under-estimation" (with finite samples)

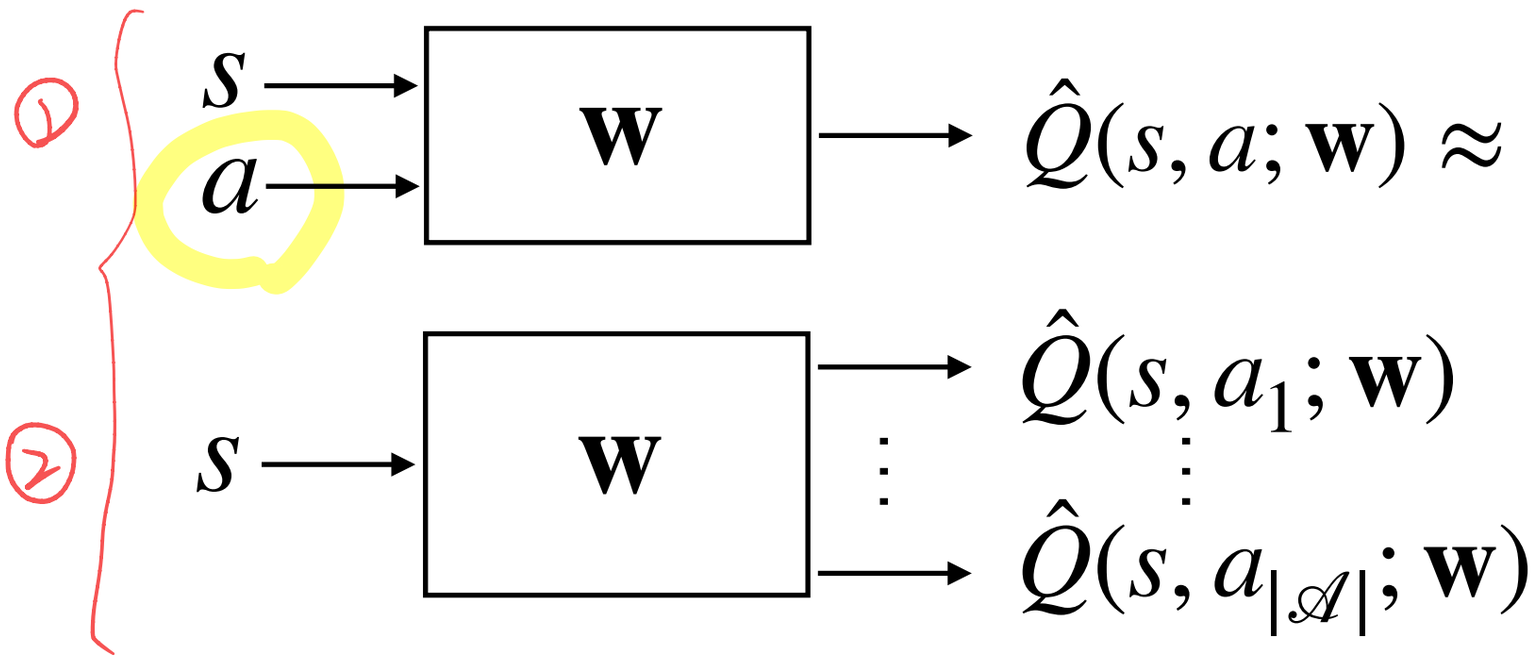
van Hasselt et al., Double Q Learning, NIPS 2010

Q-Learning With Value Function Approximation

Q-Learning With VFA



- Idea: $Q^*(s,a) \approx \hat{Q}(s,a;\mathbf{w})$ using some parametric function



- Question: Which way is preferred?
- Question: How to learn a proper $\mathbf{w}$?

Minimize MSE between estimated Q-value and a "target"

Q-Value Function Approximation

- Goal: Find $\mathbf{w}$ that minimizes Bellman error w.r.t. $\hat{Q}(s, a; \mathbf{w})$

$$\mathbf{w} = \arg\min_{\mathbf{w}'} \underbrace{\mathbb{E}_{(s,a,r,s')\sim\rho}\left[\left(r + \gamma \max_{a' \in A} \hat{Q}(s', a'; \mathbf{w}') - \hat{Q}(s, a; \mathbf{w}')\right)^2\right]}_{=:F(\mathbf{w}')}$$

- Question: If $\hat{Q}(s, a; \mathbf{w}) = Q^*(s, a)$ and $r = R(s, a)$, the loss =?

- The loss can be minimized by iterative GD / SGD update:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \nabla_{\mathbf{w}} F(\mathbf{w})$$

$$\text{(GD)} = \mathbf{w}_k + \alpha_k \mathbb{E}_{(s,a,r,s')\sim\rho}\left[2\left(r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w}_k) - \hat{Q}(s, a; \mathbf{w}_k)\right) \nabla_{\mathbf{w}} \hat{Q}(s, a; \mathbf{w}_k)\right]$$

$$\text{(SGD)} \approx \mathbf{w}_k + \alpha_k \sum_{(s,a,r,s')\in D} \left[\left(r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w}_k) - \hat{Q}(s, a; \mathbf{w}_k)\right) \nabla_{\mathbf{w}} \hat{Q}(s, a; \mathbf{w}_k)\right]$$

- Question: How does "off-policy learning" come into play? And ρ?

Q-Learning With Value Function Approximation: A Prototypic "Online" Algorithm

- Q-Learning With Value Function Approximation:

Step 1: Initialize $\mathbf{w}$ for $Q(s, a; \mathbf{w})$ and initial state s_0

Step 2: For each step $t = 0,1,2,\cdots$

Select a_t using ε-greedy w.r.t $Q(s_t, a; \mathbf{w})$

Observe (r_{t+1}, s_{t+1})

Update $\mathbf{w}$ as follows:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_k \left[\left(r_{t+1} + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'; \mathbf{w}) - \hat{Q}(s_t, a_t; \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{Q}(s_t, a_t; \mathbf{w}) \right]$$

(Special case where $|\mathcal{D}| = 1$)

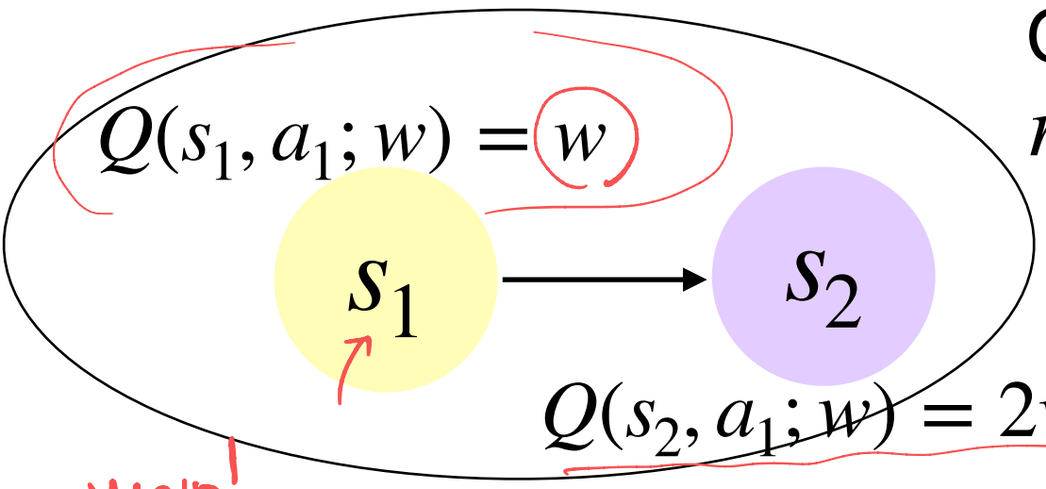
Does Q-Learning converge with function approximation?

In general, Q-learning may diverge with VFA
(even with linear VFA)

Divergence of Q-Learning With VFA

$Q(s,a;w) = \phi(s,a)^T w$

- Example: 2 states in a potentially large MDP (with linear VFA)



Only 1 action a_1 available at state s_1, and
$r(s_1, a_1) = 0, P(s_2 \mid s_1, a_1) = 1$

- Question: Given $w_k = 1, \gamma = 0.9$. Under Q-learning, what is w_{k+1}?
- Question: What will happen if we keep using the transition $s_1 \to s_2$ to update w?

$w \in \mathbb{R}^1$

$\phi(s_1, a_1) = 1, \quad \phi(s_2, a_1) = 2$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_k \left[\left(r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w}) - \hat{Q}(s, a; \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{Q}(s, a; \mathbf{w}) \right]$$

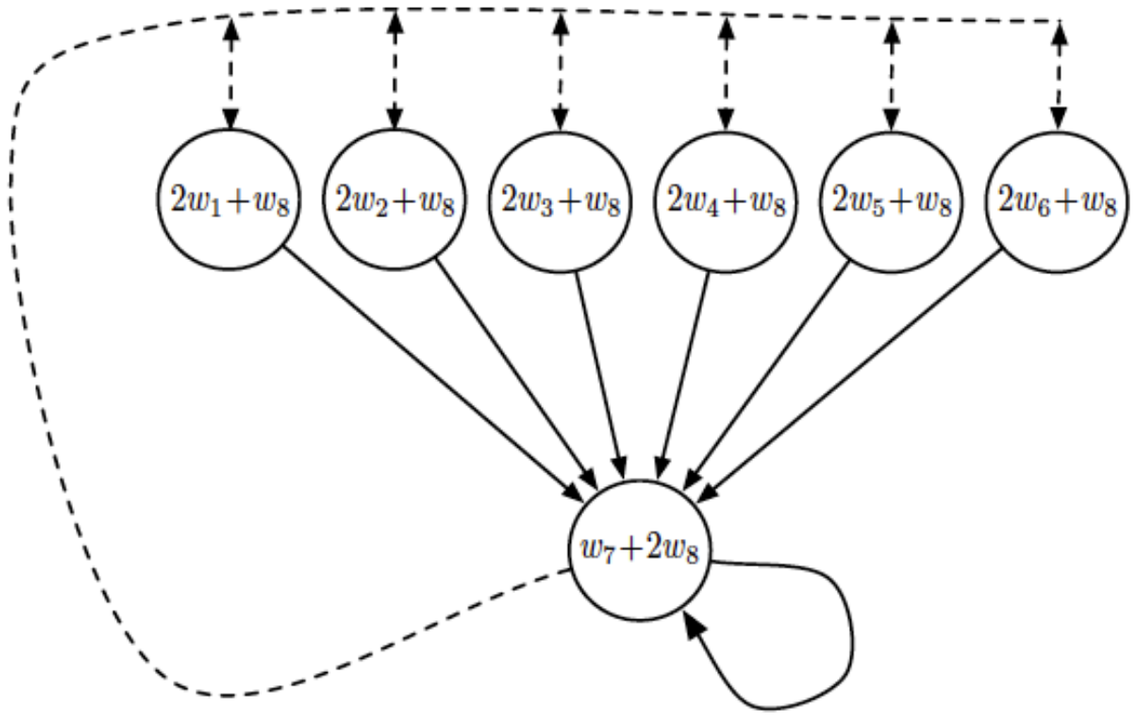
$$w_{k+1} = w_k + \alpha_k \cdot \left[\left(\underbrace{(0 + 0.9 \cdot 2w_k) - w_k}_{0.8 w_k} \right) \cdot (1) \right]$$

$$= w_k + \alpha_k \cdot 0.8 w_k$$

$$= w_k (1 + 0.8 \alpha_k)$$

Divergence of Q-Learning: A Classic Example

- Example: Baird's counterexample (with linear VFA)



$(\sqrt{J} = \|\Pi T Q_w - Q_w\|)$

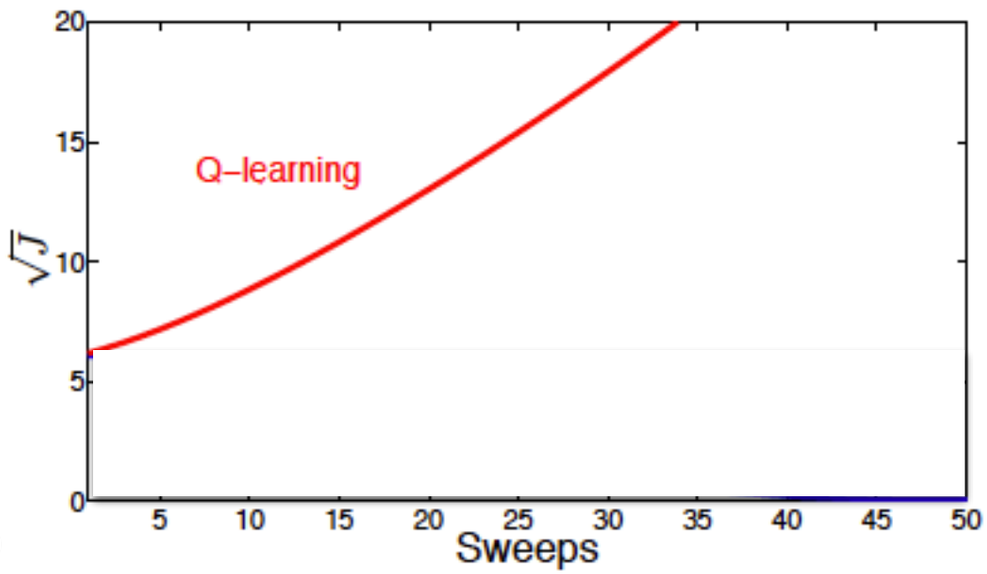
$\pi(\text{solid}|\cdot) = 1$

$b(\text{dashed}|\cdot) = 6/7$

$b(\text{solid}|\cdot) = 1/7$

$\gamma = 0.99$

$R(s,a) = 0, \forall (s,a)$



Baird, Residual Algorithms: Reinforcement Learning with Function Approximation, ICML 1995

Maei et al., Towards Off-Policy Learning Control with Function Approximation, ICML 2010

- Fortunately, Q learning usually converges in practice when target policy π is close to the behavior policy β (e.g., ε-greedy)

(Sutton and Barto, Chapter 11.2)

Why Q-Learning Diverges?

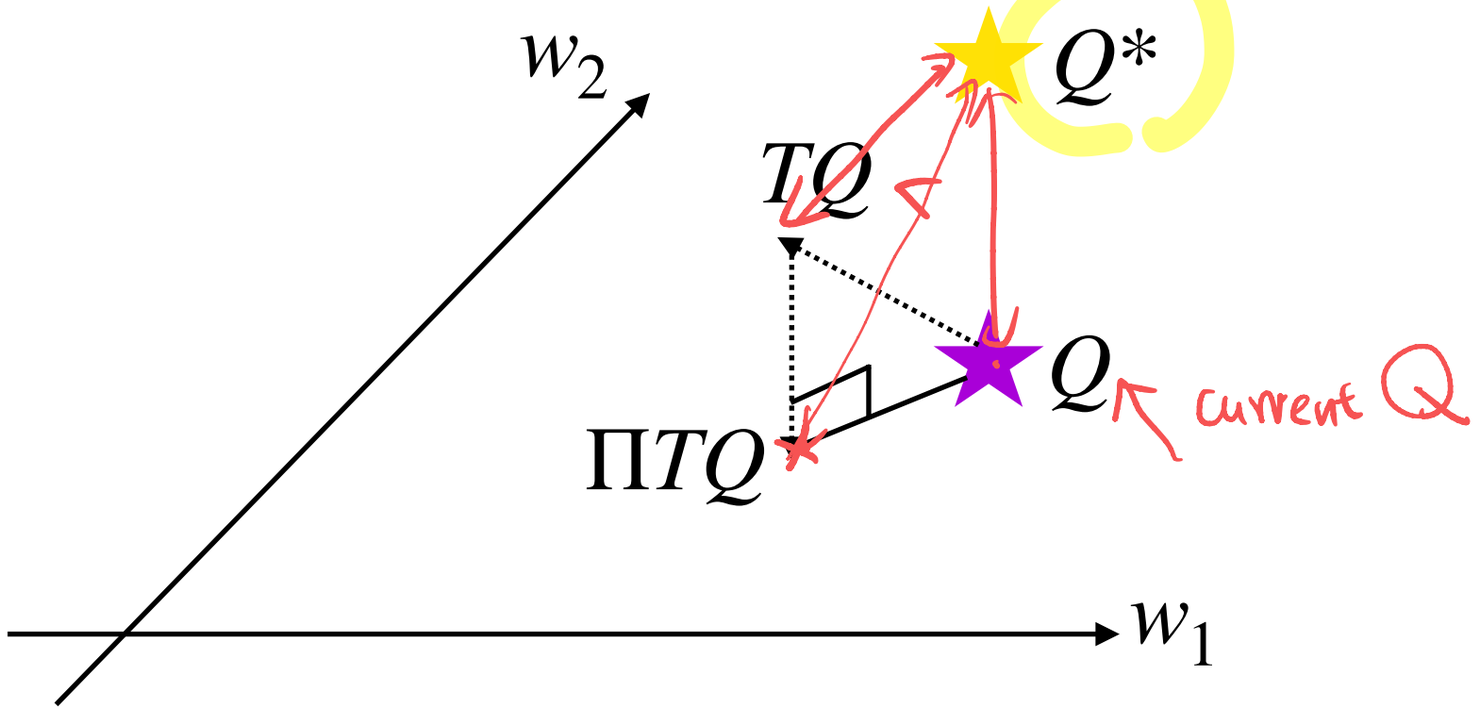
An Illustration of Divergence of Q-Learning With VFA

- **Q-learning with VFA can be equivalently decompose into two steps**:

  Step 1. Let $y_i = r(s_i, a_i) + \gamma \max_{a} Q_{\mathbf{w}}(s_i', a)$, for each i

  Step 2. Set $\mathbf{w} \leftarrow \arg\min_{\mathbf{w}'} \frac{1}{2} \sum_i \| Q_{\mathbf{w}'}(s_i, a_i) - y_i \|_2^2$

- Bellman operators are contractions, but it can become an expansion with value function approximation fitting



(Sutton and Barto, Chapter 11.4)

Deep Q-Network (DQN)

Mnih et al., Human-level control through deep reinforcement learning, Nature 2015

What is DQN?

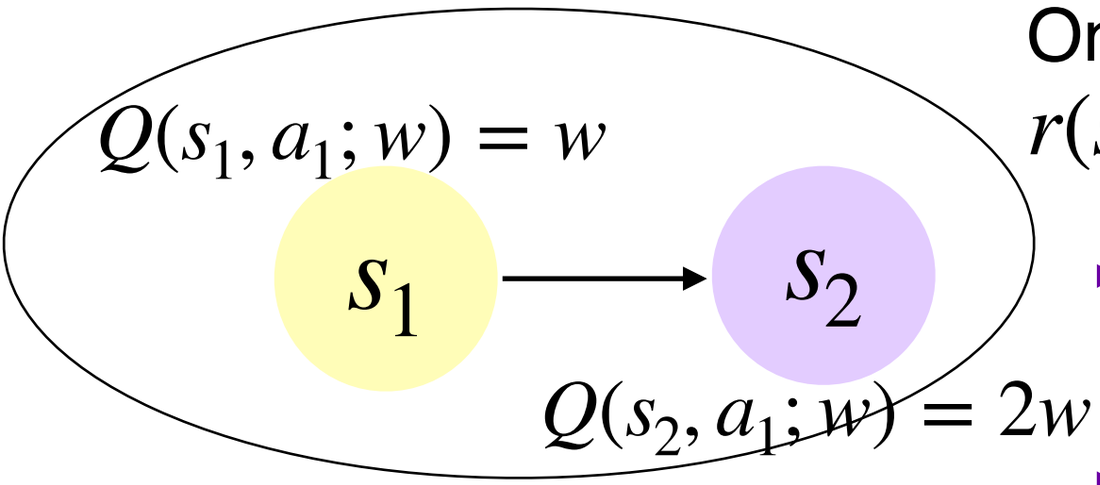
- DQN = Combine Q-Learning with NN-based nonlinear VFA
- Recall: ”Q-learning + VFA + Off-policy learning” has divergence issue
- To tackle the divergence issue, DQN applies two techniques:

  (T1) Experience replay (via a replay buffer)

  (T2) Using 2 networks: Q-network and target network

Recall: Divergence of Q-Learning With VFA

- Example: 2 states in a potentially large MDP (with linear VFA)



Only 1 action a_1 available at state s_1, and $r(s_1, a_1) = 0, P(s_2 \mid s_1, a_1) = 1$

- Question: Given $w_k = 1, \gamma = 0.9$. Under Q-learning, what is w_{k+1}?
- Question: What will happen if we keep using the transition $s_1 \rightarrow s_2$ to update w?

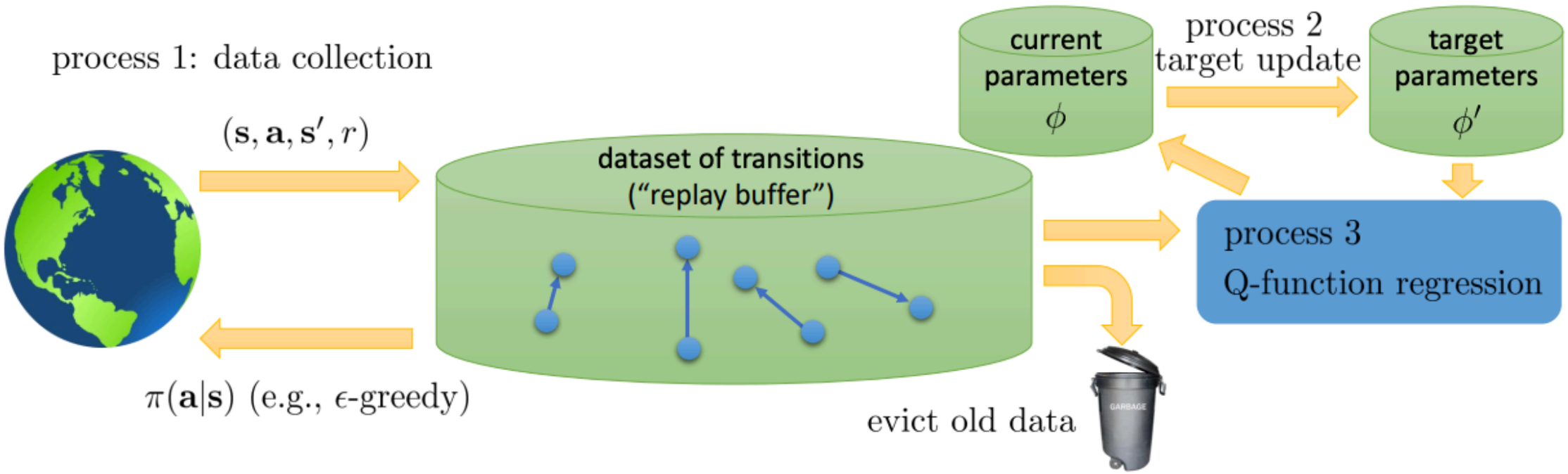
$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_k \left[\left(r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w}) - \hat{Q}(s, a; \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{Q}(s, a; \mathbf{w}) \right]$$

- Insight: Divergence can occur if the following two things happen

1. Keep using the transition $(s_1, a_1, 0, s_2)$ to update Q function ⇒ replay buffer
2. Using the latest w_k in the TD target for the update of iteration $(k + 1)$ ⇒ target network

(T1) Experience Replay

- Idea: 1. Store the previous experiences (s, a, s', r) into a buffer
  2. Sample a mini-batch from the buffer at each update
  (similar to mini-batch SGD in supervised learning)



- Purpose:

1. *Stable learning*: Break correlations between successive updates
2. *Data efficiency*: Reuse interactions with environment

(Figure Credit: Sergey Levine)

(T2) Target Network and Q-Network

- Idea: Use a separate target network (denoted by $Q_{\bar{\mathbf{w}}}$) that are updated only periodically
- Loss function:

$$F(\mathbf{w}) := \frac{1}{2}\mathbb{E}_{(s,a,r,s')\sim\rho}\left[\left(r + \gamma \max_{a'\in A} Q(s', a'; \bar{\mathbf{w}}) - Q(s, a; \mathbf{w})\right)^2\right]$$

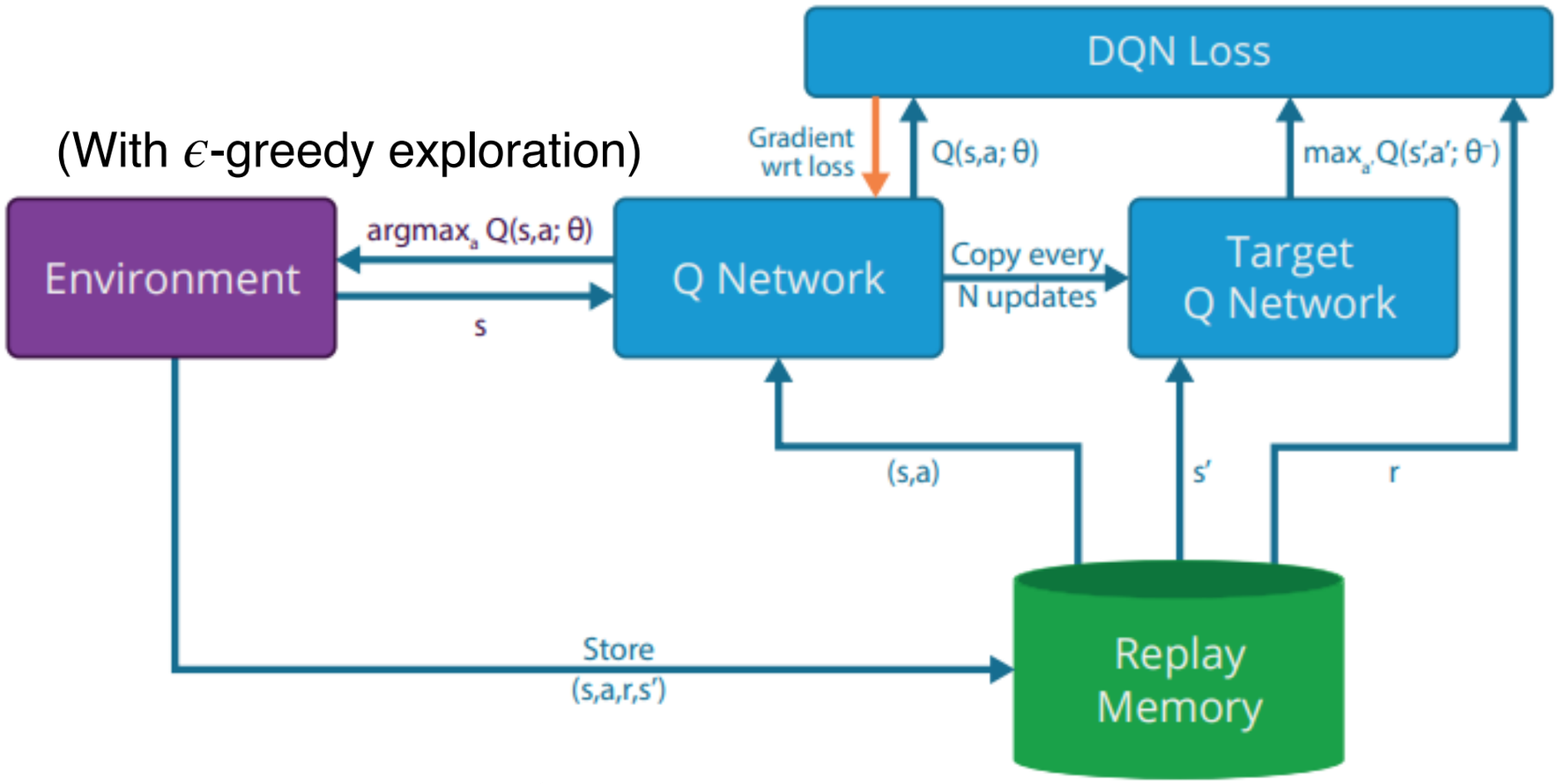
$$\approx \frac{1}{2}\sum_{(s,a,r,s')\in D}\left[\left(r + \gamma \max_{a'\in A} Q(s', a'; \bar{\mathbf{w}}) - Q(s, a; \mathbf{w})\right)^2\right]$$

- Update of the Q-network:

$$\Delta\mathbf{w}_k = \alpha_{\mathbf{w}} \sum_{(s,a,r,s')\in D} \left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \bar{\mathbf{w}})}_{\text{target}} - Q(s, a; \mathbf{w}_k)\right) \nabla_{\mathbf{w}} Q_{\mathbf{w}}(s, a; \mathbf{w}_k)$$

- Purpose: Mitigate divergence

Architecture of Vanilla DQN



$$L(\mathbf{w}) := \sum_{(s,a,r,s')\in D} \frac{1}{2}\left[\left(r + \gamma \max_{a'} Q(s', a'; \bar{\mathbf{w}}) - Q(s, a; \mathbf{w})\right)^2\right]$$

$$\nabla_{\mathbf{w}} L(\mathbf{w})\Big|_{\mathbf{w}=\mathbf{w}_k} = \sum_{(s,a,r,s')\in D} \left[\left(r + \gamma \max_{a'} Q(s', a'; \bar{\mathbf{w}}) - Q(s, a; \mathbf{w}_k)\right) \nabla_{\mathbf{w}} Q(s, a; \mathbf{w}_k)\right]$$

- Question: How to sample from the replay buffer?
- Question: How to update the replay buffer?

 (Figure Source: https://kaustabpal.github.io/dqn)

Pseudo Code of DQN

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

1. Replay buffer

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

2. Target network

For episode $= 1, M$ **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1$,T **do**

With probability ε select a random action a_t
otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

3. ε-greedy exploration

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 D

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

4. Update Q-network by mini-batch SGD

(Need to handle terminal states)

Every C steps reset $\hat{Q} = Q$

5. Periodic update of the target network

End For

End For

Mnih et al., Human-level control through deep reinforcement learning, Nature 2015

Some recent results on the convergence of Q-learning with function approximation

[NeurIPS 2020]

A new convergent variant of Q-learning with linear function approximation

Diogo S. Carvalho **Francisco S. Melo** **Pedro A. Santos**
INESC-ID & Instituto Superior Técnico, University of Lisbon
Lisbon, Portugal
{diogo.s.carvalho, pedro.santos}@tecnico.ulisboa.pt
fmelo@inesc-id.pt

Abstract

In this work, we identify a novel set of conditions that ensure convergence with probability 1 of Q-learning with linear function approximation, by proposing a two time-scale variation thereof. In the faster time scale, the algorithm features an update similar to that of DQN, where the impact of bootstrapping is attenuated by using a Q-value estimate akin to that of the target network in DQN. The slower time-scale, in turn, can be seen as a modified target network update. We establish the convergence of our algorithm, provide an error bound and discuss our results in light of existing convergence results on reinforcement learning with function approximation. Finally, we illustrate the convergent behavior of our method in domains where standard Q-learning has previously been shown to diverge.

[NeurIPS 2023]

On the Convergence and Sample Complexity Analysis of Deep Q-Networks with ε-Greedy Exploration

Shuai Zhang **Hongkang Li**
New Jersey Institute of Technology Rensselaer Polytechnic Institute

Meng Wang **Miao Liu** **Pin-Yu Chen** **Songtao Lu**
Rensselaer Polytechnic Institute IBM Research IBM Research IBM Research

Sijia Liu **Keerthiram Murugesan** **Subhajit Chaudhury**
Michigan State University IBM Research IBM Research

Abstract

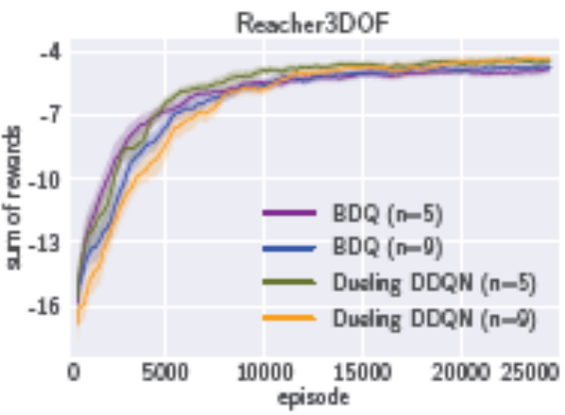
This paper provides a theoretical understanding of Deep Q-Network (DQN) with the ε-greedy exploration in deep reinforcement learning. Despite the tremendous empirical achievement of the DQN, its theoretical characterization remains underexplored. First, the exploration strategy is either impractical or ignored in the existing analysis. Second, in contrast to conventional Q-learning algorithms, the DQN employs the target network and experience replay to acquire an unbiased estimation of the mean-square Bellman error (MSBE) utilized in training the Q-network. However, the existing theoretical analysis of DQNs lacks convergence analysis or bypasses the technical challenges by deploying a significantly over-parameterized neural network, which is not computationally efficient. This paper provides the first theoretical convergence and sample complexity analysis of the practical setting of DQNs with ε-greedy policy. We prove an iterative procedure with decaying ε converges to the optimal Q-value function geometrically. Moreover, a higher level of ε values enlarges the region of convergence but slows down the convergence, while the opposite holds for a lower level of ε values. Experiments justify our established theoretical insights on DQNs.

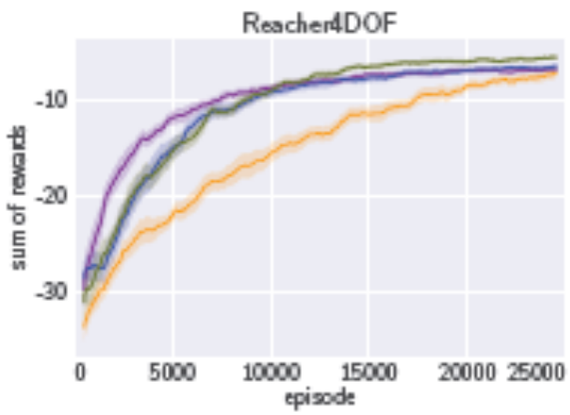
Can DQN be Applied Under Continuous Actions?

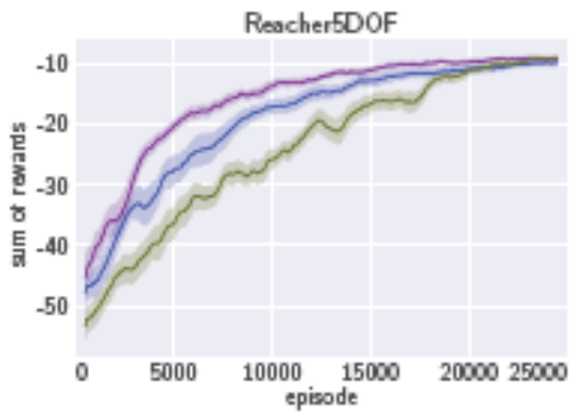
The difficulty lies in the "$\arg\max_{a \in \mathcal{A}} Q(s, a; \mathbf{w})$" operation

Existing methods that adapts DQN to continuous actions

1. Naive Action Discretization [Tavakoli et al., AAAI 2020]





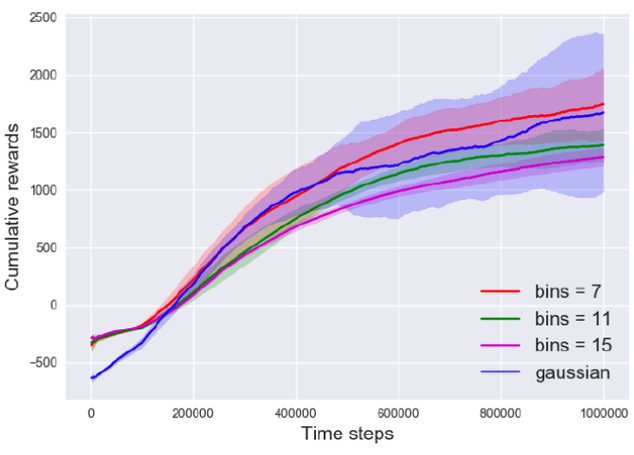


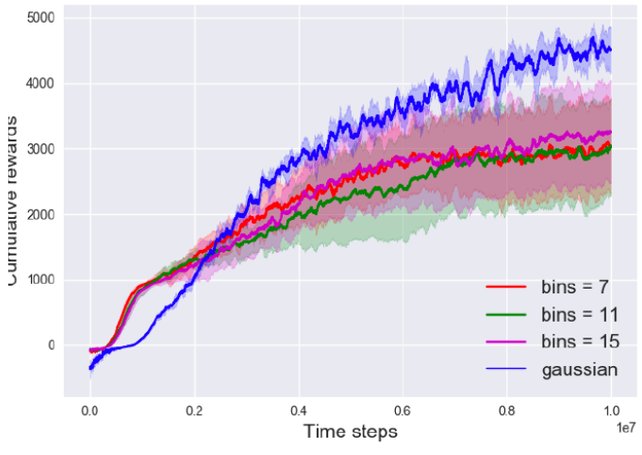
Reacher (3-5 DOF)

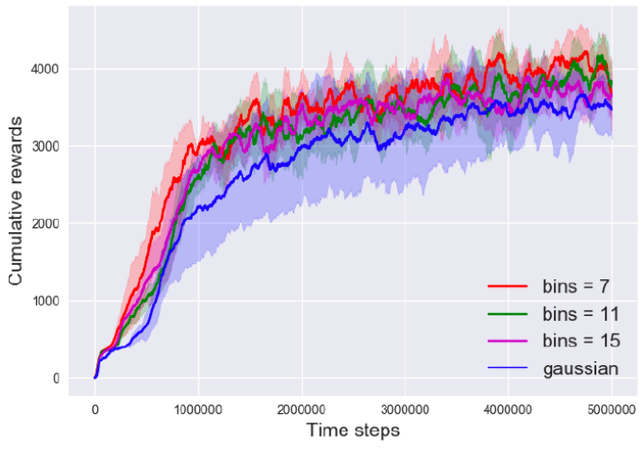
Issue: Naive discretization suffers from exponential growth of cardinality

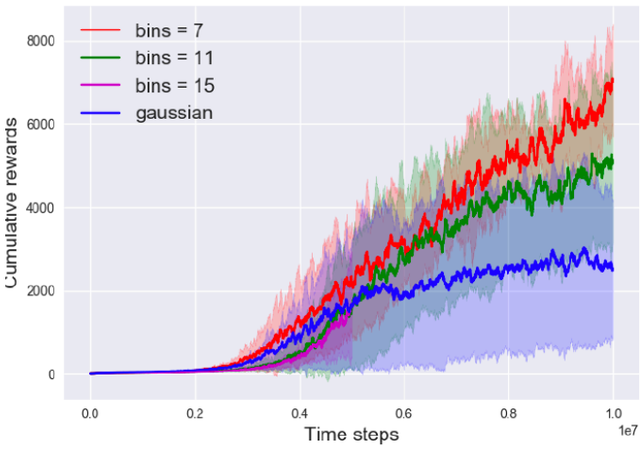
2. Discretization + Factorization [Tang and Agrawal, AAAI 2020]

$$\pi(a|s) := \Pi_{i=1}^{d} \pi_{\theta_i}(a_i|s) \quad \text{where } a = [a_0, a_1, \cdots, a_{d-1}]^\top$$





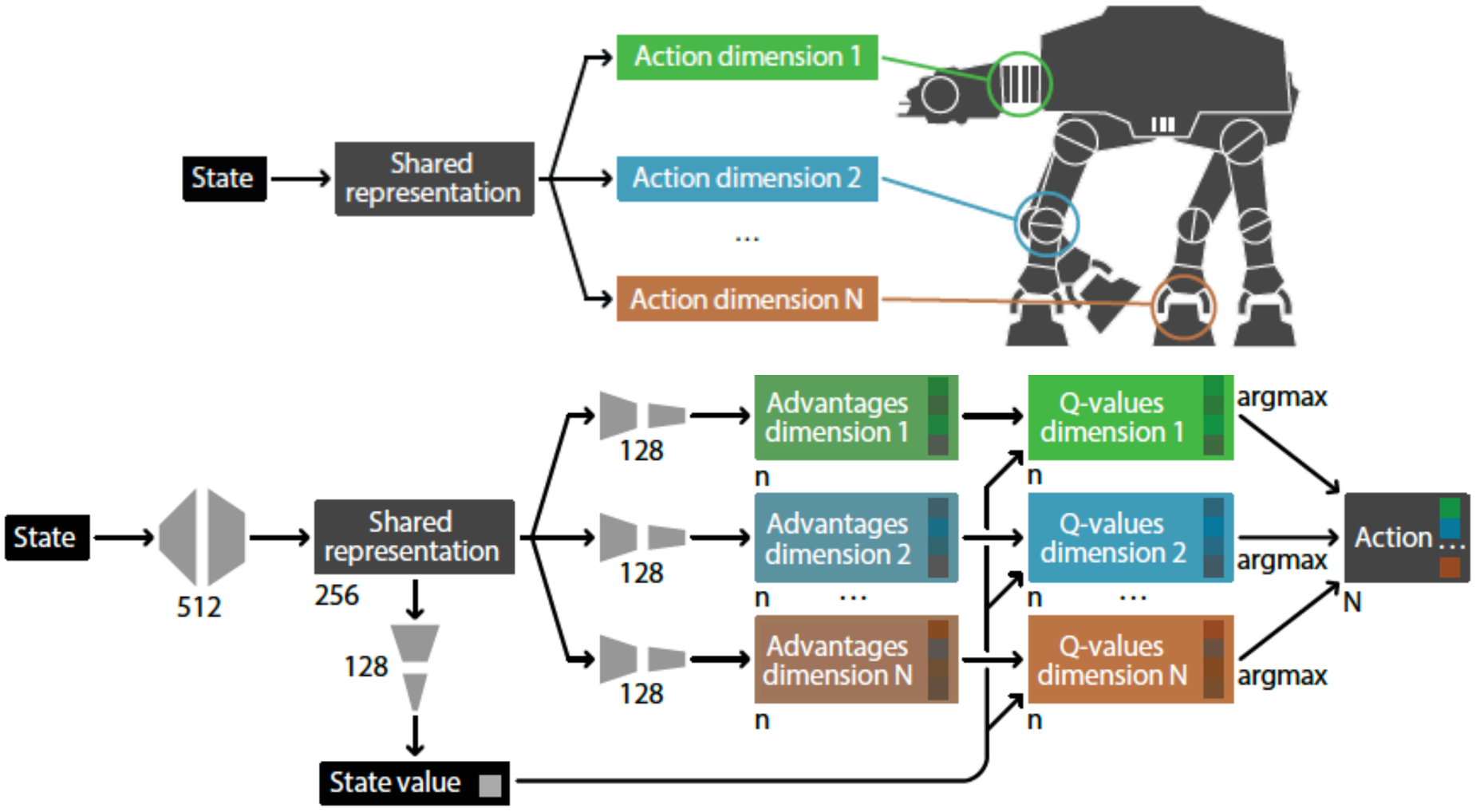


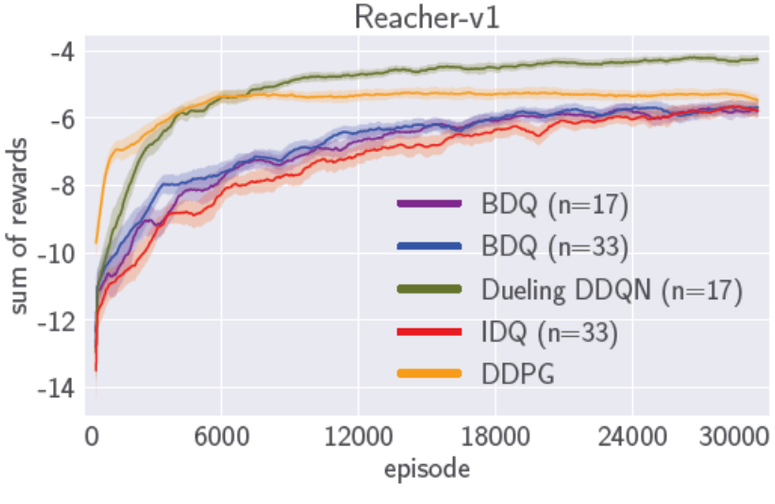


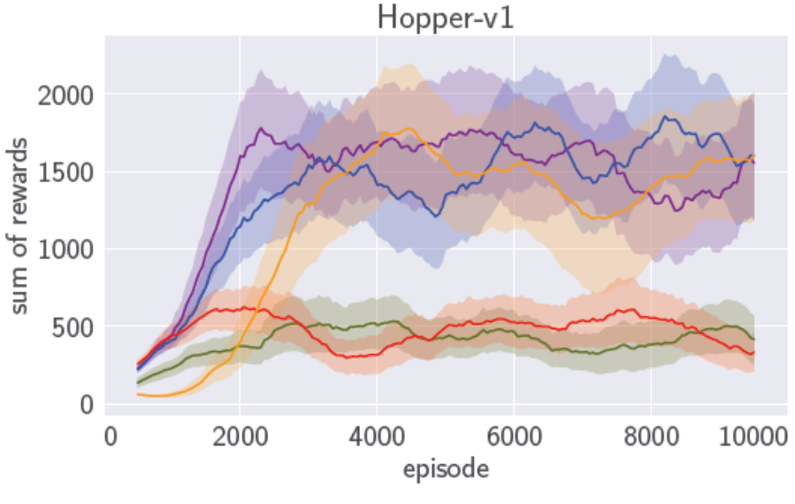
(a) **HalfCheetah + PPO** (b) **Ant + PPO** (c) **Walker + PPO** (d) **Humanoid (R) + PPO**

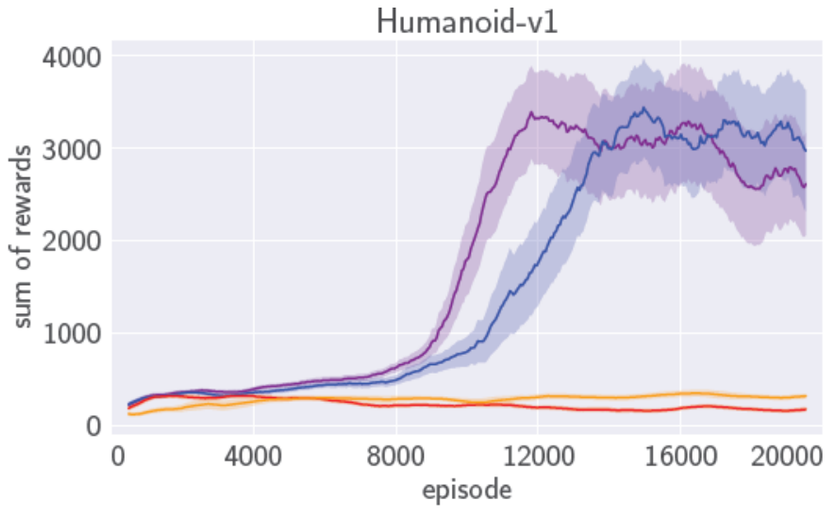
Tavakoli et al., Action Branching Architectures for Deep Reinforcement Learning, AAAI 2018
Tang and Agrawal et al., Discretizing Continuous Action Space for On-Policy Optimization, AAAI 2020

3. Discretization + Branching [Tavakoli et al., AAAI 2018]









Tavakoli et al., Action Branching Architectures for Deep Reinforcement Learning, AAAI 2018

4. Normalized Advantage Functions (NAF): Quadratic Approximation!

Continuous Deep Q-Learning with Model-based Acceleration

[Gu et al., ICML 2016]

Shixiang Gu[1 2 3] SG717@CAM.AC.UK
Timothy Lillicrap[4] COUNTZERO@GOOGLE.COM
Ilya Sutskever[3] ILYASU@GOOGLE.COM
Sergey Levine[3] SLEVINE@GOOGLE.COM
[1]University of Cambridge [2]Max Planck Institute for Intelligent Systems [3]Google Brain [4]Google DeepMind

$$Q(s,a;\phi_A,\phi_V) = A(s,a;\phi_A) + V(s;\phi_V)$$

(P is state-dependent, positive definite)

$$A(s,a;\phi_A) := -\frac{1}{2}(a-\mu(s;\phi_\mu))^\top P(s;\phi_P)(a-\mu(s;\phi_\mu))$$

$$P(s;\phi_P) := L(s|;\phi_P)L(s|;\phi_P)^\top$$ (This is known as the "Cholesky decomposition")

(L is a lower-triangular matrix)

Gu et al., Continuous Deep Q-Learning with Model-based Acceleration, ICML 2016

5. Amortized Q-Learning (AQL): Sampling!

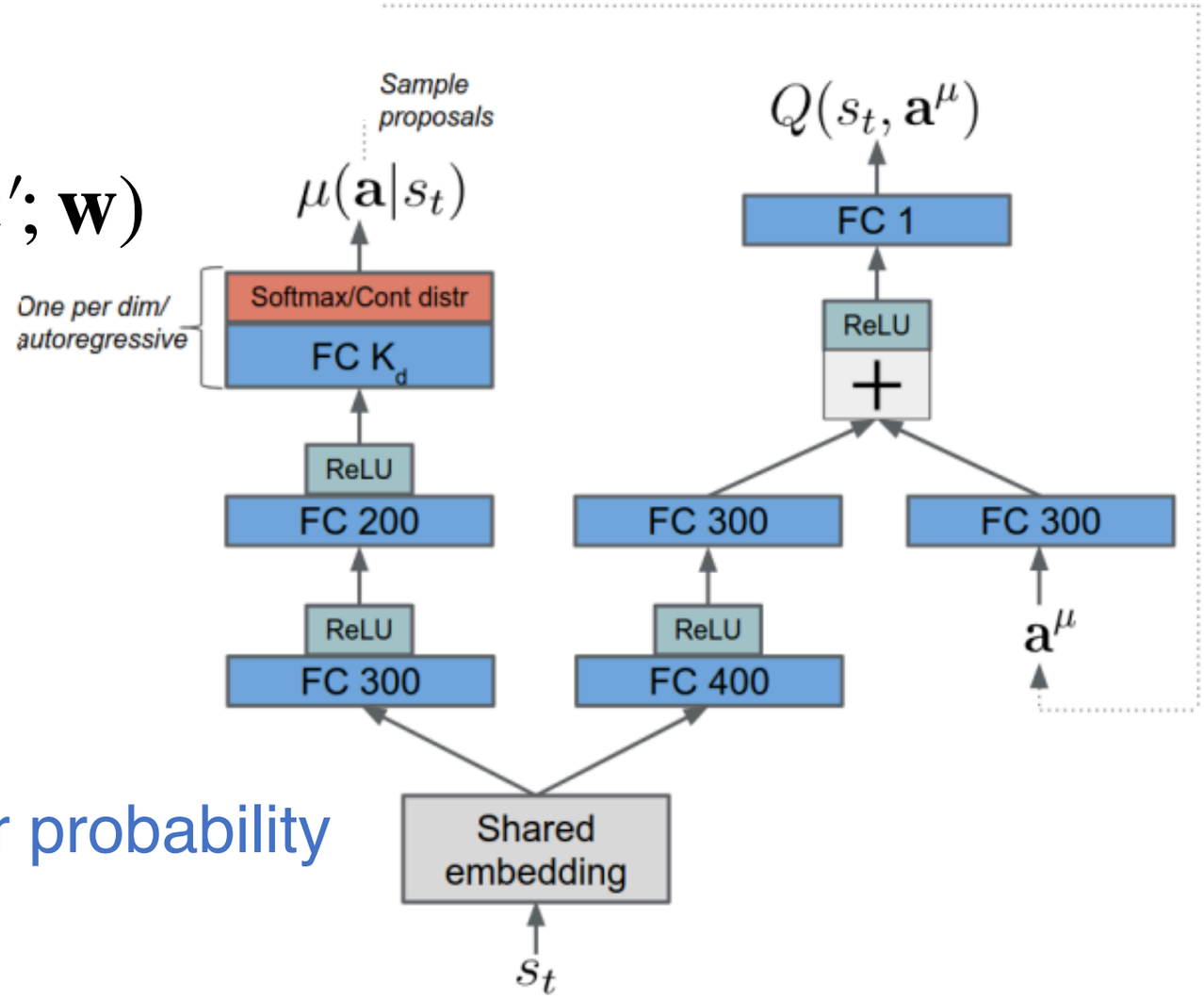
Q-LEARNING IN ENORMOUS ACTION SPACES VIA AMORTIZED APPROXIMATE MAXIMIZATION

[NeurIPS 2018 Workshop]

Tom Van de Wiele, David Warde-Farley, Andriy Mnih & Volodymyr Mnih
DeepMind
London, United Kingdom
tvdwiele@gmail.com, {dwf,amnih,vmnih}@google.com

$$\max_{a \in \mathcal{A}} Q(s, a; \mathbf{w}) \approx \max_{a' \in D \sim \mu(a)} Q(s, a'; \mathbf{w})$$

"proposal distribution"
(Learned by maximum likelihood)



Intuition: Larger $|D|$ induces a higher probability of seeing max-Q actions

Van de Wiele et al., Q-Learning in Enormous Action Spaces via Amortized Approximate Maximization, NeurIPS Workshop 2018

6. DDPG: Reinterpret DDPG as an Adaptation of DQN for Continuous Actions!

(Quick Review)

Off-Policy Deterministic PG: $\nabla_\theta J_\beta^{\pi_\theta} \approx \mathbb{E}_{s\sim d_\mu^\beta}\left[\nabla_\theta \pi_\theta(s) \nabla_a Q^{\pi_\theta}(s,a)|_{a=\pi_\theta(s)}\right]$

- Critic: estimate $Q_w \approx Q^{\pi_\theta}$ by bootstrapping
- Actor: updates policy parameters θ by deterministic policy gradient

Step 1: Initialize θ_0, w_0 and step sizes α_θ, α_w

Step 2: Sample a trajectory $\tau = (s_0, a_0, r_1, \cdots) \sim P_\mu^\beta$

For each step of the current trajectory $t = 0,1,2,\cdots$

$$\Delta w_k \leftarrow \Delta w_k + \alpha_w\left(r_t + \gamma Q_{w_k}(s_{t+1}, \pi_\theta(s_{t+1})) - Q_{w_k}(s_t, a_t)\right)\nabla_w Q_w(s_t, a_t)|_{w=w_k}$$

$$\Delta\theta_k \leftarrow \Delta\theta_k + \alpha_\theta \gamma^t\left(\nabla_\theta \pi_\theta(s_t) \nabla_a Q_{w_k}(s_t, a)|_{a=\pi_\theta(s_t)}\right)$$

$$= \nabla_\theta Q_{w_k}(s_t, \pi_\theta(s_t))|_{\theta=\theta_k}$$

Alternative Interpretation of DDPG: An Adaptation of DQN for Continuous Actions (Cont.)

- DDPG can be reinterpreted as DQN for continuous actions

1. Deterministic policy: $\pi_\theta(s) \approx \arg\max_a Q_w(s,a)$

2. How to find θ: solve $\theta \leftarrow \arg\max_\theta Q_w(s, \pi_\theta(s))$ by SGD

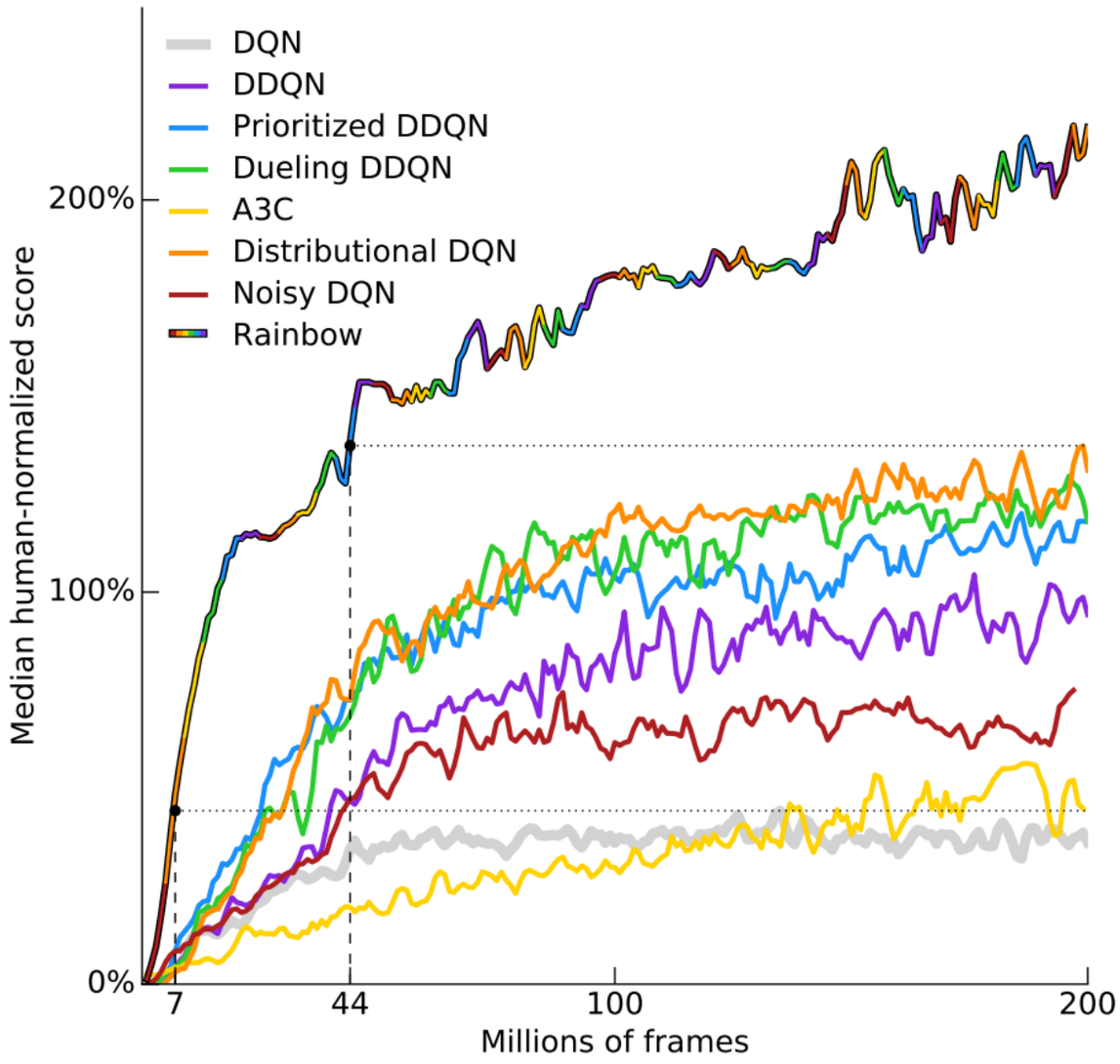
$$\Delta\theta_k \leftarrow \Delta\theta_k + \alpha_\theta \gamma^t \underbrace{\left(\nabla_\theta \pi_\theta(s_t) \nabla_a Q_{w_k}(s_t, a)|_{a=\pi_\theta(s_t)} \right)}_{= \nabla_\theta Q_{w_k}(s_t, \pi_\theta(s_t))|_{\theta=\theta_k}}$$

3. DQN and DDPG have a similar TD update scheme

$$\Delta w_k \leftarrow \Delta w_k + \alpha_w \left(r_t + \gamma Q_{w_k}(s_{t+1}, \pi_\theta(s_{t+1})) - Q_{w_k}(s_t, a_t) \right) \nabla_w Q_w(s_t, a_t)|_{w=w_k}$$

Next Topic: Rainbow (= DQN with 6 Useful Tricks)

1. Double DQN (DDQN)
2. Distributional Q-learning
3. Prioritized experience replay (PER)
4. Dueling networks
5. Multi-step return in TD target
6. Noisy networks for exploration



Hessel et al., Rainbow: Combining Improvements in Deep Reinforcement Learning, AAAI 2018

Double DQN (DDQN)

Hado van Hasselt, Arthur Guez, and David Silver,
“Deep Reinforcement Learning with Double Q-learning,” AAAI 2016

Recall: Double Q-Learning

Step 1: Initialize $Q^A(s,a), Q^B(s,a)$ for all (s,a), and initial state s_0

Step 2: For each step $t = 0,1,2,\cdots$

Select a_t using ε-greedy w.r.t $Q^A(s_t,a) + Q^B(s_t,a)$

Observe (r_{t+1}, s_{t+1})

Choose one of the following updates uniformly at random

$$Q^A(s_t,a_t) \leftarrow Q^A(s_t,a_t) + \alpha\left(r_{t+1} + \gamma Q^B(s_{t+1}, \arg\max_a Q^A(s_{t+1},a)) - Q^A(s_t,a_t)\right)$$

$$Q^B(s_t,a_t) \leftarrow Q^B(s_t,a_t) + \alpha\left(r_{t+1} + \gamma Q^A(s_{t+1}, \arg\max_a Q^B(s_{t+1},a)) - Q^B(s_t,a_t)\right)$$

- Key Idea: Decouple “Q value” and “greedy action selection”
- Question: How to apply this to DQN?

Double DQN

- Loss function of DQN:

$$F(\mathbf{w}) := \frac{1}{2}\mathbb{E}_{(s,a,r,s')\sim\rho}\left[\left(r + \gamma \max_{a'\in A} Q(s', a'; \bar{\mathbf{w}}) - Q(s, a; \mathbf{w})\right)^2\right]$$

$$\approx \frac{1}{2}\sum_{(s,a,r,s')\in D}\left[\left(r + \gamma \max_{a'\in A} Q(s', a'; \mathbf{w}) - Q(s, a; \mathbf{w})\right)^2\right]$$

- Loss function of Double DQN:

$$F(\mathbf{w}) := \frac{1}{2}\mathbb{E}_{(s,a,r,s')\sim\rho}\left[\left(r + \gamma Q\left(s', \arg\max_{a'\in A} Q(s, a; \mathbf{w}); \bar{\mathbf{w}}\right) - Q(s, a; \mathbf{w})\right)^2\right]$$

$$\approx \frac{1}{2}\sum_{(s,a,r,s')\sim D}\left[\left(r + \gamma Q\left(s', \arg\max_{a'\in A} Q(s, a; \mathbf{w}); \bar{\mathbf{w}}\right) - Q(s, a; \mathbf{w})\right)^2\right]$$

"*We therefore propose to evaluate the greedy policy according to the online network, but using the target network to estimate its value.*" — [van Hasselt et al., AAAI 2016]

Distributional Q-Learning

(Learn value distribution $Z(s, a)$ & use $E[Z(s, a)]$ as $Q(s, a)$ in Q-Learning)

Why Shall We Consider “Value Distributions”?

- Risky vs safe choices
  - E.g., Same expected return but different variance
- Good empirical performance (despite that the underlying root cause is not fully known)
  - C51 [Belleware et al., ICML 2017]
  - QR-DQN [Dabney et al., AAAI 2018]
  - IQN [Dabney et al., ICML 2018]
- New approaches for exploration
  - Information-directed exploration [Nikolov et al., ICLR 2019]
  - Distributional RL for efficient exploration [Mavrin et al., ICML 2019]
- Learn better critics
  - Truncated Quantile Critics (TQC) [Kuznetsov et al., ICML 2020]

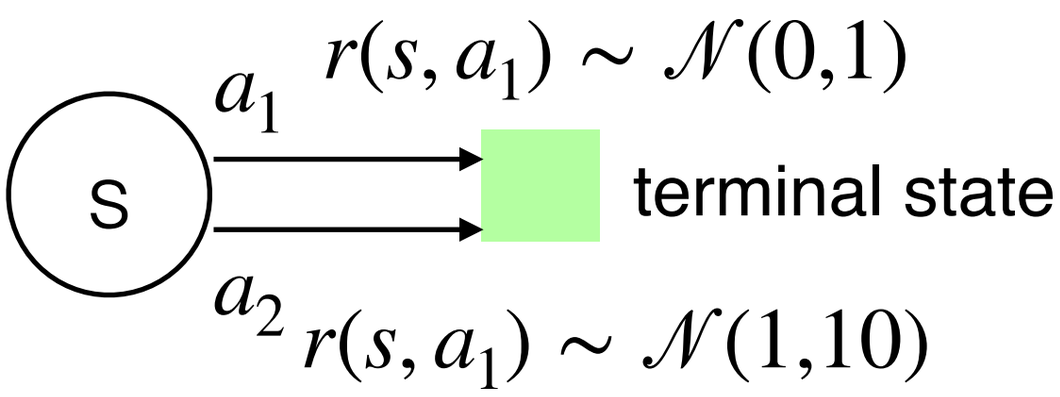
Question: How to learn the complete value distribution (instead of merely the expectation)?

Sample Action-Value $Z^{\pi}(s,a)$

- Sample action-value $Z^{\pi}(s,a)$: sample return if we start from state s and take action a, and then follow policy π

$$Q^{\pi}(s,a) = \mathbb{E}[Z^{\pi}(s,a)] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$$

- $Z^{\pi}(s,a)$ is essentially a random variable (and hence follows some distribution)
- Example: 1-state MDP with 2 actions and $\pi(s) = a_1$
  - $Q^{\pi}(s,a_1)$ = ? $Z^{\pi}(s,a_1)$?
  - $Q^{\pi}(s,a_2)$ = ? $Z^{\pi}(s,a_2)$?



Finding Z^π via Distributional Bellman Equation

- Mild assumption: $Z^\pi(s,a)$ has bounded moments
- Distributional Bellman equation for $Z^\pi(s,a)$:

$$Z^\pi(s,a) \overset{D}{=} r(s,a) + \gamma Z^\pi(s',a')$$

($\overset{D}{=}$: equal in distribution)

- Question: How to interpret this equation?
- Question: Are $r(s,a)$ and $Z^\pi(s',a')$ independent?
- Question: Is this consistent with Bellman expectation equation?