Deep Q Network
深度Q网络

朱圆恒

国科大-强化学习课程第8讲

Reminder

- Homework 1 submission deadline
  - April 4th, 8:00 am

Scaling up RL

- Generalization of RL to tackle practical problems such as self-driving cars, Atari, consumer marketing, healthcare, education
- Most of these practical problems have enormous state and/or action spaces
- It requires the representations of models/values/policies that can generalize across states and/or actions
- Solution: to represent a value function with a parameterized function instead of a lookup table

$$\hat{v}(s, \mathbf{w}) \approx v^{\pi}(s)$$
$$\hat{q}(s, a, \mathbf{w}) \approx q^{\pi}(s, a)$$

Linear Function Approximation

- Represent value function by a linear combination of features

$$\hat{v}(s, \mathbf{w}) = \mathbf{x}(s)^T \mathbf{w} = \sum_{j=1}^{n} x_j(s) w_j$$

- Objective function is $J(\mathbf{w}) = \mathbb{E}_\pi \left[(v^\pi(s) - \mathbf{x}(s)^T \mathbf{w})^2 \right]$

- Update is as simple as $\Delta \mathbf{w} = \alpha (v^\pi(s) - \hat{v}(s, \mathbf{w})) \mathbf{x}(s)$

- But there is no oracle for the true value $v^\pi(s)$, we substitute with the target from MC or TD
  - for MC policy evaluation,

$$\Delta \mathbf{w} = \alpha \Big(G_t - \hat{v}(s_t, \mathbf{w}) \Big) \mathbf{x}(s_t)$$

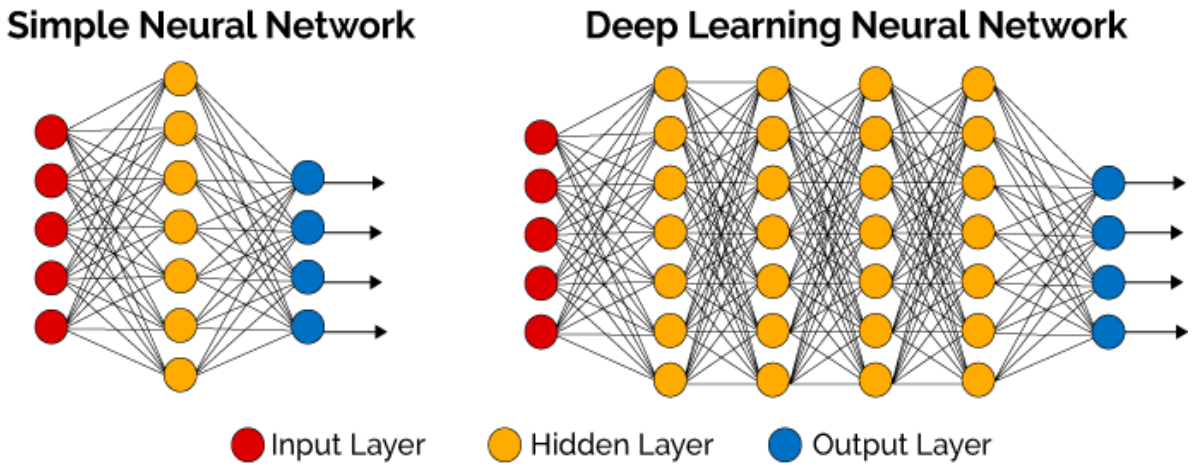
  - for TD policy evaluation,

$$\Delta \mathbf{w} = \alpha \Big(R_{t+1} + \gamma \hat{v}(s_{t+1}, \mathbf{w}) - \hat{v}(s_t, \mathbf{w}) \Big) \mathbf{x}(s_t)$$

Linear vs Nonlinear Value Function Approximation

- Linear VFA often works well given the right set of features
- But it requires manual designing of the feature set
- Alternative is to use a much richer function approximator that is able to directly learn from states without requiring the feature design
- Nonlinear function approximator: Deep neural networks

Deep Neural Networks



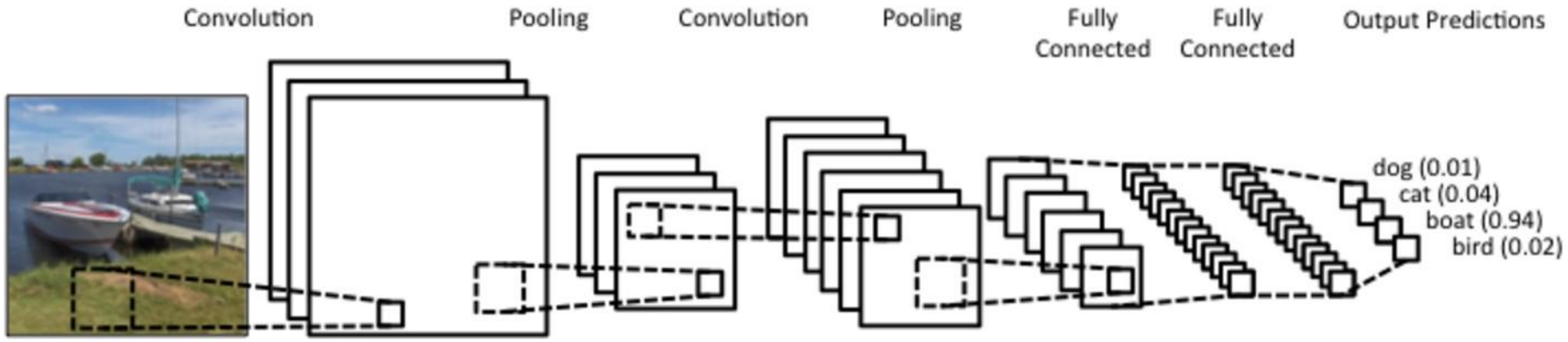
- Multiple layers of linear functions, with non-linear operators between layers

$$f(\mathbf{x};\theta) = \mathbf{W}_{L+1}^T \sigma(\mathbf{W}_L^T \sigma(...\sigma(\mathbf{W}_1^T \mathbf{x} + \mathbf{b}_1) + ... + \mathbf{b}_{L-1}) + \mathbf{b}_L) + \mathbf{b}_{L+1}$$

- The chain rule to backpropagate the gradient to update the weights using the loss function

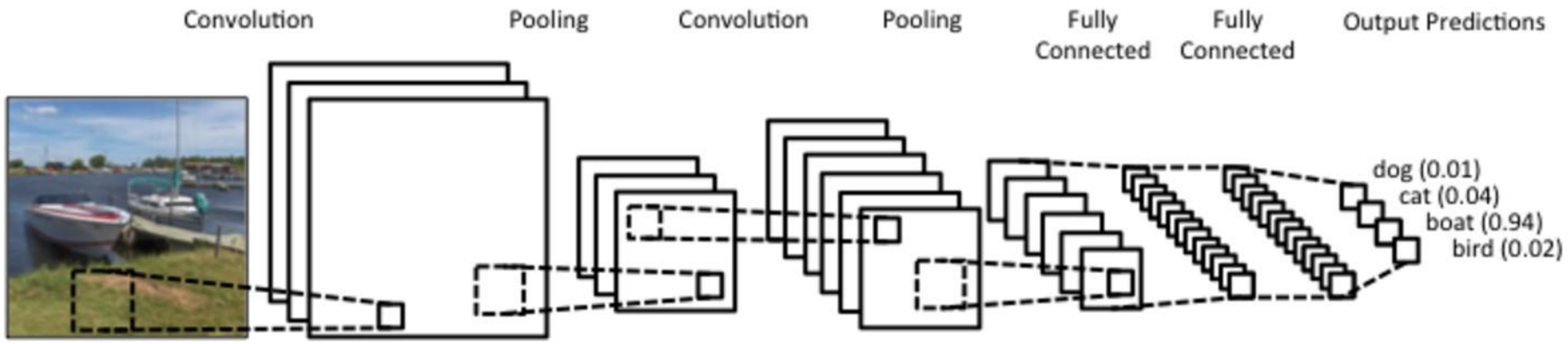
$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - f(\mathbf{x};\theta) \right)^2$$

Convolutional Neural Networks



- Convolution encodes the local information in 2D feature map
- Layers of convolution, reLU, batch normalization, pooling, etc.
- CNNs are widely used in computer vision (more than 70% top conference papers using CNNs)
- A detailed introduction on CNNs: http://cs231n.github.io/convolutional-networks/

Convolutional Neural Networks



- Consider local structure and common extraction of features
- Not fully connected
- Locality of processing
- Weight sharing for parameter reduction
- Learn the parameters of multiple convolutional filter banks
- Compress to extract salient features & favor generalization

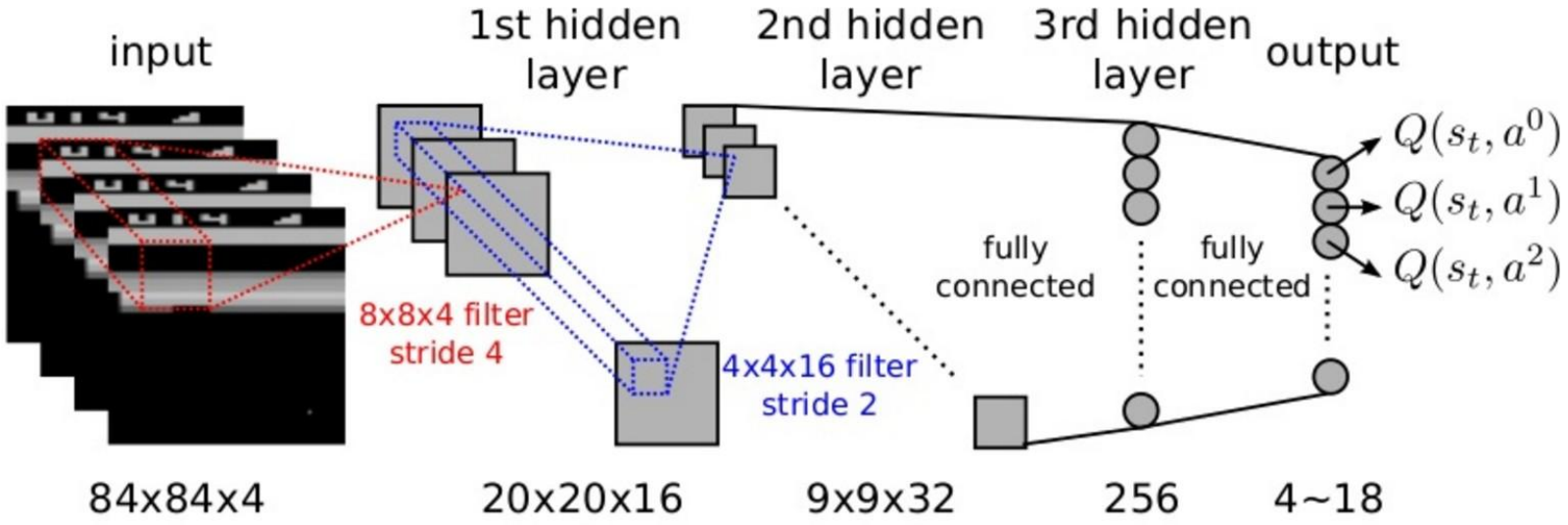
Deep Q-Networks (DQN)

- DeepMind’ s **Nature** paper: Mnih, Volodymyr; et al. (2015). **Human-level control through deep reinforcement learning**
- DQN represents the action value function with neural network approximator
- DQN reaches a professional human gaming level across many Atari games using the same network and hyperparameters

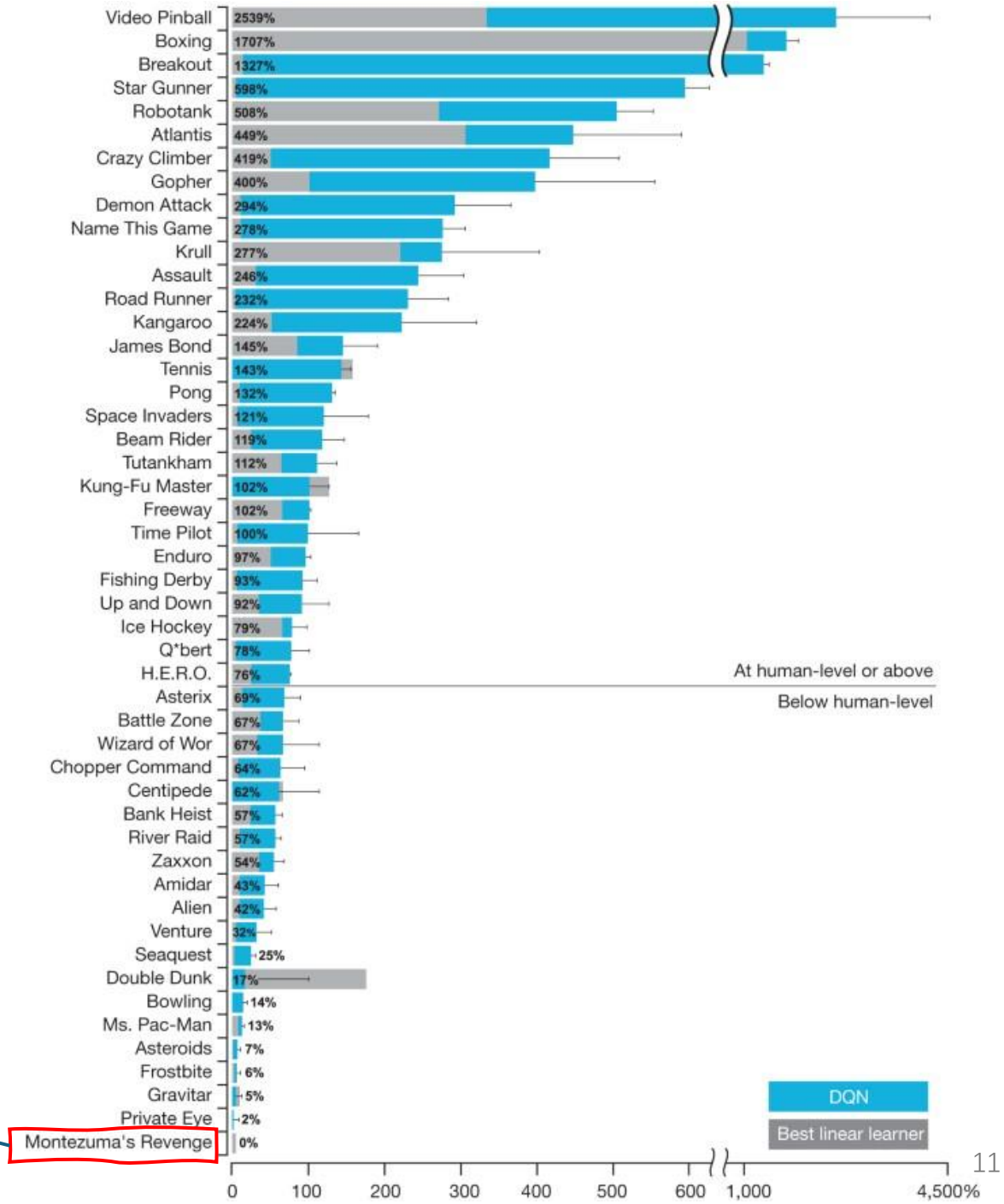
4 Atari Games: Breakout, Pong, Montezuma’s Revenge, Private Eye

DQN for Playing Atari Games

- End-to-end learning of values $Q(s, a)$ from input pixel frame
- Input state s is a stack of raw pixels from latest 4 frames
- Output of $Q(s, a)$ is 18 joystick/button positions
- Reward is the change in score for that step
- Network architecture and hyperparameters fixed across all games



Performance of DQNs on Atari



Performance of DQNs on Atari

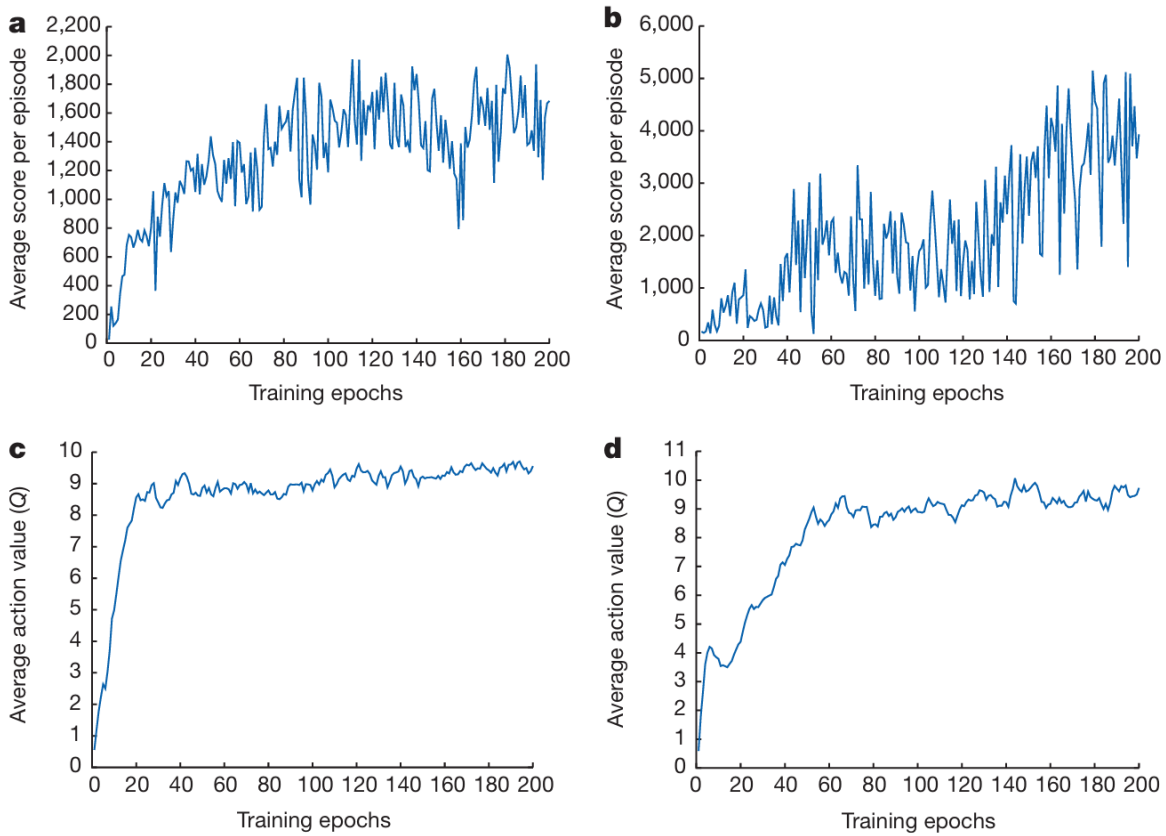
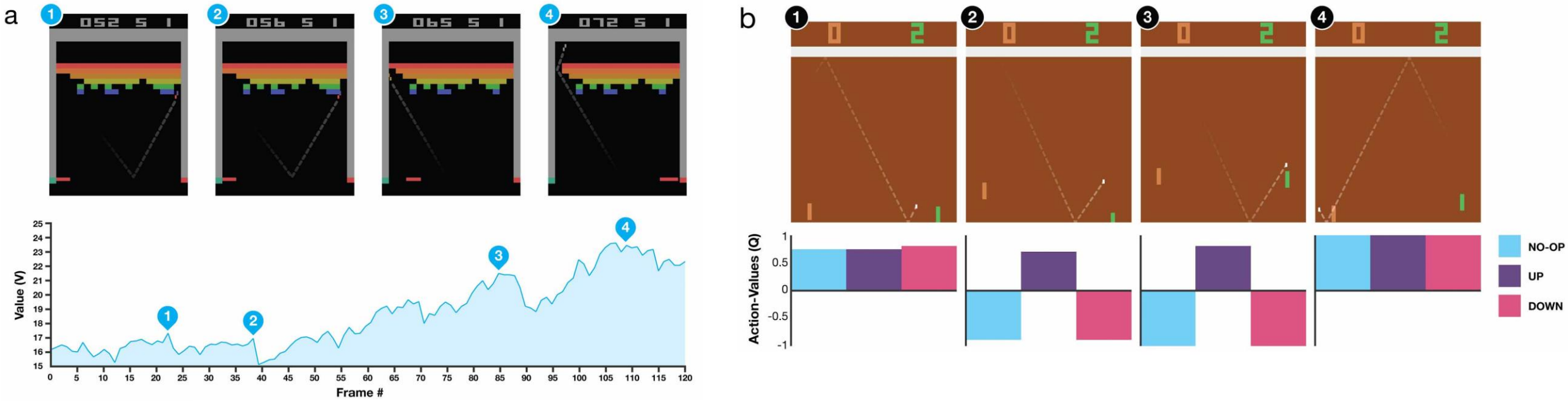


Figure 2 | Training curves tracking the agent's average score and average predicted action-value. **a**, Each point is the average score achieved per episode after the agent is run with ε-greedy policy ($\varepsilon = 0.05$) for 520 k frames on Space Invaders. **b**, Average score achieved per episode for Seaquest. **c**, Average predicted action-value on a held-out set of states on Space Invaders. Each point on the curve is the average of the action-value Q computed over the held-out set of states. Note that Q-values are scaled due to clipping of rewards (see Methods). **d**, Average predicted action-value on Seaquest. See Supplementary Discussion for details.

Human-Level Control Through Deep Reinforcement Learning, Nature2015

Performance of DQNs on Atari



Extended Data Figure 2 | Visualization of learned value functions on two games, Breakout and Pong. **a**, A visualization of the learned value function on the game Breakout. At time points 1 and 2, the state value is predicted to be ~17 and the agent is clearing the bricks at the lowest level. Each of the peaks in the value function curve corresponds to a reward obtained by clearing a brick. At time point 3, the agent is about to break through to the top level of bricks and the value increases to ~21 in anticipation of breaking out and clearing a large set of bricks. At point 4, the value is above 23 and the agent has broken through. After this point, the ball will bounce at the upper part of the bricks clearing many of them by itself. **b**, A visualization of the learned action-value function on the game Pong. At time point 1, the ball is moving towards the paddle controlled by the agent on the right side of the screen and the values of all actions are around 0.7, reflecting the expected value of this state based on previous experience. At time point 2, the agent starts moving the paddle towards the ball and the value of the 'up' action stays high while the value of the 'down' action falls to −0.9. This reflects the fact that pressing 'down' would lead to the agent losing the ball and incurring a reward of −1. At time point 3, the agent hits the ball by pressing 'up' and the expected reward keeps increasing until time point 4, when the ball reaches the left edge of the screen and the value of all actions reflects that the agent is about to receive a reward of 1. Note, the dashed line shows the past trajectory of the ball purely for illustrative purposes (that is, not shown during the game). With permission from Atari Interactive, Inc.

Human-Level Control Through Deep Reinforcement Learning, Nature2015

Summary of Deep Reinforcement Learning

- Frontier in machine learning and artificial intelligence
- Deep neural networks are used to represent
  - Value function
  - Policy function (policy gradient methods to be introduced)
  - World model
- Loss function is optimized by stochastic gradient descent (SGD)
- Challenges
  - Efficiency: too many model parameters to optimize
  - The Deadly Triad for the danger of instability and divergence in training
    - Nonlinear function approximation
    - Bootstrapping
    - Off-policy training

Action-Value Functions

Action-Value Functions $Q(s, a)$

Definition: Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma\, R_{t+1} + \gamma^2\, R_{t+2} + \gamma^3\, R_{t+3} + \cdots$

Definition: Action-value function for policy π.

- $Q_\pi(s_t, a_t) = \mathbb{E}\,[U_t | S_t = s_t, A_t = a_t]$.

  - Taken w.r.t. actions $A_{t+1}, A_{t+2}, A_{t+3}, \cdots$ and states $S_{t+1}, S_{t+2}, S_{t+3}, \cdots$
  - Integrate out everything except for the observations: $A_t = a_t$ and $S_t = s_t$.

Action-Value Functions $Q(s, a)$

Definition: Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma\, R_{t+1} + \gamma^2\, R_{t+2} + \gamma^3\, R_{t+3} + \cdots$

Definition: Action-value function for policy π.

- $Q_\pi(s_t, a_t) = \mathbb{E}\left[U_t | S_t = s_t, A_t = a_t\right].$

Definition: Optimal action-value function.

- $Q^\star(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t).$
- Whatever policy function π is used, the result of taking a_t at state s_t cannot be better than $Q^\star(s_t, a_t)$.

Deep Q-Network (DQN)

Approximate the Q Function

Goal: Win the game ($\approx$ maximize the total reward.)

Question: If we know $Q^\star(s, a)$, what is the best action?

- Obviously, the best action is $a^\star = \underset{a}{\operatorname{argmax}}\, Q^\star(s, a)$.

$Q^\star$ is an indicator of how good it is for an agent to pick action a while being in state s.

Approximate the Q Function

Goal: Win the game ($\approx$ maximize the total reward.)

Question: If we know $Q^\star(s, a)$, what is the best action?

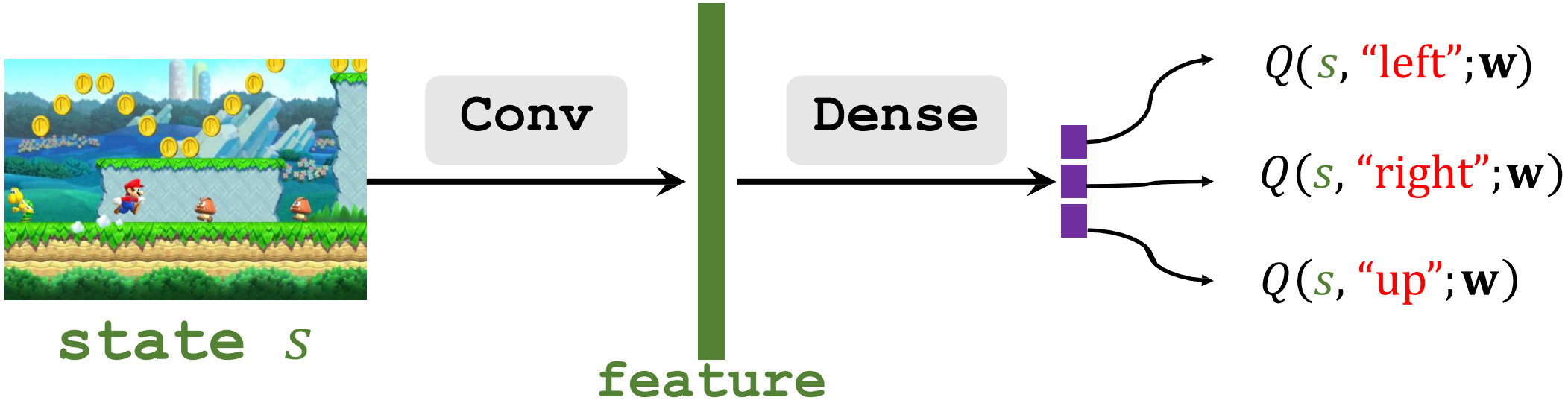
- Obviously, the best action is $a^\star = \underset{a}{\operatorname{argmax}}\, Q^\star(s, a)$.

Challenge: We do not know $Q^\star(s, a)$.

- Solution: Deep Q Network (DQN)
- Use neural network $Q(s, a; \mathbf{w})$ to approximate $Q^\star(s, a)$.

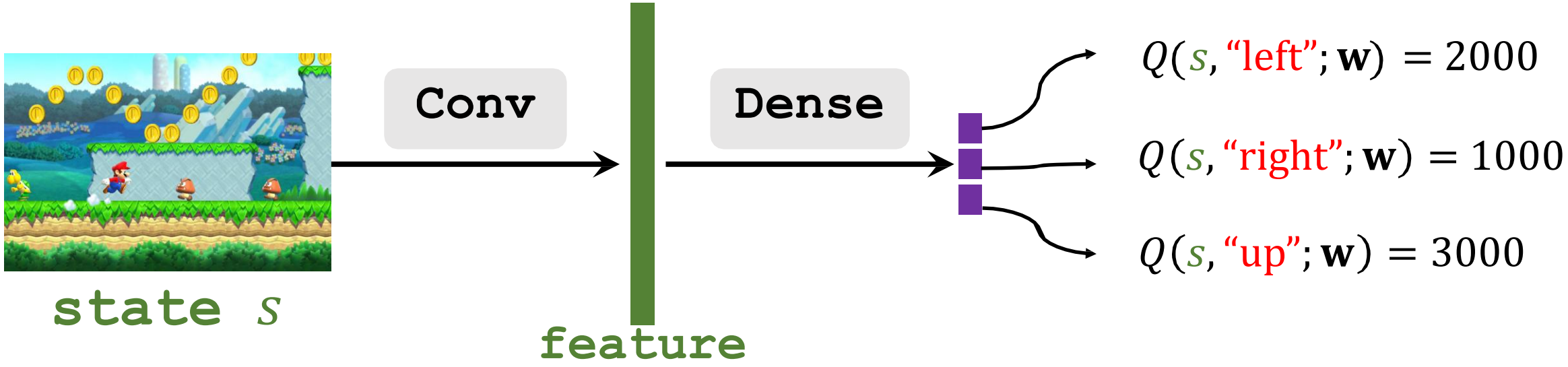
Deep Q Network (DQN)

- Input shape: size of the screenshot.
- Output shape: dimension of action space.



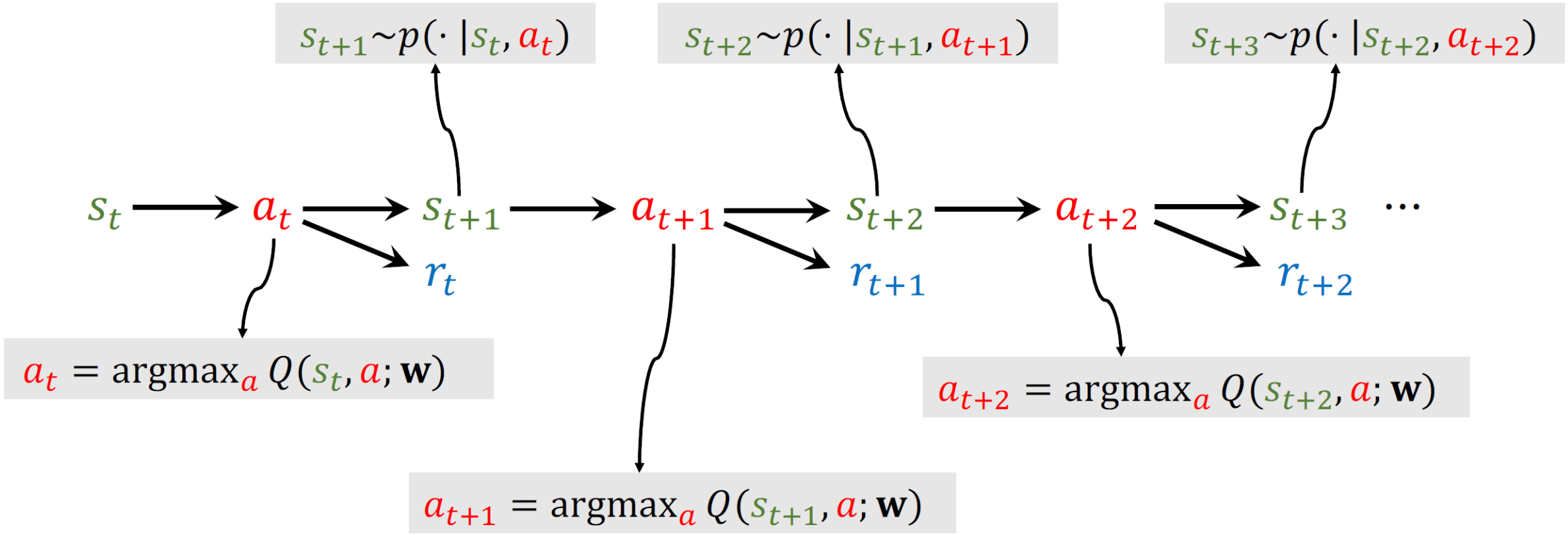
Deep Q Network (DQN)

- Input shape: size of the screenshot.
- Output shape: dimension of action space.



Question: Based on the predictions, what should be the action?

Apply DQN to Play Game



Temporal Difference (TD) Learning

Reference

1. Sutton and others: A convergent O(n) algorithm for off-policy temporal-difference learning with linear function approximation. In *NIPS*, 2008.
2. Sutton and others: Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *ICML*, 2009.

TD Learning for DQN

How to apply TD learning to DQN?

- Bellman expected equation:

$$Q(s_t, a_t) = \mathbb{E}[r_t + \gamma Q(s_{t+1}, a_{t+1})]$$

- (tabular) Temporal Difference learning:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

How to apply TD learning to DQN?

- Bellman optimal equation:

$$Q(s_t, a_t) = \mathbb{E}[r_t + \gamma \max_a Q(s_{t+1}, a)]$$

- (tabular) Q-learning:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

How to apply TD learning to DQN?

TD learning for DQN:

- DQN's output, $Q\ (s_t, a_t; \mathbf{w})$, is an estimate of U_t.
- DQN's output, $Q\ (s_{t+1}, a_{t+1}; \mathbf{w})$, is an estimate of U_{t+1}
- Thus, $Q\ (s_t, a_t; \mathbf{w}) \approx r_t + \gamma \cdot \max_a Q\ (s_{t+1}, a; \mathbf{w})$.

Prediction

TD target

Train DQN using TD learning

- Prediction: $Q\ (s_t, a_t; \mathbf{w})$
- TD target:

$$y_t = r_t + \gamma \cdot \max_a Q\ (s_{t+1}, a; \mathbf{w})$$

- Loss: $L_t = \frac{1}{2}[Q(s_t, a_t; \mathbf{w}) - y_t]^2.$
- Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial L_t}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}.$

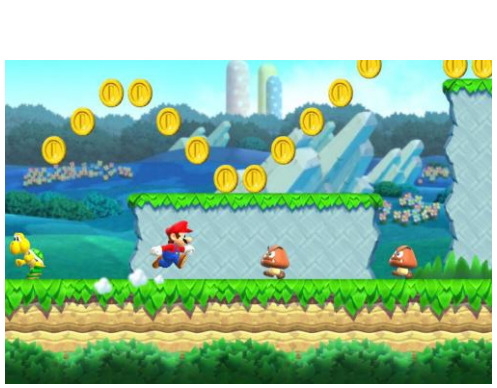
Temporal Difference (TD) Learning

Algorithm: One iteration of TD learning.

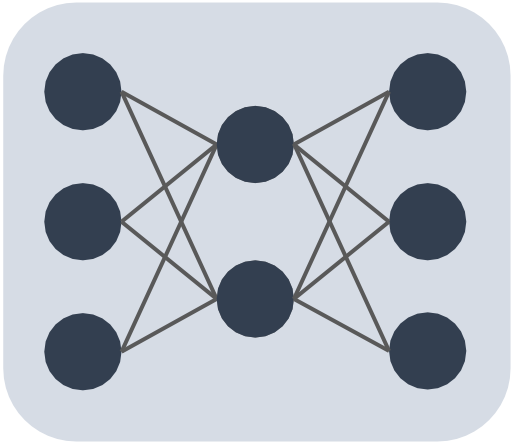
1. Observe state $S_t = s_t$ and perform action $A_t = a_t$.
2. Predict the value: $q_t = Q(s_t, a_t; \mathbf{w})$.
3. Differentiate the value network: $\mathbf{d}_t = \frac{\partial Q(s_t, a_t; \mathbf{w})}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$.
4. Environment provides new state s_{t+1} and reward r_t.
5. Compute TD target: $y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w})$.
6. Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot (q_t - y_t) \cdot \mathbf{d}_t$.

Deep Q Network (DQN)

Approximate the optimal action-value function, $Q^{\star}(s, a)$, by $Q(s, a; \mathbf{w})$.



state s



DQN

(parameterized by $\mathbf{w}$)

$Q(s, \text{“left”}; \mathbf{w})$

$Q(s, \text{“right”}; \mathbf{w})$

$Q(s, \text{“up”}; \mathbf{w})$

Temporal Difference (TD) Learning

Algorithm: One iteration of TD learning.

1. Observe state $S_t = s_t$ and perform action $A_t = a_t$.
2. Predict the value: $q_t = Q(s_t, a_t; \mathbf{w})$.
3. Differentiate the value network: $\mathbf{d}_t = \frac{\partial\, Q(s_t, a_t; \mathbf{w})}{\partial\, \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$.
4. Environment provides new state s_{t+1} and reward r_t .
5. Compute TD target: $y_t = r_t + \gamma \cdot \max_a Q\,(s_{t+1}, a; \mathbf{w})$.
6. Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot (q_t - y_t) \cdot \mathbf{d}_t$.

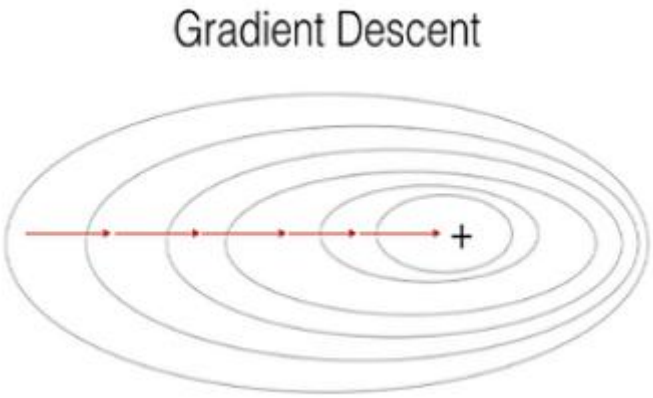
Discard (s_t, a_t, r_t, s_{t+1}) after using it.

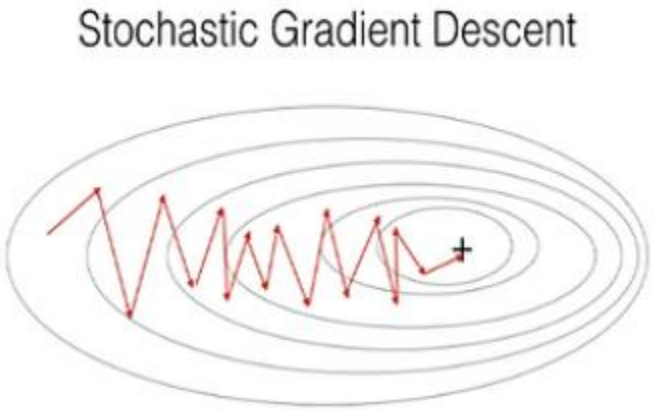
Shortcoming **1: Waste of** Experience

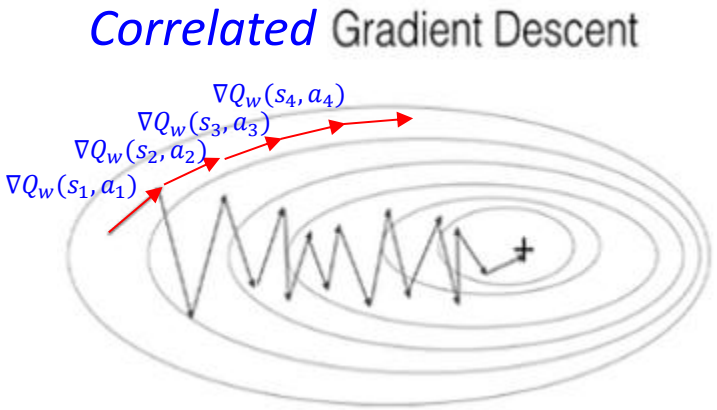
- **A transition:** (s_t, a_t, r_t, s_{t+1}) .
- **Experience:** all the transitions, for $t = 1, 2, \cdots$.
- Previously, we discard (s_t, a_t, r_t, s_{t+1}) after using it.
- It is a waste…

Shortcoming 2: Correlated Updates

- Previously, we use (s_t, a_t, r_t, s_{t+1}) sequentially, for $t = 1, 2, \cdots$, to update $\mathbf{w}$.
- Consecutive states, s_t and s_{t+1} ,are strongly correlated (which is bad.)





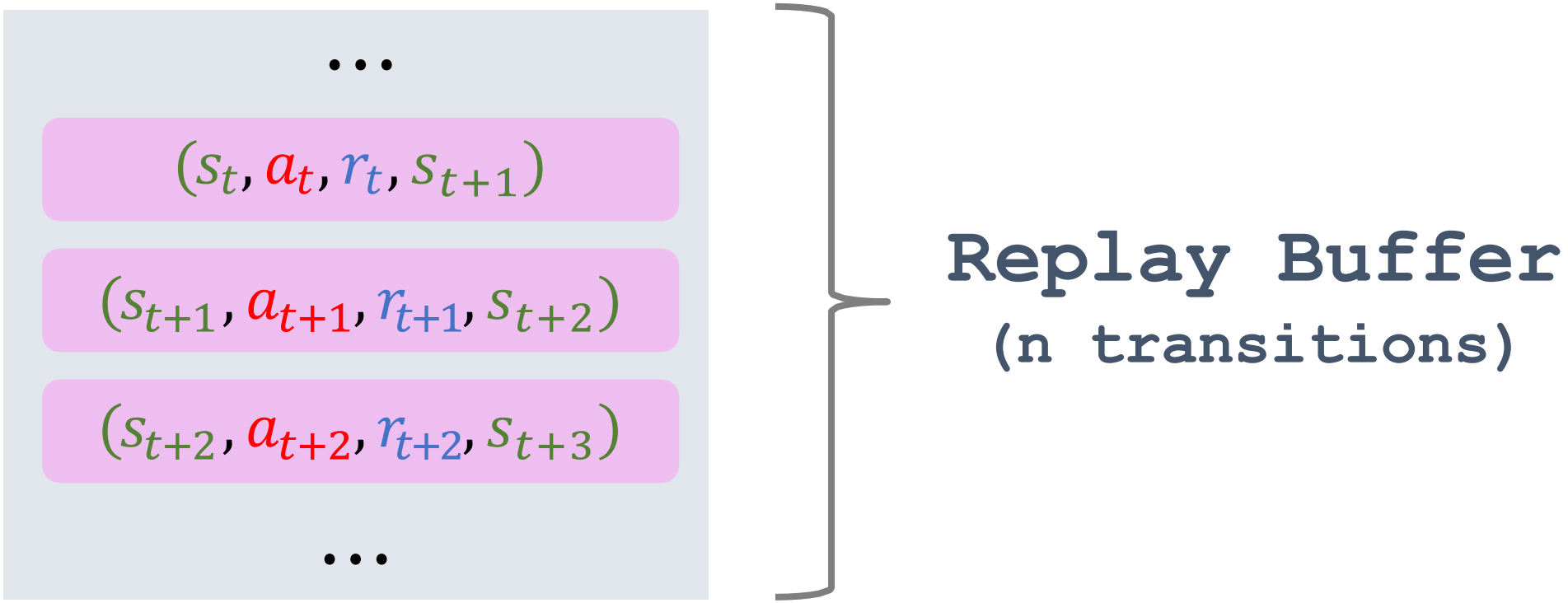


No longer stochastic but fixed *bias*

Experience Replay

Experience Replay

- **A transition:** (s_t, a_t, r_t, s_{t+1}) .
- Store recent n transitions in a **replay buffer**.



Experience Replay

- **A transition:** (s_t, a_t, r_t, s_{t+1}).
- Store recent n transitions in a **replay buffer**.
- Remove old transitions so that the buffer has at most n transitions.
- Buffer capacity n is a tuning hyper-parameter [1, 2].
  - n is typically large, e.g., $10^5 \sim 10^6$.
  - The setting of n is application-specific.

Reference:

1. Zhang & Sutton. A deeper look at experience replay. In *NIPS workshop*, 2017.
2. Fedus et al. Revisiting fundamentals of experience replay. In *ICML*, 2019.

TD with Experience Replay

- Mini-batch Stochastic gradient descent:
  - Randomly sample a mini batch, $\{(s_i, a_i, r_i, s_{i+1})\}_{i=1,\dots,K}$, from the buffer.
  - Compute TD error, $\{\delta_i\}$.
  - Stochastic gradient: $\mathbf{g} = \frac{1}{K}\sum_i \frac{\partial \delta_i^2}{\partial \mathbf{w}} = \frac{1}{K}\sum_i \delta_i \cdot \frac{\partial Q(s_i,a_i;\mathbf{w})}{\partial \mathbf{w}}$
  - SGD: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \mathbf{g}$.
- And we can do mini-batch SGD for multiple times at one online time step

Benefits of Experience Replay

1. Make the updates uncorrelated.
2. Reuse collected experience many times.

Abalation Study on DQNs

- Game score under difference conditions

	Replay Fixed-Q	Replay Q-learning	No replay Fixed-Q	No replay Q-learning
Breakout	316.81	240.73	10.16	3.17
Enduro	1006.3	831.25	141.89	29.1
River Raid	7446.62	4102.81	2867.66	1453.02
Seaquest	2894.4	822.55	1003	275.81
Space Invaders	1088.94	826.33	373.22	301.99

- Replay is hugely important

How to select Experience Replay hyper-parameter

- Does a larger ER buffer lead to better performance?
- No!

经验池最大容量

策略老旧程度

样本被从经验池中丢弃时，对应的生成时刻与当前时刻策略的update次数

Oldest Policy \ Replay Capacity	100,000	316,228	**1,000,000**	3,162,278	10,000,000
25,000,000	-74.9	-76.6	-77.4	-72.1	-54.6
2,500,000	-78.1	-73.9	-56.8	-16.7	**28.7**
250,000	-70.0	-57.4	**0.0**	13.0	18.3
25,000	**-31.9**	12.4	16.9	--	--

Figure 2. **Performance consistently improves with increased replay capacity and generally improves with reducing the age of the oldest policy.** Median percentage improvement over the Rainbow baseline when varying the replay capacity and age of the oldest policy in Rainbow on a 14 game subset of Atari. We do not run the two cells in the bottom-right because they are extremely expensive due to the need to collect a large number of transitions per policy.

- 增大经验池容量有助于提升minibatch的多样性和独立同分布式
- 减少策略老旧程度使得经验池中样本更像是 on-policy，有助于提升算法性能

Revisiting Fundamentals of Experience Replay, ICML2019

History

- Experience replay was proposed by Long-Ji Lin [1].
- The DQN paper [2] popularized experience replay.
- There are many improvements, e.g., [3].

Reference:

1. Lin. Reinforcement Learning for Robots Using Neural Networks. *PhD Dissertation*, 1993.
2. Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 2015.
3. Schaul et al. Prioritized experience replay. In *ICLR*, 2016.

Prioritized Experience Replay

Reference:

1. Schaul, Quan, Antonoglou, & Silver. Prioritized experience replay. In *ICLR*, 2016.

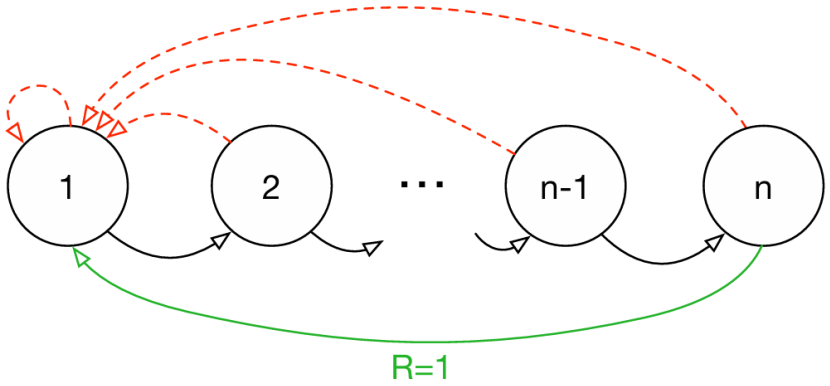
Basic Idea

- Not all transitions are equally important.
- Which kind of transition is more important, left or right?

Basic Idea

- How do we know which transition is important?
- If a transition has high TD error $|\delta_t|$, it will be given high priority.

Example: Blink Cliffwalk



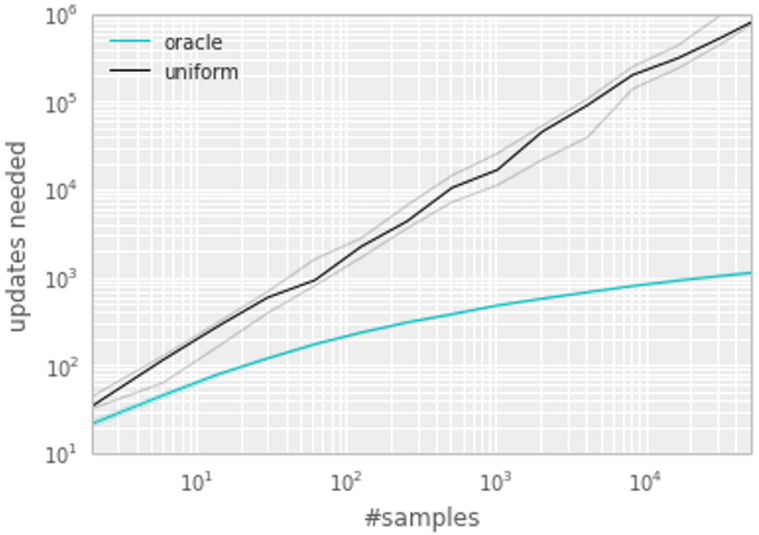
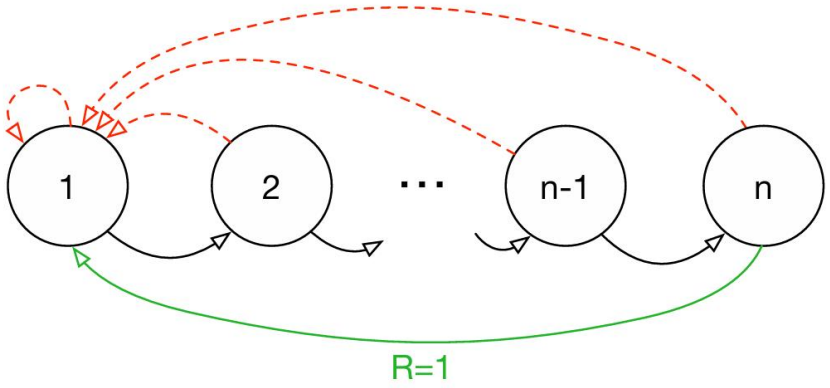
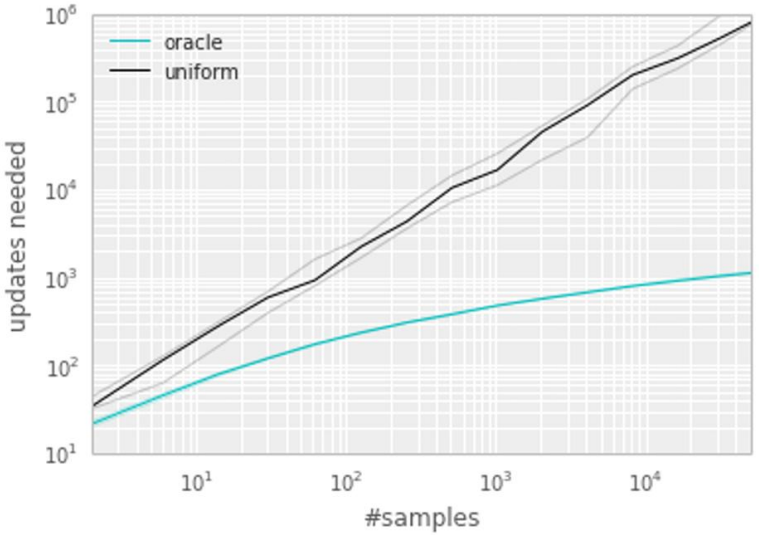


Figure 1: **Left**: Illustration of the 'Blind Cliffwalk' example domain: there are two actions, a 'right' and a 'wrong' one, and the episode is terminated whenever the agent takes the 'wrong' action (dashed red arrows). Taking the 'right' action progresses through a sequence of n states (black arrows), at the end of which lies a final reward of 1 (green arrow); reward is 0 elsewhere. We chose a representation such that generalizing over what action is 'right' is not possible. **Right**: Median number of learning steps required to learn the value function as a function of the size of the total number of transitions in the replay memory. Note the log-log scale, which highlights the exponential speed-up from replaying with an oracle (bright blue), compared to uniform replay (black); faint lines are min/max values from 10 independent runs.

Prioritized Experience Replay, ICLR2016

Example: Blink Cliffwalk





- Both agents perform Q-learning updates on transitions drawn from the same replay memory.
- The first agent (uniform sampling) replays transitions uniformly at random,
- while the second agent (oracle) invokes an oracle to prioritize transitions.
  - This oracle greedily selects the transition that maximally reduces the global loss in its current state (in hindsight, after the parameter update).
- Convergence is defined as a mean-squared error (MSE) between the Q-value estimates and the ground truth below 10e−3. We plot the number of updates needed for convergence.

Prioritized Experience Replay, ICLR2016

Prioritized Replay

- Use prioritized sampling instead of uniform sampling.
- Option 1: Sampling probability $p_t \propto |\delta_t| + \epsilon$.
- Option 2: Sampling probability $p_t \propto \frac{1}{\text{rank}(t)}$
  - The transitions are sorted so that $|\delta_t|$ is in the descending order.
  - $\text{rank}(t)$ is the rank of the t-th transition.
- In sum, big $|\delta_t|$ shall be given high priority.

Importance Sampling Weight

- SGD: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \mathbf{g}_t$, where α is the learning rate.

- If uniform sampling is used, sampled gradient is unbiased:

$$\mathbb{E}_{t\sim\mathcal{D}}[\mathbf{g}_t] = \nabla\mathbb{E}_{\mathcal{D}}[\delta_t^2]$$

  so α is the same for all transitions.

- If prioritized sampling is used, sampled gradient is biased:

$$\mathbb{E}_{t\sim\text{Prioritized}(\mathcal{D})}[\mathbf{g}_t] \neq \nabla\mathbb{E}_{\mathcal{D}}[\delta_t^2]$$

  α shall be adjusted according to the importance to correct the bias and update on the same distribution as its expectation.

Importance Sampling Weight

- Scale the learning rate by $(n\,p_t)^{-\beta}$ to compensate the non-uniform probability p_t, where $\beta \in (0,1)$.
- If $p_1 = \cdots = p_n = \frac{1}{n}$ (uniform sampling), the scaling factor is equal to 1.
- High-priority transitions (with high p_t) have low learning rates.
- In the beginning, set β small; increase β to 1 over time.
  (unbiased nature of the updates is most important near convergence at the end of training)

Update **TD** Error

- Associate each transition, (s_t, a_t, r_t, s_{t+1}) , with a TD error, δ_t .
- If a transition is newly collected, we do not know its δ_t .
  - Simply set its δ_t to the maximum.
  - It has the highest priority.
- Each time (s_t, a_t, r_t, s_{t+1}) is selected from the buffer, we update its δ_t .

Transitions	Sampling Probabilities	Learning Rates
...	...	...
$(s_t, a_t, r_t, s_{t+1}), \delta_t$	$p_t \propto \lvert\delta_t\rvert + \epsilon$	$\alpha \cdot (n\, p_t)^{-\beta}$
$(s_{t+1}, a_{t+1}, r_{t+1}, s_{t+2}), \delta_{t+1}$	$p_{t+1} \propto \lvert\delta_{t+1}\rvert + \epsilon$	$\alpha \cdot (n\, p_{t+1})^{-\beta}$
$(s_{t+2}, a_{t+2}, r_{t+2}, s_{t+3}), \delta_{t+2}$	$p_{t+2} \propto \lvert\delta_{t+2}\rvert + \epsilon$	$\alpha \cdot (n\, p_{t+2})^{-\beta}$
...	...	...

Big $|\delta_t|$ ==> High probability ==> Small learning rate

Example: Blink Cliffwalk

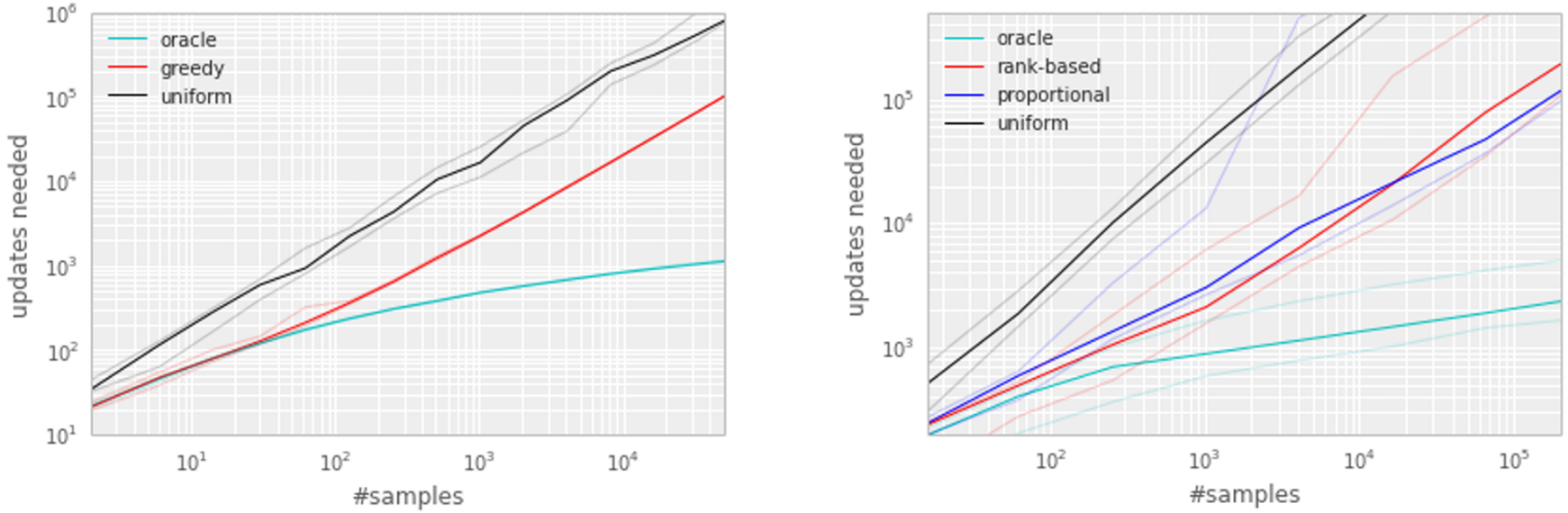
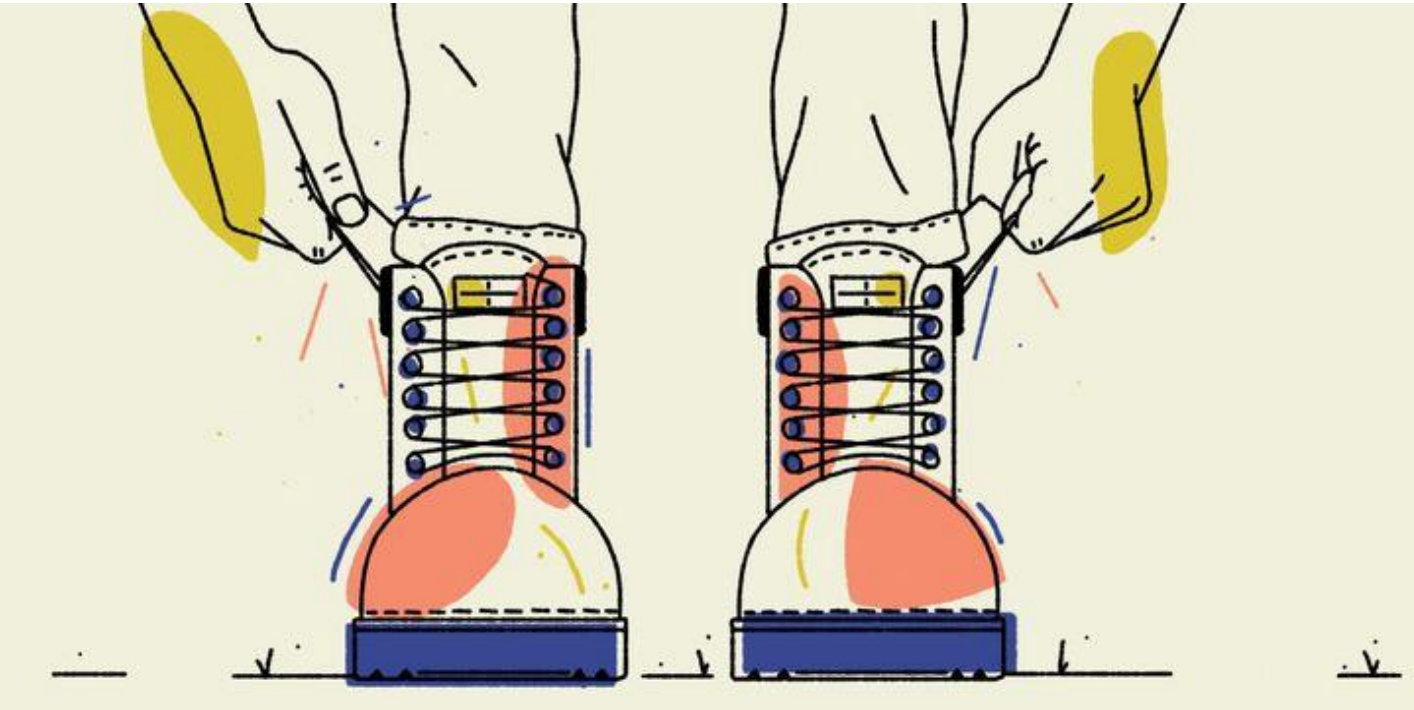


Figure 2: Median number of updates required for Q-learning to learn the value function on the Blind Cliffwalk example, as a function of the total number of transitions (only a single one of which was successful and saw the non-zero reward). Faint lines are min/max values from 10 random initializations. Black is uniform random replay, cyan uses the hindsight-oracle to select transitions, red and blue use prioritized replay (rank-based and proportional respectively). The results differ by multiple orders of magnitude, thus the need for a log-log plot. In both subplots it is evident that replaying experience in the right order makes an enormous difference to the number of updates required. See Appendix B.1 for details. **Left:** Tabular representation, greedy prioritization. **Right**: Linear function approximation, both variants of stochastic prioritization.

Bootstrapping



Bootstrapping: To lift oneself up by his bootstraps.

TD Learning for DQN

- In RL, bootstrapping means "*using an estimated value in the update step for the same kind of estimated value*".
- Use a transition, (s_t, a_t, r_t, s_{t+1}), to update $\mathbf{w}$.
  - TD target: $y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w})$.
  - TD error: $\delta_t = Q(s_t, a_t; \mathbf{w}) - y_t$.
  - SGD: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \delta_t \cdot \frac{\partial\, Q(s_t, a_t; \mathbf{w})}{\partial\, \mathbf{w}}$.

TD target y_t is partly an estimate made by the DQN Q.

We use y_t , which is partly based on Q, to update Q itself.

Problem of Overestimation

Problem of Overestimation

- TD learning makes DQN overestimate action-values. (Why?)
- **Reason 1:** The maximization.
  - TD target: $y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w})$.
  - TD target is bigger than the real action-value.
- **Reason 2:** Bootstrapping propagates the overestimation.

Reason 1: Maximization

- Let $x_1, x_2, \ldots, x_n$ be observed real numbers.
- Add zero-mean random noise to $x_1, x_2, \ldots, x_n$ and obtain $Q_1, \ldots, Q_n$.
- The zero-mean noise does not affect the mean:

$$\mathbb{E}[\text{mean}_i(Q_i)] = \text{mean}_i(x_i).$$

- The zero-mean noise increases the maximum:

$$\mathbb{E}[\max_i(Q_i)] \geq \max_i(x_i).$$

- The zero-mean noise decreases the minimum:

$$\mathbb{E}[\min_i(Q_i)] \leq \min_i(x_i).$$

Reason 1: Maximization

- True action-values: $x(a_1), \ldots, x(a_n)$.
- Noisy estimations made by DQN: $Q(s, a_1; \mathbf{w}), \cdots, Q(s, a_n; \mathbf{w})$.
- Suppose the estimation is unbiased:

$$\operatorname{mean}_a\big(x(a)\big) = \operatorname{mean}_a\big(Q(s, a; \mathbf{w})\big).$$

- $q = \max_a Q(s, a; \mathbf{w})$, is typically an overestimation:

$$q \geq \max_a\big(x(a)\big).$$

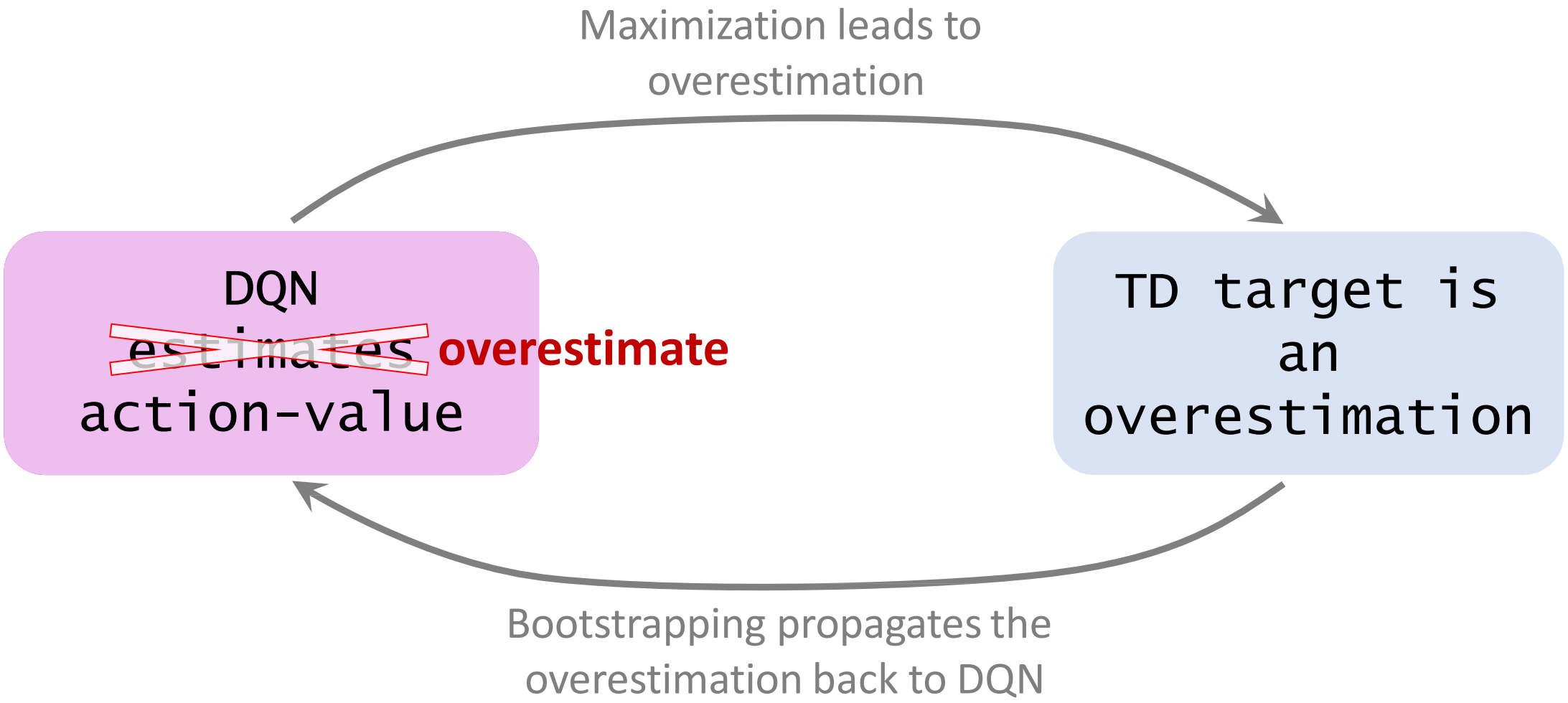
Reason 1: Maximization

- We conclude that $q_{t+1} = \max_a Q(s_{t+1}, a; \mathbf{w})$ is an overestimation of the true action-value at time $t + 1$.
- The TD target, $y_t = r_t + \gamma \cdot q_{t+1}$, is thereby an overestimation.
- TD learning pushes $Q(s_t, a_t; \mathbf{w})$ towards y_t which overestimates the true action-value.

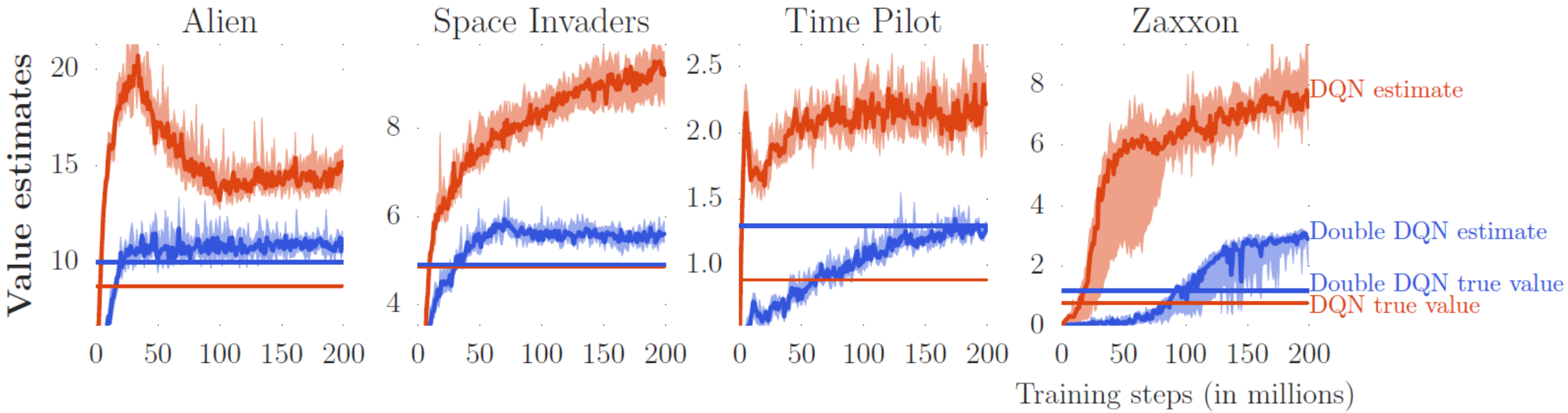
Reason 2: Bootstrapping

- TD learning performs bootstrapping.
  - TD target in part uses $q_{t+1} = \max_a Q(s_{t+1}, a; \mathbf{w})$.
  - Use the TD target for updating $Q(s_t, a_t; \mathbf{w})$.
- Suppose DQN overestimates the action-value.
- Then $Q(s_{t+1}, a; \mathbf{w})$ is an overestimation.
- The maximization further pushes q_{t+1} up.
- When q_{t+1} is used for updating $Q(s_t, a_t; \mathbf{w})$, the overestimation is propagated back to DQN.

Why does overestimation happen?



Why does overestimation happen?

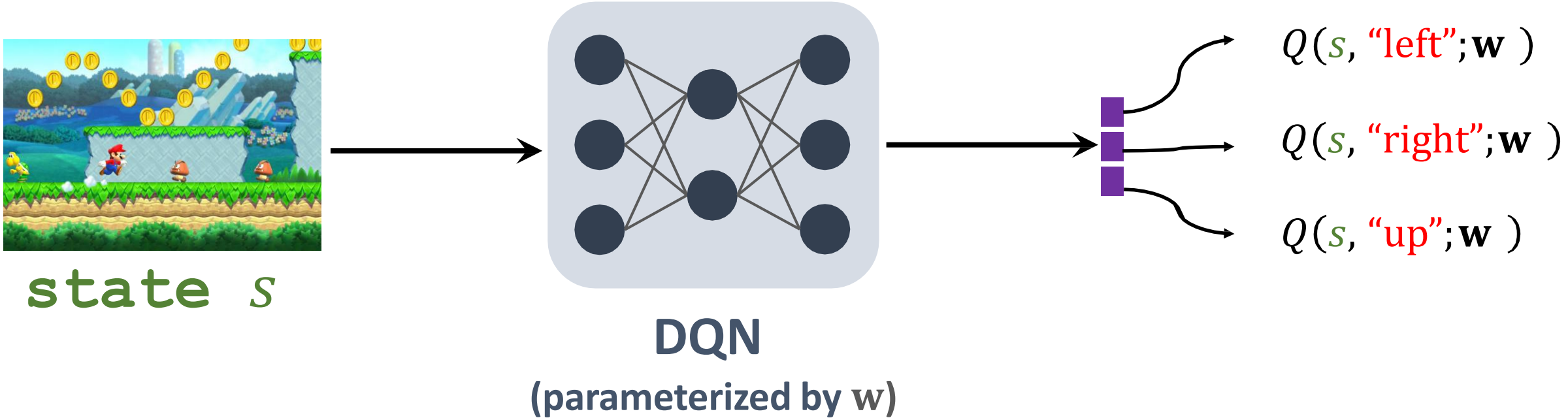


Deep reinforcement learning with double Q-Learning, AAAI'16

Why is overestimation harmful?

The agent is controlled by the DQN: $a_t = \underset{a}{\operatorname{argmax}}\, Q(s_t, a; \mathbf{w})$.

Uniform overestimation is not a problem.



Why is overestimation harmful?

The agent is controlled by the DQN: $a_t = \underset{a}{\operatorname{argmax}}\, Q(s_t, a; \mathbf{w})$.

Uniform overestimation is not a problem.

- $Q^\star(s, a^1) = 200$, $Q^\star(s, a^2) = 100$, and $Q^\star(s, a^3) = 230$.
- Action a^3 will be selected.
- Suppose $Q(s, a^i; \mathbf{w}) = Q^\star(s, a^i) + 100$, for all a^i.
- Then DQN believes a^3 has the highest value and will select a^3 .

Why is overestimation harmful?

The agent is controlled by the DQN: $a_t = \underset{a}{\operatorname{argmax}}\, Q(s_t, a; \mathbf{w})$.

Uniform overestimation is not a problem.

Non-uniform overestimation is problematic.

- $Q^\star(s, a^1) = 200$, $Q^\star(s, a^2) = 100$, and $Q^\star(s, a^3) = 230$.
- $Q(s, a^1; \mathbf{w}) = 280$, $Q(s, a^2; \mathbf{w}) = 300$, and $Q(s, a^3; \mathbf{w}) = 240$.
- Then a^2 (which is bad) will be selected.

Why is overestimation harmful?

Unfortunately, the overestimation is non-uniform.

- Use a transition, (s_t, a_t, r_t, s_{t+1}), to update $\mathbf{w}$.
- The TD target, y_t, overestimates $Q^\star(s_t, a_t)$.
- TD algorithm pushes $Q(s_t, a_t; \mathbf{w})$ towards y_t.
- Thus, $Q(s_t, a_t; \mathbf{w})$ overestimates $Q^\star(s_t, a_t)$.

The more frequently (s, a) appears in the replay buffer,
the worse $Q(s, a; \mathbf{w})$ overestimates $Q^\star(s, a)$.

Solutions

- **Problem:** DQN trained by TD overestimates action-values.
- **Solution 1:** Use a target network [1] to compute TD targets. (Address the problem caused by bootstrapping.)
- **Solution 2:** Use double DQN [2] to alleviate the overestimation caused by maximization.

Reference:

1. Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 2015.
2. Van Hasselt, Guez, & Silver. Deep reinforcement learning with double Q-learning. In *AAAI*, 2016.

Target Network

Reference:

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

Target Network

- Target network: $Q(s, a; \mathbf{w}^-)$
  - The same network structure as the DQN, $Q(s, a; \mathbf{w})$.
  - Different parameters: $\mathbf{w}^- \neq \mathbf{w}$.
- Use $Q(s, a; \mathbf{w})$ to control the agent and collect experience:

$$\{(s_t, a_t, r_t, s_{t+1})\}.$$

- Use $Q(s, a; \mathbf{w}^-)$ to compute TD target:

$$y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w}^-).$$

TD Learning with Target Network

- Use a transition, (s_t, a_t, r_t, s_{t+1}), to update $\mathbf{w}$.
  - TD target: $y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w}^-)$.
  - TD error: $\delta_t = Q(s_t, a_t; \mathbf{w}) - y_t$.
  - SGD: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \delta_t \cdot \frac{\partial\, Q(s_t, a_t; \mathbf{w})}{\partial\, \mathbf{w}}$.

Update Target Network

- Periodically update $\mathbf{w}^-$.
- Option 1: $\mathbf{w}^- \leftarrow \mathbf{w}$.
- Option 2: $\mathbf{w}^- \leftarrow \tau \cdot \mathbf{w} + (1-\tau) \cdot \mathbf{w}^-$.

Complete Implementation of DQN

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

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

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 $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    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

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

  End For

End For

Initialization

Collection online data

Experience replay

Compute TD target with TargetNet

Update CurrentNet

Update TargetNet

Human-Level Control Through Deep Reinforcement Learning, Nature'15

Comparisons

- TD learning with naïve update:

$$\text{TD Target: } y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w}).$$

- TD learning with target network:

$$\text{TD Target: } y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w}^-).$$

- Though better than the naïve update, TD learning with target network nevertheless overestimates action-values.

Double DQN

Reference:

1. Van Hasselt, Guez, & Silver. Deep reinforcement learning with double Q-learning. In *AAAI*, 2016.

Naïve Update

TD Target: $y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w})$.

Reference:

1. Mnih et al. Playing Atari with deep reinforcement learning. In *NIPS Workshop*, 2013.

Naïve Update

- Selection using DQN:

$$a^\star = \operatorname*{argmax}_a Q(s_{t+1}, a; \mathbf{w}).$$

- Evaluation using DQN:

$$y_t = r_t + \gamma \cdot Q(s_{t+1}, a^*; \mathbf{w}).$$

- Serious overestimation.

Reference:

1. Mnih et al. Playing Atari with deep reinforcement learning. In *NIPS Workshop*, 2013.

Using Target Network

- Selection using target network:

$$a^{\star} = \underset{a}{\operatorname{argmax}}\, Q(s_{t+1}, a; \mathbf{w}^{-}).$$

- Evaluation using target network:

$$y_t = r_t + \gamma \cdot Q(s_{t+1}, a^{*}; \mathbf{w}^{-}).$$

- It works better, but overestimation is still serious.

Reference:

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

Double DQN

- Selection using DQN:

$$a^\star = \underset{a}{\operatorname{argmax}}\, Q(s_{t+1}, a; \mathbf{w}).$$

- Evaluation using target network:

$$y_t = r_t + \gamma \cdot Q(s_{t+1}, a^*; \mathbf{w}^-).$$

- It is the best among the three; but overestimation still happens.

Reference:

1. Van Hasselt, Guez, & Silver. Deep reinforcement learning with double Q-learning. In *AAAI*, 2016.

Why does double DQN work better?

- Double DQN decouples selection from evaluation.
- Selection using DQN: $a^{\star} = \underset{a}{\operatorname{argmax}}\, Q(s_{t+1}, a; \mathbf{w})$.
- Evaluation using target network: $y_t = r_t + \gamma \cdot Q(s_{t+1}, a^{*}; \mathbf{w}^{-})$.

Why does double DQN work better?

- Double DQN decouples selection from evaluation.
- Selection using DQN: $a^{\star} = \underset{a}{\operatorname{argmax}}\, Q(s_{t+1}, a; \mathbf{w})$.
- Evaluation using target network: $y_t = r_t + \gamma \cdot Q(s_{t+1}, a^{*}; \mathbf{w}^{-})$.
- Double DQN alleviates overestimation:

$$Q(s_{t+1}, a^{*}; \mathbf{w}^{-}) \quad \leq \quad \max_{a} Q(s_{t+1}, a^{*}; \mathbf{w}^{-}) \ .$$

Estimation by Double DQN

Estimation by target network

Summary of Overestimation & Solutions

- Because of the maximization, the TD target overestimates the true action-value.
- By creating a "positive feedback loop", bootstrapping further exacerbates the overestimation.
- Target network can partly avoid bootstrapping. (Not completely, because $\mathbf{w}^-$ depends on $\mathbf{w}$.)
- Double DQN alleviates the overestimation caused by the maximization.

Computing TD Targets

	Selection	Evaluation
Naïve Update	DQN	DQN
Using Target Network	Target Network	Target Network
Double DQN	DQN	Target Network

Dueling Network

Reference:

1. Wang et al. Dueling network architectures for deep reinforcement learning. In *ICML*, 2016.

Optimal Value Functions

Definition: Optimal action-value function.

- $Q^*(s, a) = \max_\pi Q_\pi(s, a)$.

Definition: Optimal state-value function.

- $V^*(s) = \max_\pi V_\pi(s)$.

Definition: Optimal advantage function.

- $A^\star(s, a) = Q^\star(s, a) - V^\star(s)$.

Properties of Advantage Function

Theorem 1: $V^\star(s) = \max_a Q^*(s, a)$.

- Recall the definition of the optimal advantage function:

$$A^\star(s, a) = Q^\star(s, a) - V^\star(s).$$

- It follows that

$$\max_a A^\star(s, a) = \max_a Q^\star(s, a) - V^\star(s).$$

$$= 0$$

Properties of Advantage Function

Definition of advantage: $A^\star(s,a) = Q^\star(s,a) - V^\star(s).$

$$Q^\star(s,a) = V^\star(s) + A^\star(s,a)$$

Properties of Advantage Function

Definition of advantage: $A^\star(s,a) = Q^\star(s,a) - V^\star(s)$

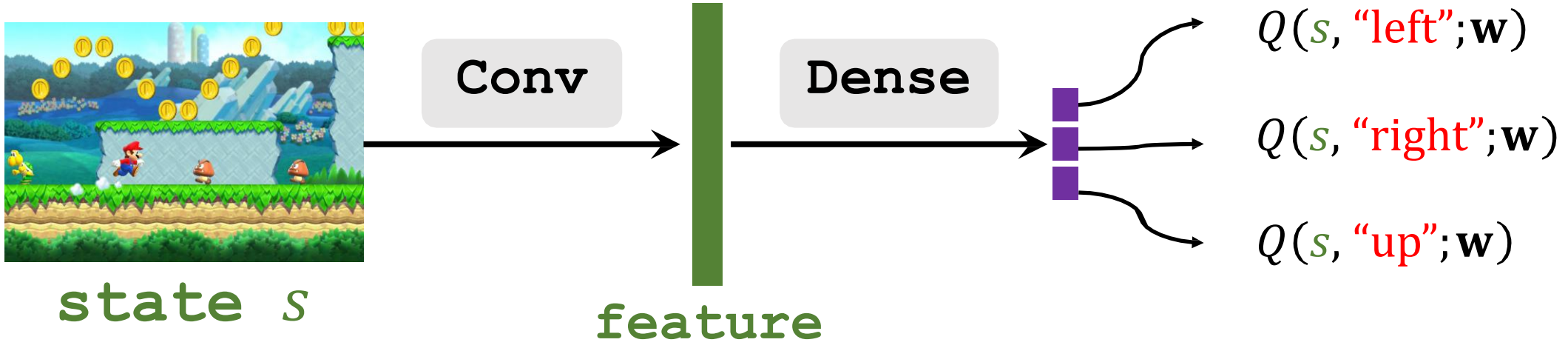
.

Theorem 2: $Q^\star(s,a) = V^\star(s) + A^\star(s,a) - \max_a A^\star(s,a).$

$= 0$

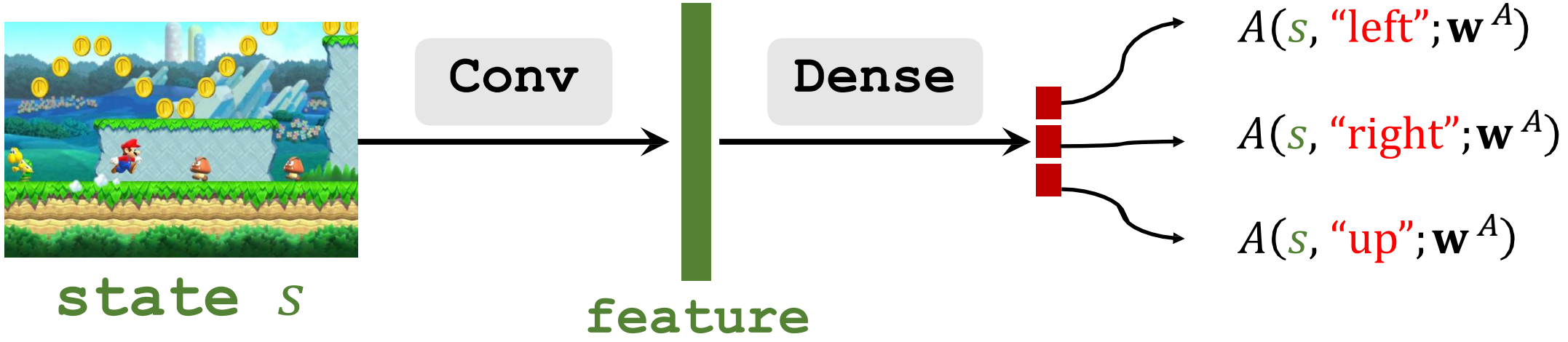
Revisiting DQN

- Approximate $Q^{\star}(s, a)$ by a neural network, $Q(s, a; \mathbf{w})$.



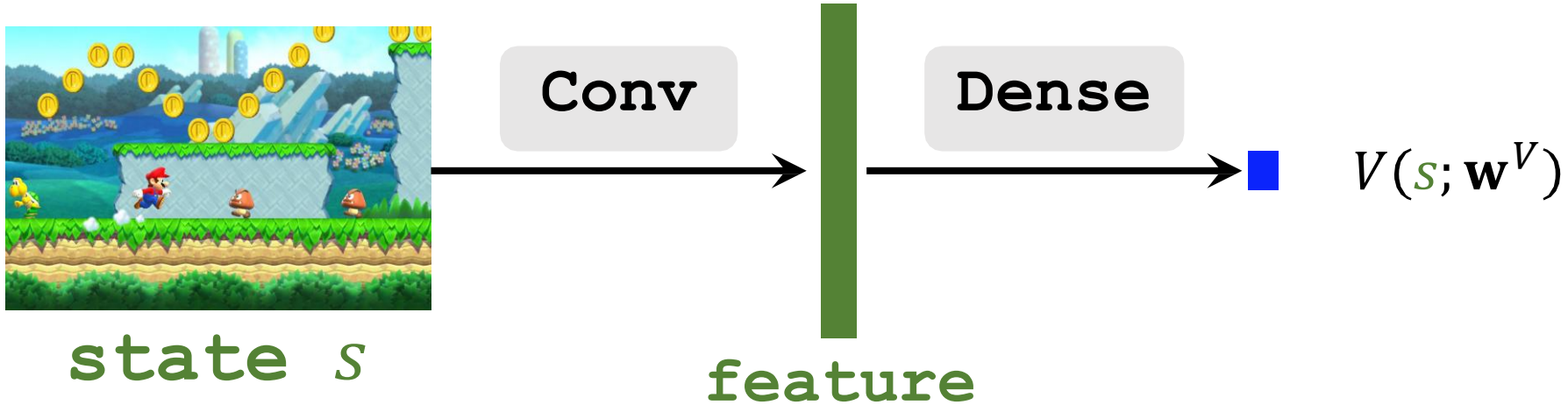
Approximating Advantage Function

- Approximate $A^{\star}(s, a)$ by a neural network, $A(s, a; \mathbf{w}^{A})$.



Approximating State-Value Function

- Approximate $V^{\star}(s)$ by a neural network, $V(s; \mathbf{w}^V)$.



Dueling Network: Formulation

Theorem 2: $Q^\star(s,a) = V^\star(s) + A^\star(s,a) - \max_a A^\star(s,a)$.

- Approximate $V^\star(s)$ by a neural network, $V(s;\mathbf{w}^{\mathrm{V}})$.
- Approximate $A^\star(s,a)$ by a neural network, $A(s,a;\mathbf{w}^{\mathrm{A}})$.
- Thus, approximate $Q^\star(s,a)$ by the dueling network:

$$Q(s,a;\mathbf{w}^{\mathrm{A}},\mathbf{w}^{\mathrm{V}}) = V(s;\mathbf{w}^{\mathrm{V}}) + A(s,a;\mathbf{w}^{\mathrm{A}}) - \max_a A(s,a;\mathbf{w}^{\mathrm{A}}).$$

$$\mathbf{w} = (\mathbf{w}^{\mathrm{A}},\mathbf{w}^{\mathrm{V}})$$

Dueling Network

$$Q(s, a; \mathbf{w}) = V(s; \mathbf{w}^{\mathrm{V}}) + A(s, a; \mathbf{w}^{\mathrm{A}}) - \max_{a} A(s, a; \mathbf{w}^{\mathrm{A}}).$$

state s

Conv

feature

Dense1

Dense2

$A(s, \text{“left”}; \mathbf{w}^{\mathrm{A}})$

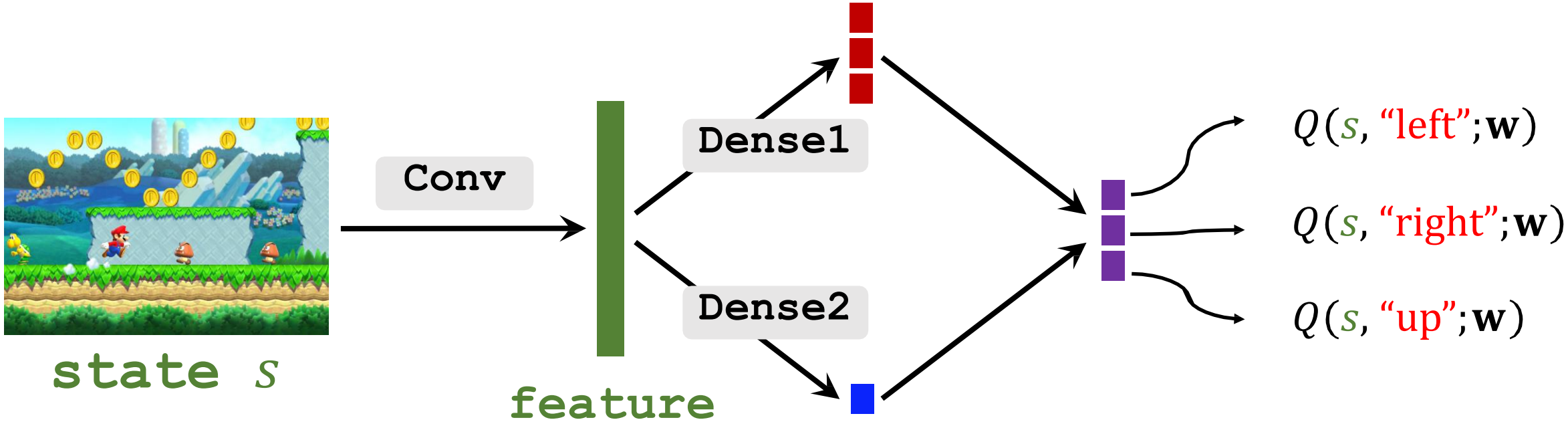
$A(s, \text{“right”}; \mathbf{w}^{\mathrm{A}})$

$A(s, \text{“up”}; \mathbf{w}^{\mathrm{A}})$

$V(s; \mathbf{w}^{\mathrm{V}})$

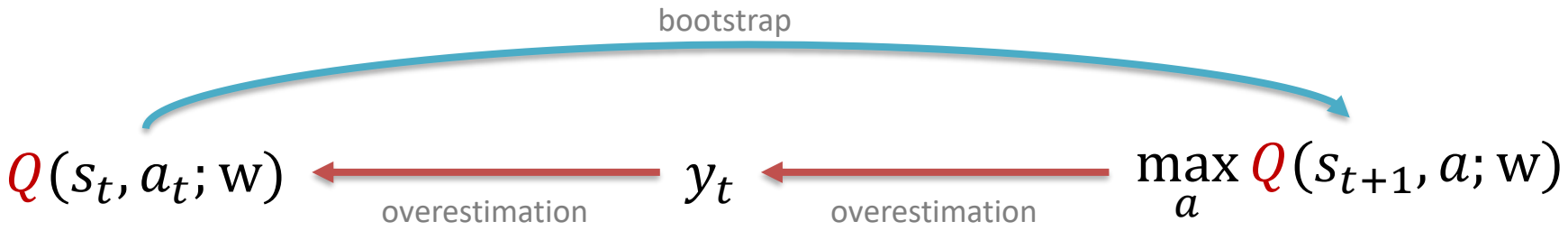
Dueling Network

$$Q(s, a; \mathbf{w}) = V(s; \mathbf{w}^{\mathrm{V}}) + A(s, a; \mathbf{w}^{\mathrm{A}}) - \max_{a} A(s, a; \mathbf{w}^{\mathrm{A}}).$$

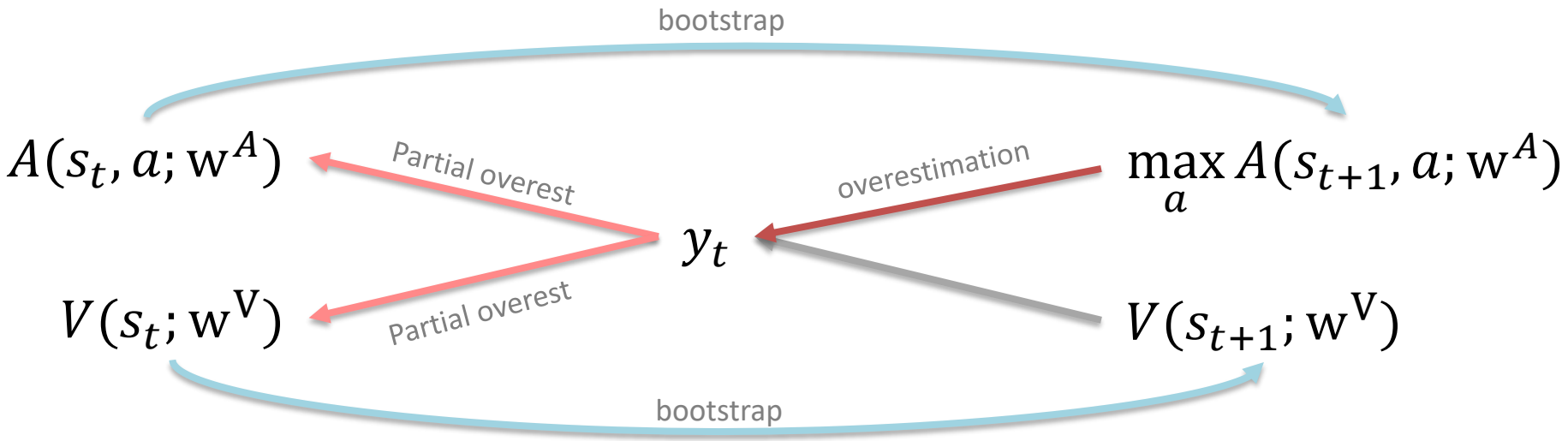


Benefit of Dueling Network

- DQN overestimation propagation



- Dueling network overestimation propagation



Dueling Network

- Visualization

Dueling Network

- Visualization

Training

- Dueling network, $Q(s, a; \mathbf{w})$, is an approximation to $Q^{\star}(s, a)$.
- Learn the parameter, $\mathbf{w} = (\mathbf{w}^{\mathrm{A}}, \mathbf{w}^{\mathrm{V}})$, in the same way as the other DQNs.
- Tricks can be used in the same way.
  - Prioritized experience replay.
  - Double DQN.
  - Multi-step TD target.

Overcome Non-identifiability

Problem of Non-identifiability

- **Equation 1:** $Q^\star(s,a) = V^\star(s) + A^\star(s,a).$
- **Equation 2:** $Q^\star(s,a) = V^\star(s) + A^\star(s,a) - \max_a A^\star(s,a).$

Question: Why is the zero term necessary?

Problem of Non-identifiability

- **Equation 1:** $Q^\star(s,a) = V^\star(s) + A^\star(s,a).$
- Equation 1 has the problem of *non-identifiability*.
  - Let $V' = V^\star + 10$ and $A' = A^\star - 10.$
  - Then $Q^\star(s,a) = V^\star(s) + A^\star(s,a) = V'(s) + A'(s,a).$
- Why is non-identifiability a problem?

Problem of Non-identifiability

- **Equation 2:** $Q^\star(s,a) = V^\star(s) + A^\star(s,a) - \max_a A^\star(s,a).$
- Equation 2 does not have the problem.

Dueling Network

$$Q(s, a; \mathbf{w}) = V(s; \mathbf{w}^{\mathrm{V}}) + A(s, a; \mathbf{w}^{\mathrm{A}}) - \max_{a} A(s, a; \mathbf{w}^{\mathrm{A}}).$$

Alternative:

$$Q(s, a; \mathbf{w}) = V(s; \mathbf{w}^{\mathrm{V}}) + A(s, a; \mathbf{w}^{\mathrm{A}}) - \operatorname{mean}_{a} A(s, a; \mathbf{w}^{\mathrm{A}}).$$

(avoid a new overestimation induced by $\max_{a} A(s, a; \mathbf{w}^{\mathrm{A}})$

Summary

- **Dueling network:**

$$Q(s, a; \mathbf{w}) = V(s; \mathbf{w}^{\mathrm{V}}) + A(s, a; \mathbf{w}^{\mathrm{A}}) - \operatorname*{mean}_{a} A(s, a; \mathbf{w}^{\mathrm{A}}).$$

- Dueling network controls the agent in the same way as DQN.
- Train dueling network by TD in the same way as DQN.
- (Do not train V and A separately.)

Other techniques in DQN

Multi-step Return

- Use trajectory to compute N-step return

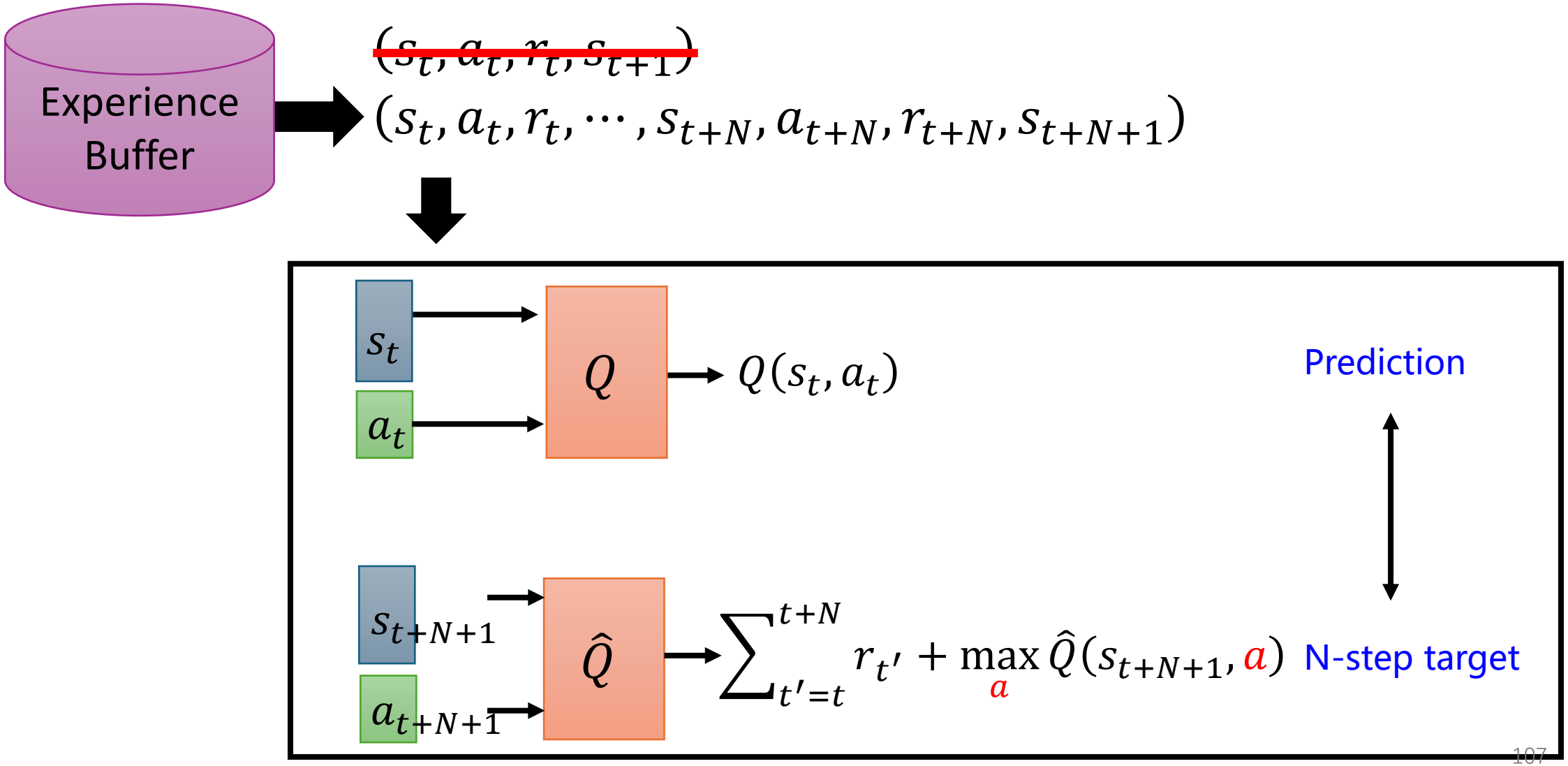
$$G_t = \sum_{t'=t}^{t+N} r_{t'} + \max_{a} \hat{Q}(s_{t+N+1}, a)$$

  - Balance between MC and TD
  - Better estimation accuracy 👍, higher variance 👎

- Samples in experience replay buffer need to keep temporal property (complete trajectory)

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

Multi-step Return



Noisy Net

- Noise on Action (Epsilon Greedy)

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

  - Most common exploration approach in Q-learning
  - Less efficiency in exploration
- Noise on Parameters

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

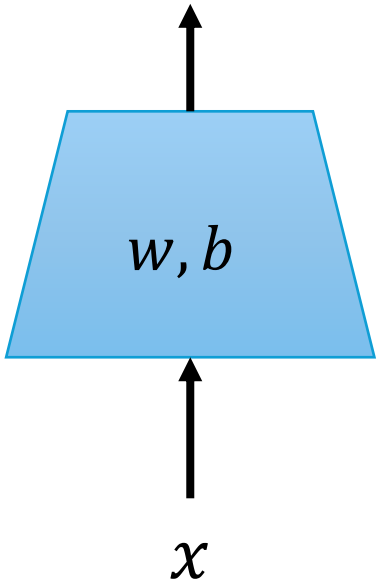
  - Inject noise into the parameters of Q-function **at the beginning of each episode**
  - The noise would **NOT** change in an episode.

Parameter Space Noise for Exploration (ICLR2018) , Noisy Networks for Exploration (ICLR2018)

Noisy Net

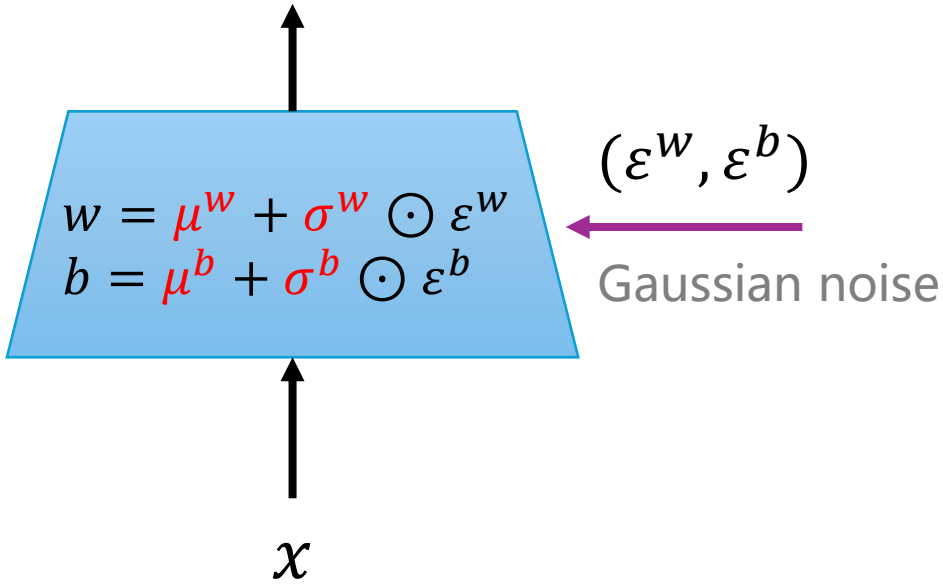
Linear layer

$y = wx + b$



Noisy linear layer

$y = wx + b$



- μ^w, σ^w, μ^b, σ^b are learnable parameters
- $\varepsilon^w, \varepsilon^b$ are noise random variables

Noisy Net

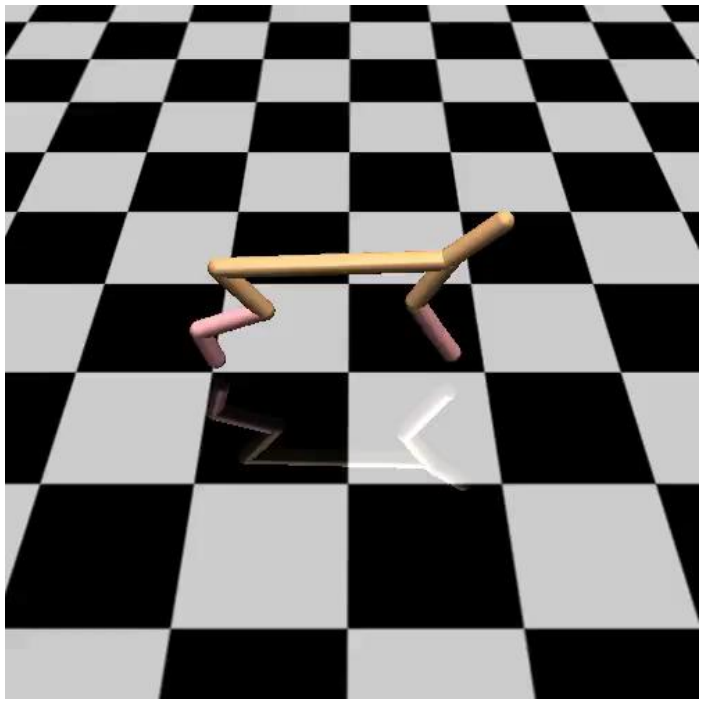
- Noise on Action
  - Given the same state, the agent may take different actions.
  - No real policy works in this way

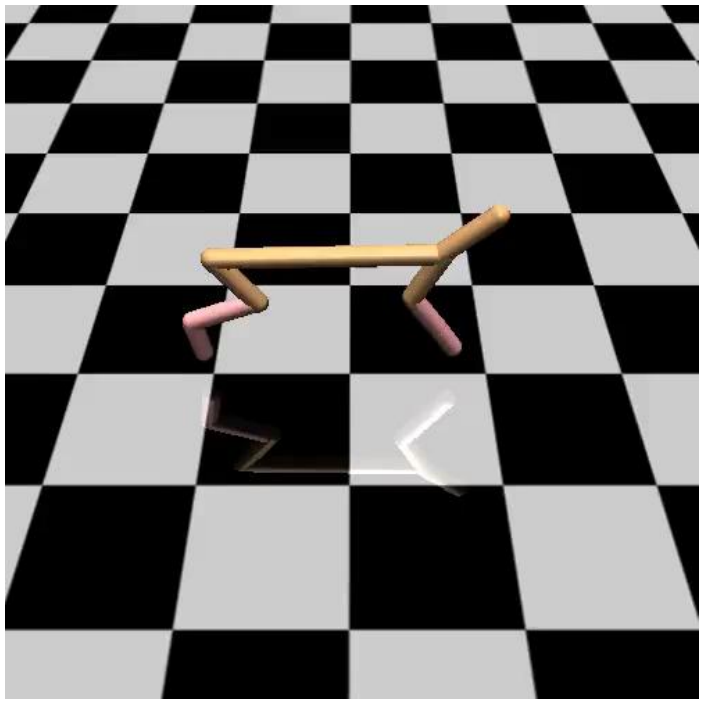
随机乱试

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

系统性地试

Demo

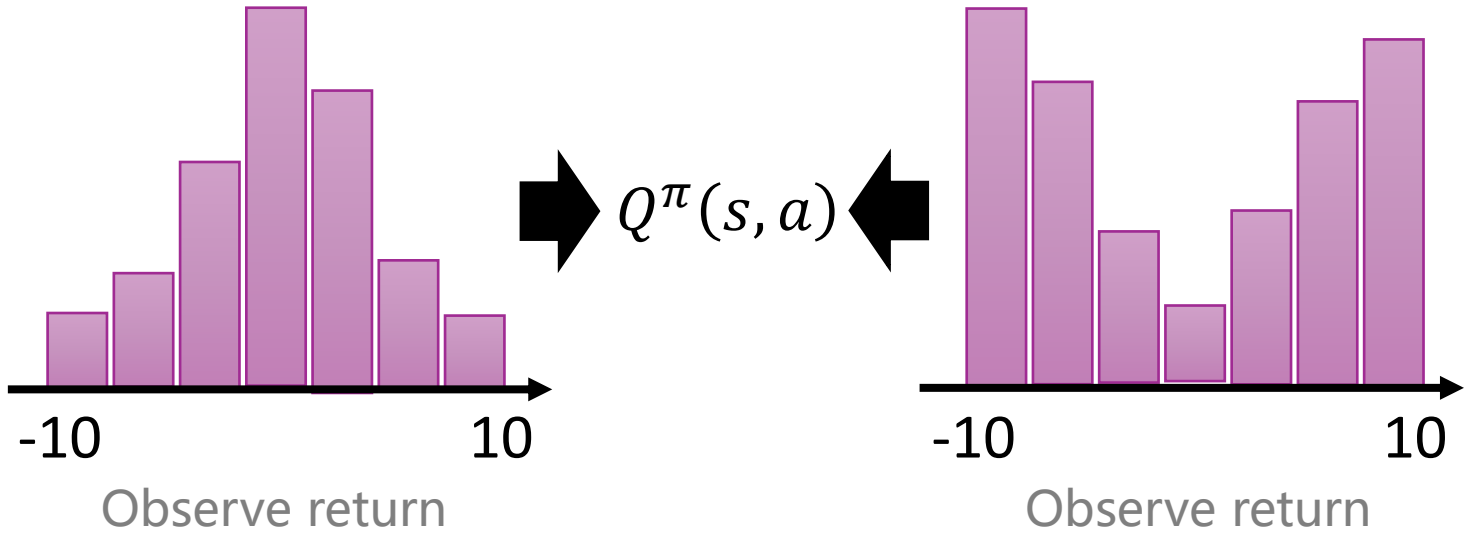




https://blog.openai.com/better-exploration-with-parameter-noise/

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

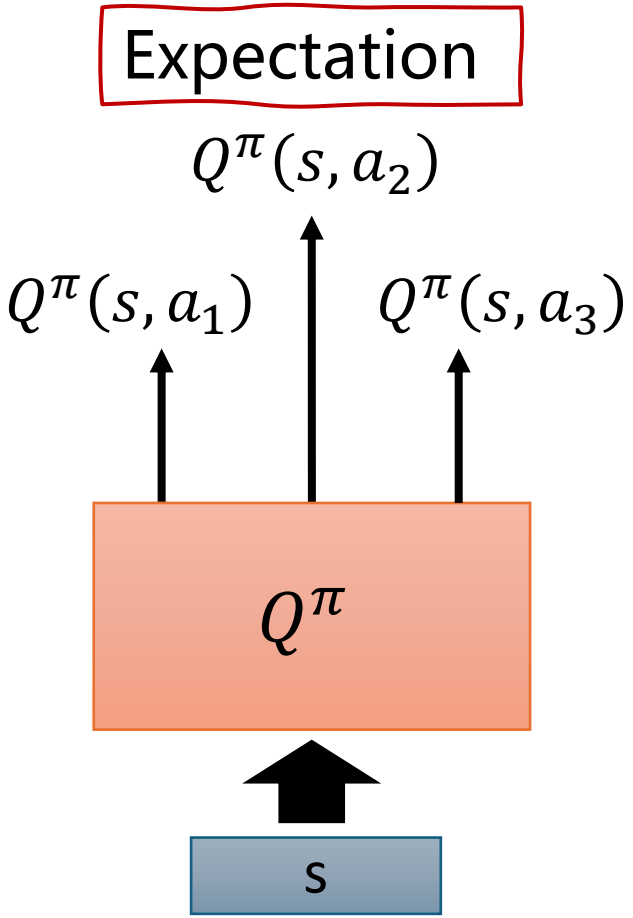
- Not only interested in its expectation $Q(s,a)$, but also Q/return distribution $Z(s,a)$
- C51 models the value distribution using a discrete distribution parametrized by

$$Z_\theta(s,a) = z_i \quad \text{w.p. } p_i(s,a) \coloneqq \frac{\exp(\theta_i(s,a))}{\sum_j \exp(\theta_j(s,a))}$$

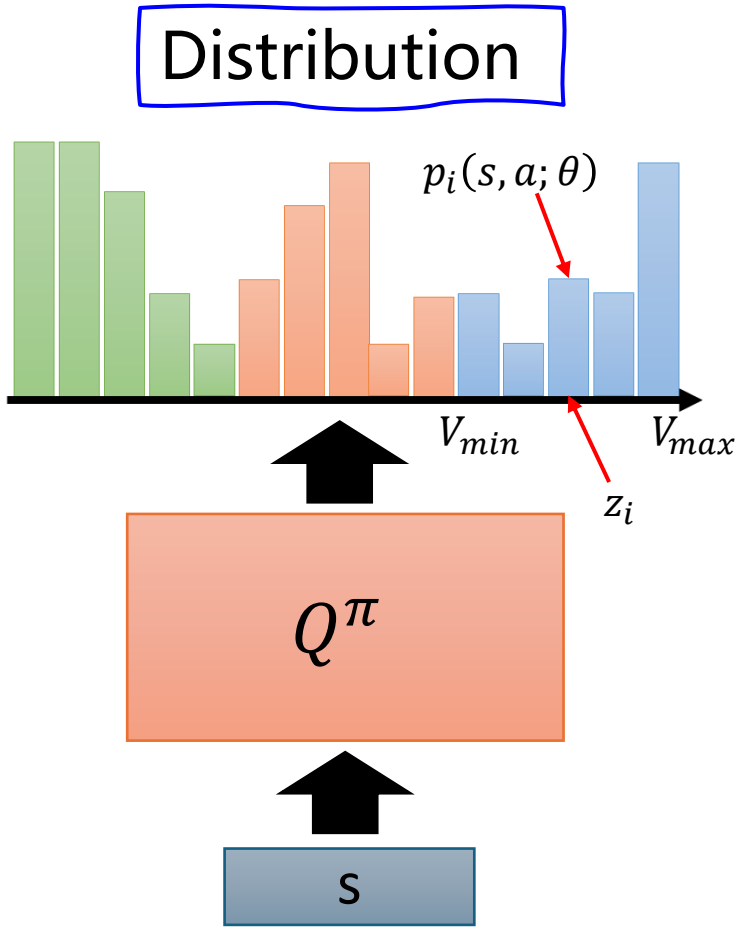
  - $N \in \mathbb{N}$, $V_{min}, V_{max} \in \mathbb{R}$,
  - the support $\{z_i = V_{min} + i\Delta z : 0 \leq i < N\}$, $\Delta z \coloneqq \frac{V_{max} - V_{min}}{N-1}$ (also called the set of atoms)
  - The atom probability is given by a parametric model $\theta : S \times A \to \mathbb{R}^N$

A Distributional Perspective on Reinforcement Learning, ICML'17

Distributional Q-function



A network with 3 outputs



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

Distributional Q-function

- Bellman distributed operator

$$\mathcal{T}^{\pi}Z(s,a) \coloneqq R(s,a) + \gamma P^{\pi}Z(s,a)$$

- Project Bellman update back to parametric distribution

Bellman update for next atom $z_j = r + \gamma z_j$

$$(\Phi\hat{\mathcal{T}}Z_\theta(x,a))_i = \sum_{j=0}^{N-1}\left[1 - \frac{|[\hat{\mathcal{T}}z_j]_{V_{\text{MIN}}}^{V_{\text{MAX}}} - z_i|}{\triangle z}\right]_0^1 p_j(x', \pi(x'))$$

atom z_i new prob

Prob p_j

$$D_{\text{KL}}(\Phi\hat{\mathcal{T}}Z_{\tilde{\theta}}(x,a) \,\|\, Z_\theta(x,a))$$

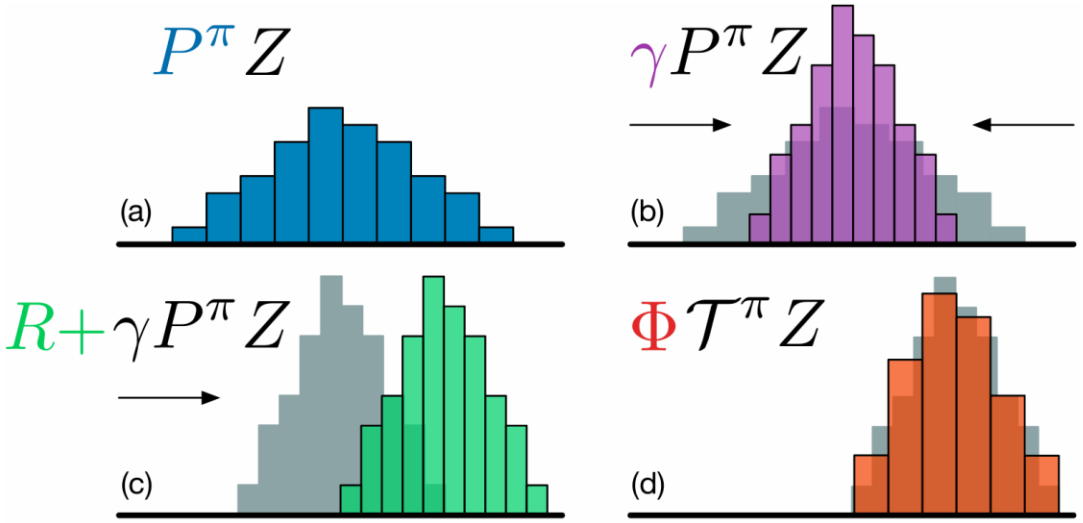


Figure 1. A distributional Bellman operator with a deterministic reward function: (a) Next state distribution under policy π, (b) Discounting shrinks the distribution towards 0, (c) The reward shifts it, and (d) Projection step (Section 4).

A Distributional Perspective on Reinforcement Learning, ICML2017

Algorithm 1 Categorical Algorithm

input A transition $x_t, a_t, r_t, x_{t+1}, \gamma_t \in [0, 1]$

$Q(x_{t+1}, a) := \sum_i z_i p_i(x_{t+1}, a)$

$a^* \leftarrow \arg\max_a Q(x_{t+1}, a)$

$m_i = 0, \quad i \in 0, \ldots, N-1$

for $j \in 0, \ldots, N-1$ **do**

Compute the projection of $\hat{\mathcal{T}} z_j$ onto the support $\{z_i\}$

$\hat{\mathcal{T}} z_j \leftarrow [r_t + \gamma_t z_j]_{V_{\text{MIN}}}^{V_{\text{MAX}}}$

$b_j \leftarrow (\hat{\mathcal{T}} z_j - V_{\text{MIN}})/\Delta z \quad \# \, b_j \in [0, N-1]$

$l \leftarrow \lfloor b_j \rfloor, u \leftarrow \lceil b_j \rceil$

Distribute probability of $\hat{\mathcal{T}} z_j$

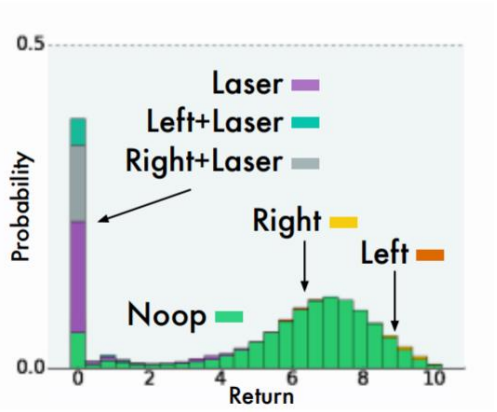
$m_l \leftarrow m_l + p_j(x_{t+1}, a^*)(u - b_j)$

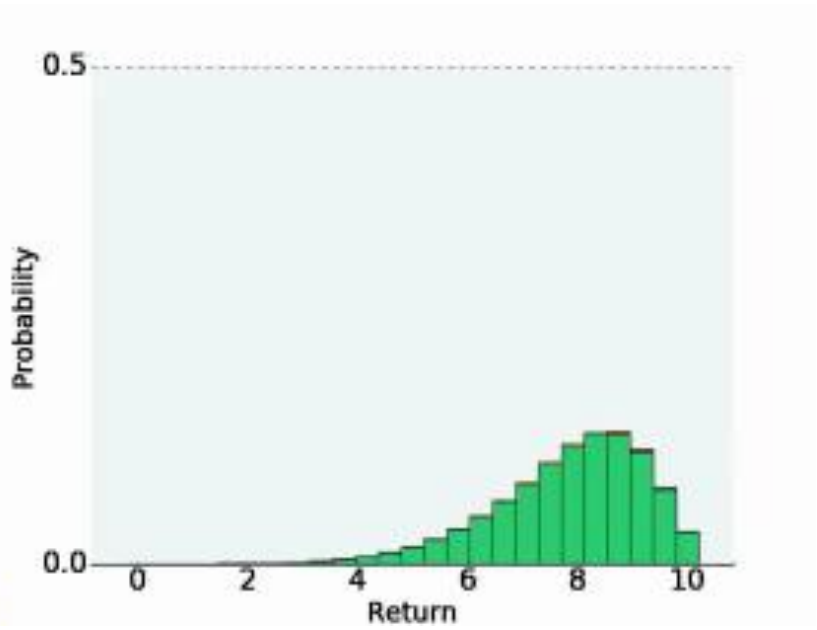
$m_u \leftarrow m_u + p_j(x_{t+1}, a^*)(b_j - l)$

end for

output $-\sum_i m_i \log p_i(x_t, a_t)$ # Cross-entropy loss

Demo

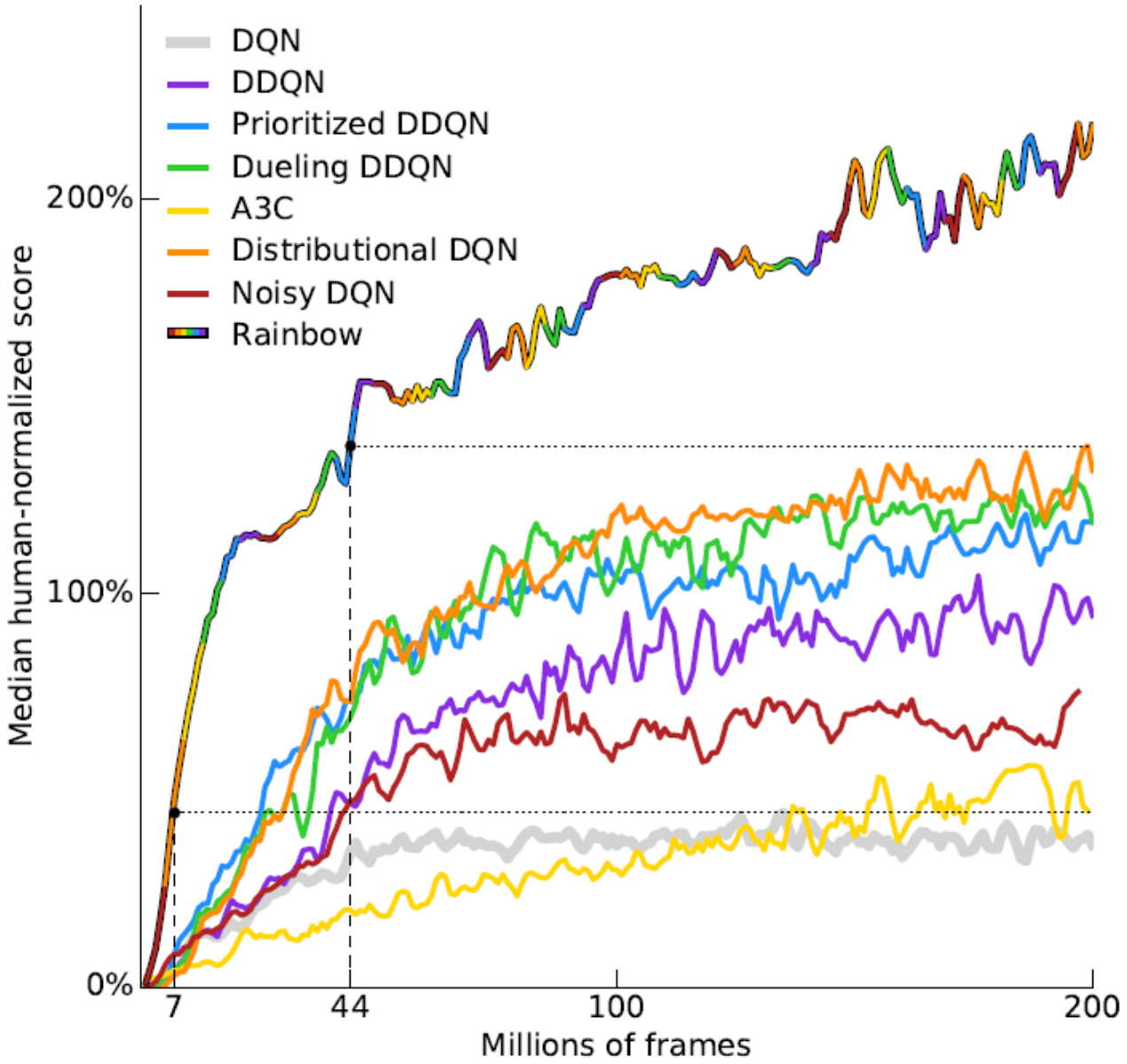




https://youtu.be/yFBwyPuO2Vg

Rainbow

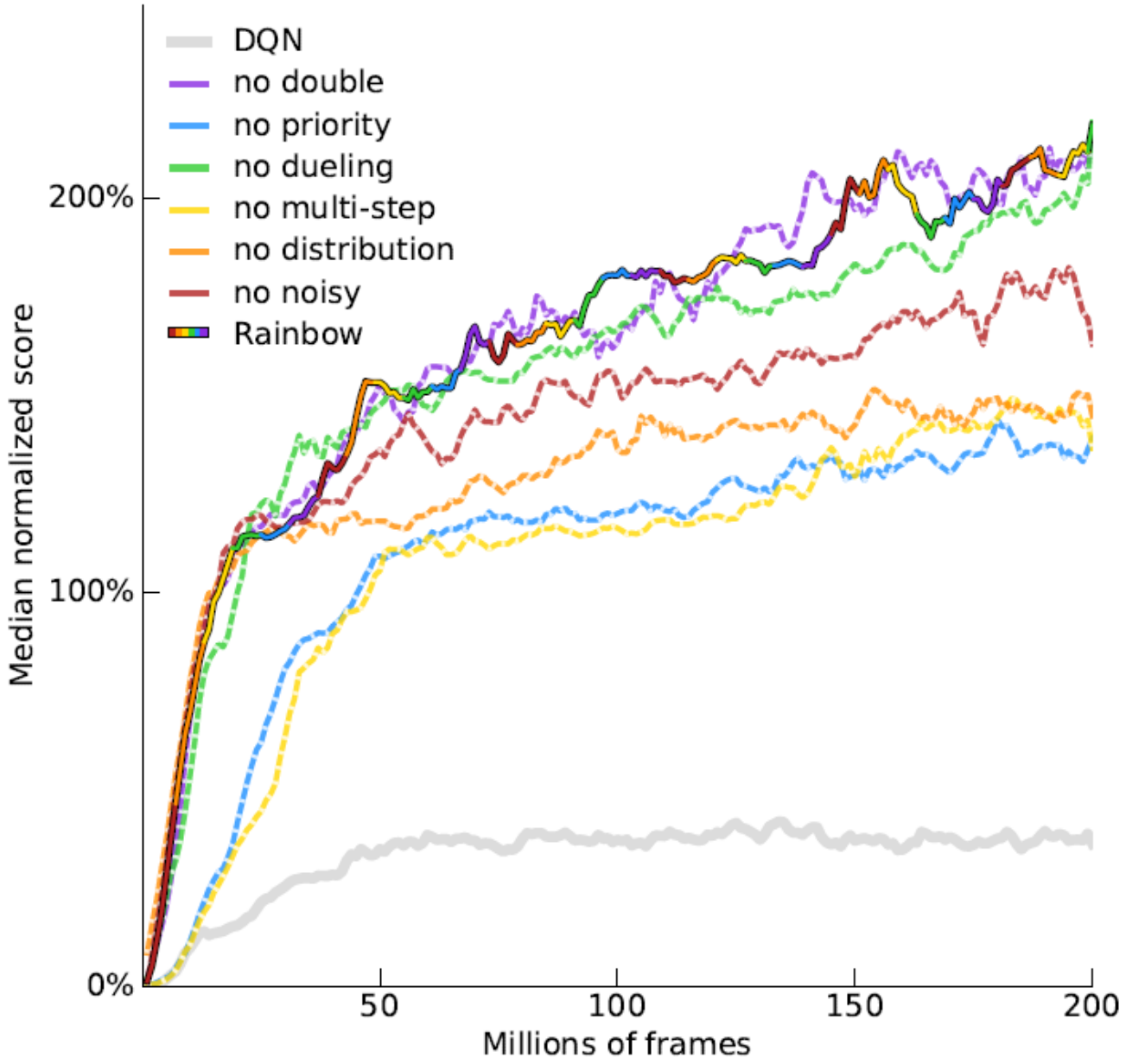
- Median human-normalized performance across 57 Atari games. We compare our integrated agent (rainbow colored) to DQN (grey) and six published baselines. Note that we match DQN’ s best performance after 7M frames, surpass any baseline within 44M frames, and reach sub stantially improved final performance. Curves are smoothed with a moving average over 5 points



Rainbow: combining improvements in deep reinforcement learning, AAAI’18

Rainbow

- Median human-normalized performance across 57 Atari games, as a function of time. We compare our in tegrated agent (rainbow-colored) to DQN (gray) and to six different ablations (dashed lines). Curves are smoothed with a moving average over 5 points.



Rainbow: combining improvements in deep reinforcement learning, AAAI'18

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

Solution 2

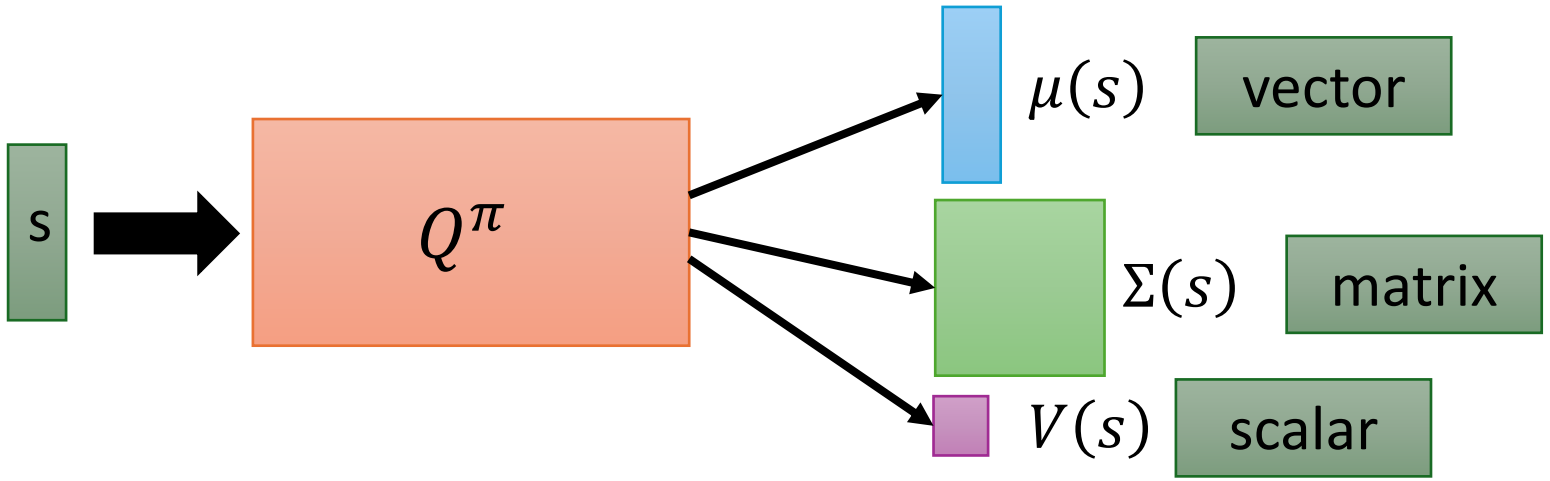
Using gradient ascent to solve the optimization problem.

Continuous Actions

Solution 3 Design a network to make the optimization easy.

$$Q(s,a) = A(s,a) + V(s)$$
$$= -\frac{1}{2}\big(a - \mu(s)\big)^T \Sigma(s)\big(a - \mu(s)\big) + V(s)$$

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



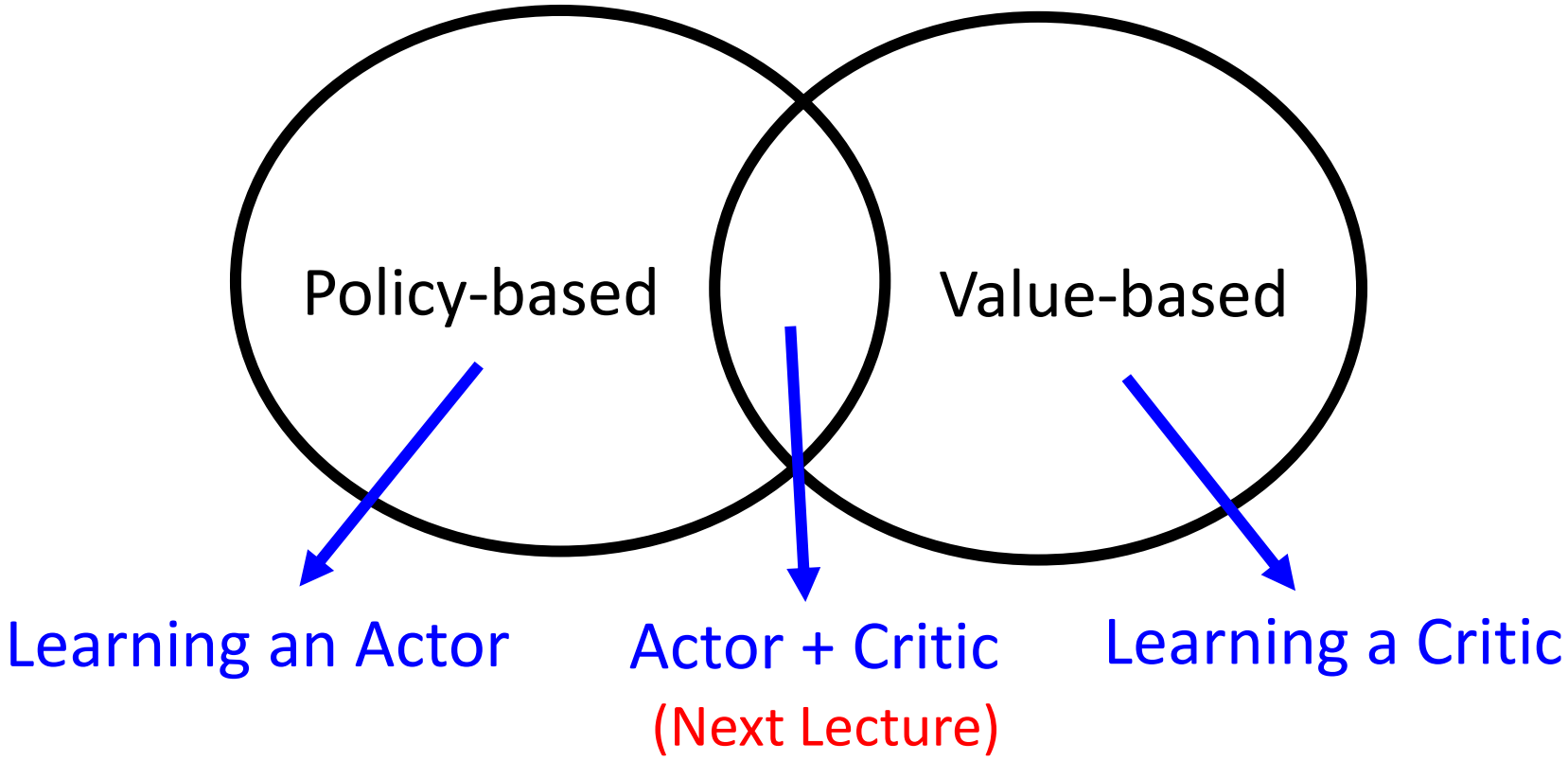
Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates, ICRA'17

Continuous Actions

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

Continuous Actions

Solution 4 Don't use Q-learning



Reference

- Shusen Wang, Deep Reinforcement Learning, https://github.com/wangshusen/DRL
- Emma Brunskill, CS234 Reinforcement Learning.
- Bolei Zhou, Intro to Reinforcement Learning, https://github.com/zhoubolei/introRL
- Hung-yi Lee, Deep Reinforcement Learning: Q-learning.
- Sergey Levine, CS 285: Deep Reinforcement Learning, Decision Making, and Control