STA303: Artificial Intelligence

Deep Reinforcement Learning

Fang Kong

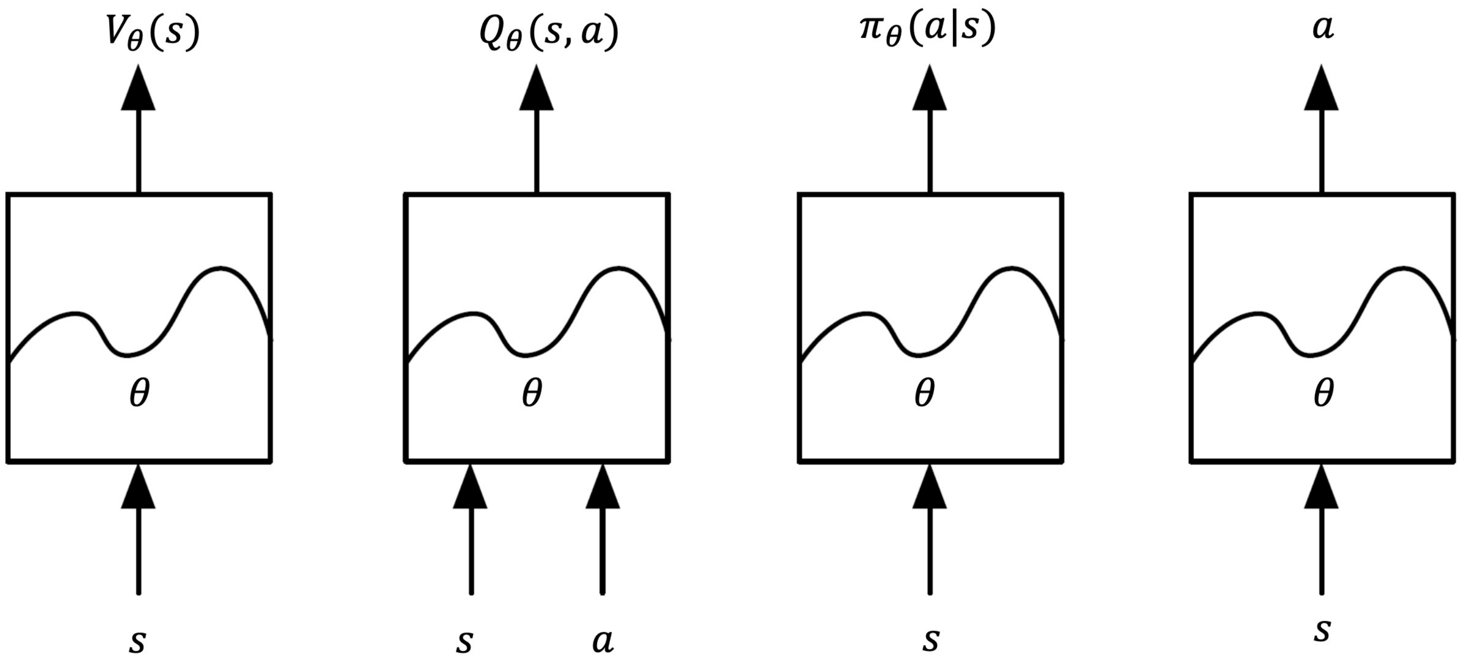
https://fangkongx.github.io/

Part of slide credits: Weinan Zhang

Outline

- Deep RL – Value methods
- Deep RL – Policy methods

Function approximation for value and policy



- What if we use deep neural networks directly to approximate these functions?

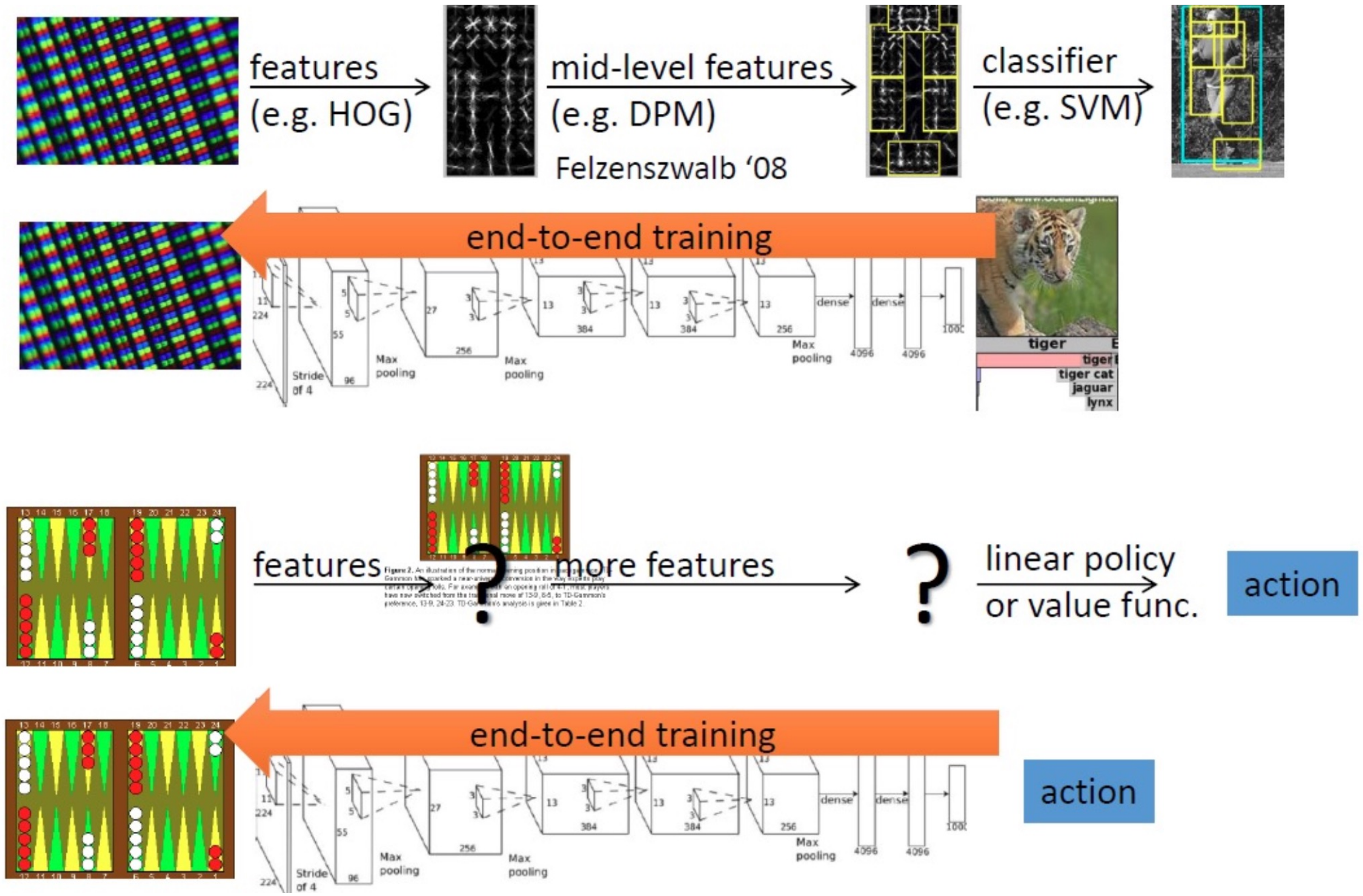
End-to-end reinforcement learning

Traditional computer vision

Deep learning

Traditional RL

Deep RL



- Deep RL enables RL algorithms to solve complex tasks in an end-to-end manner.

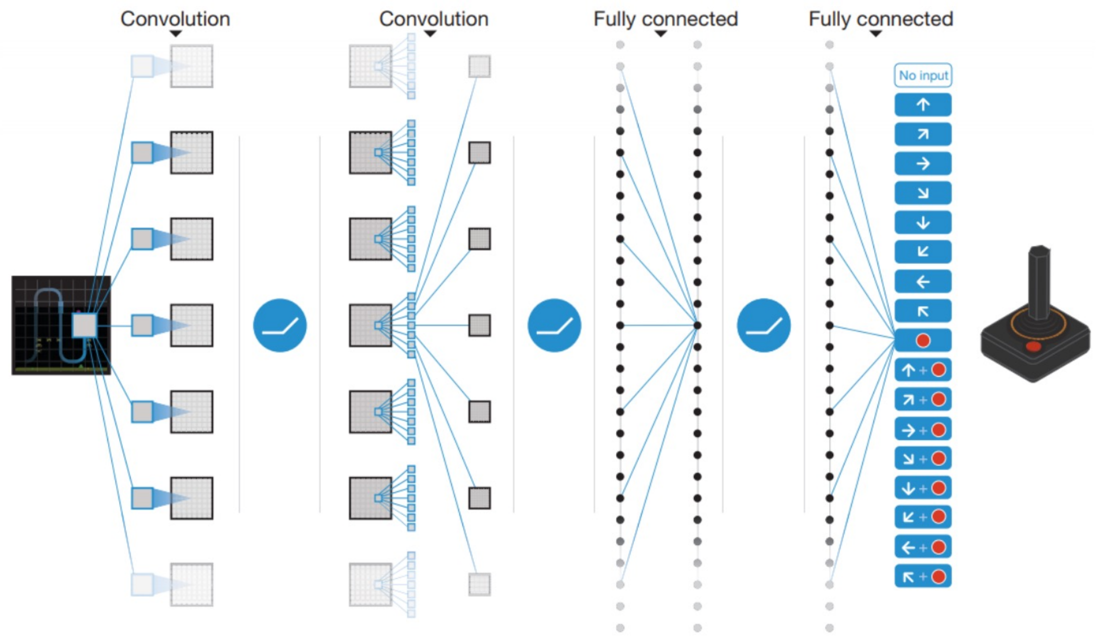
Slide from Sergey Levine. http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-1.pdf

Deep RL

- New challenges when we combine deep learning with RL?
  - Value functions and policies become deep neural networks
  - High-dimensional parameter space
  - Difficult to train stably
  - Prone to overfitting
  - Requires large amounts of data
  - High computational cost
  - CPU-GPU workload balance

Value methods: DQN

- Deep Q-Network (DQN)
  - Uses a deep neural network to approximate Q(s,a)
    - → Replaces the Q-table with a parameterized function for scalability
  - The network takes state s as input, outputs Q-values for all actions a simultaneously



Volodymyr Mnih, Koray Kavukcuoglu, David Silver et al. Playing Atari with Deep Reinforcement Learning. NIPS 2013 workshop.

DQN (cont.)

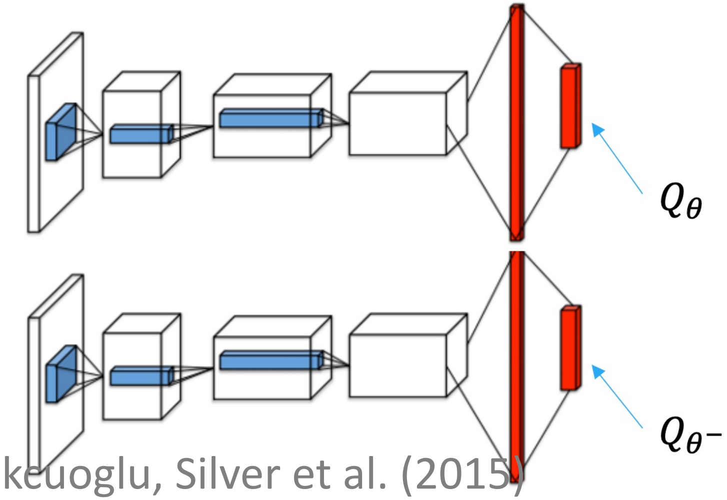
- Intuition: Use a deep neural network to approximate Q(s,a)
  - Instability arises in the learning process
    - Samples $\{(s_t, a_t, s_{t+1}, r_t)\}$ are collected sequentially and do not satisfy the i.i.d. assumption
    - Frequent updates of Q(s,a) cause instability
- Solutions: Experience replay
  - Store transitions $e_t = (s_t, a_t, s_{t+1}, r_t)$ in a replay buffer D
    Sample uniformly from D to reduce sample correlation
  - Dual network architecture: Use an evaluation network and a target network for improved stability

“Human-Level Control Through Deep Reinforcement Learning”, Mnih, Kavukcuoglu, Silver et al. (2015)

Target network

- Target network $Q_{\theta^-}(s, a)$
  - Maintains a copy of the Q-network with older parameters θ^-
  - Parameters θ^- are updated periodically (every C steps) to match the evaluation network
- Loss Function (at iteration i)

$$L_i(\theta_i) = \mathbb{E}_{s_t, a_t, s_{t+1}, r_t, p_t \sim D}\left[\frac{1}{2}\omega_t\left(r_t + \gamma \max_{a'} Q_{\theta_i^-}(s_{t+1}, a') - Q_{\theta_i}(s_t, a_t)\right)^2\right]$$



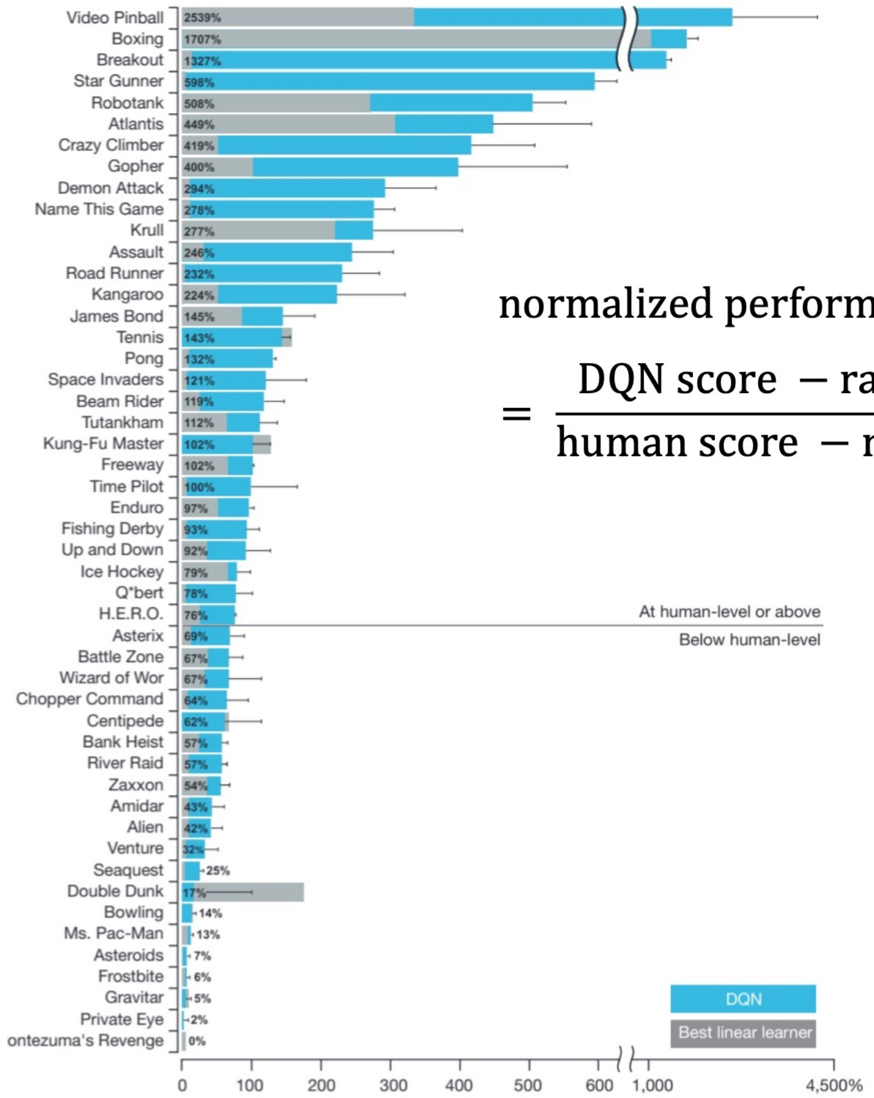
“Human-Level Control Through Deep Reinforcement Learning”, Mnih, Kavukcuoglu, Silver et al. (2015)

DQN training procedure

- Collect transitions using an ε-greedy exploration policy
  - Store $\{(s_t, a_t, s_{t+1}, r_t)\}$ into the replay buffer
- Sample a minibatch of k transitions from the buffer
- Update networks:
  - Compute the target using the sampled transitions
  - Update the evaluation network Q_θ
  - Every C steps, synchronize the target network Q_{θ^-} with the evaluation network

"Human-Level Control Through Deep Reinforcement Learning", Mnih, Kavukcuoglu, Silver et al. (2015)

DQN performance in Atari games



$$\text{normalized performance} = \frac{\text{DQN score} - \text{random play score}}{\text{human score} - \text{random play score}}$$

The performance of DQN is normalized with respect to a professional human games tester (that is, 100% level)

"Human-Level Control Through Deep Reinforcement Learning", Mnih, Kavukcuoglu, Silver et al. (2015)

Overestimation in Q-Learning

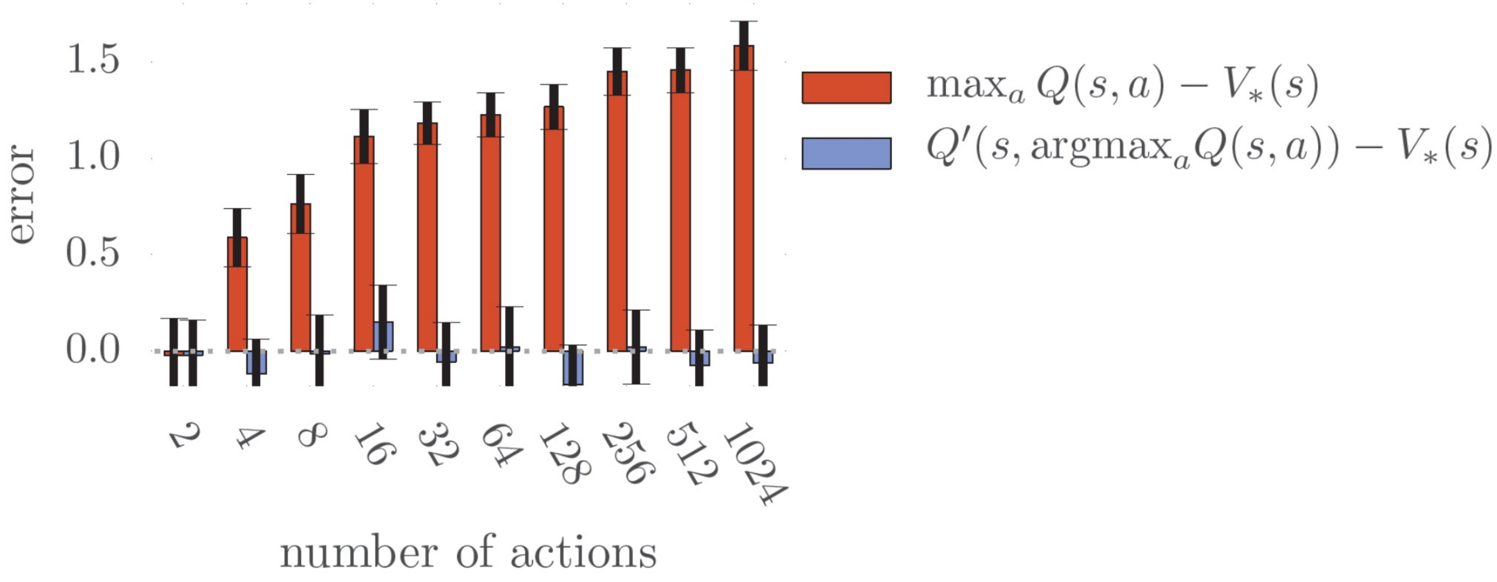
- Q-function overestimation
  - The target value is computed as: $y_t = r_t + \gamma \max_{a'} Q_\theta(s_{t+1}, a')$
  - The max operator leads to increasingly larger Q-values, potentially exceeding the true value
- Cause of overestimation

$$\max_{a' \in A} Q_{\theta'}(s_{t+1}, a') = Q_{\theta'}(s_{t+1}, \arg\max_{a'} Q_{\theta'}(s_{t+1}, a'))$$

  - The chosen action might be overestimated due to Q-function error

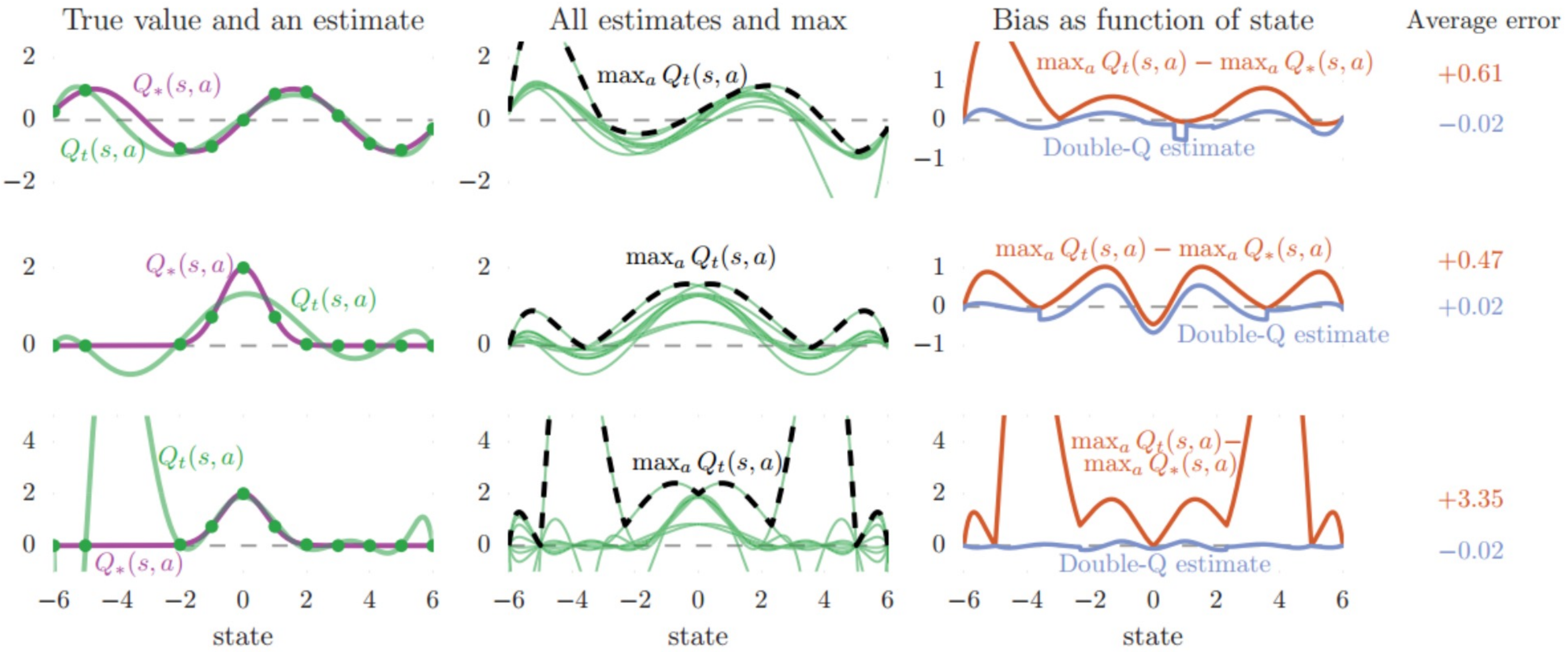
Degree of overestimation in DQN

- Overestimation increases with the number of candidate actions



- A separately trained Q'-function is used as a reference

Overestimation example in DQN



- Setup: The x-axis represents states, and each plot includes 10 candidate actions. The purple curve denotes the true Q-value function, the green dots are training data points, and the green lines are the fitted Q-value estimates.
- The middle column shows the estimated values $Q_t(s,a)$ for all 10 actions. After applying the max operator, the results deviate significantly from the true values $Q_*(s,a)$.

Double DQN

- Uses two separate networks for action selection and value estimation, respectively.

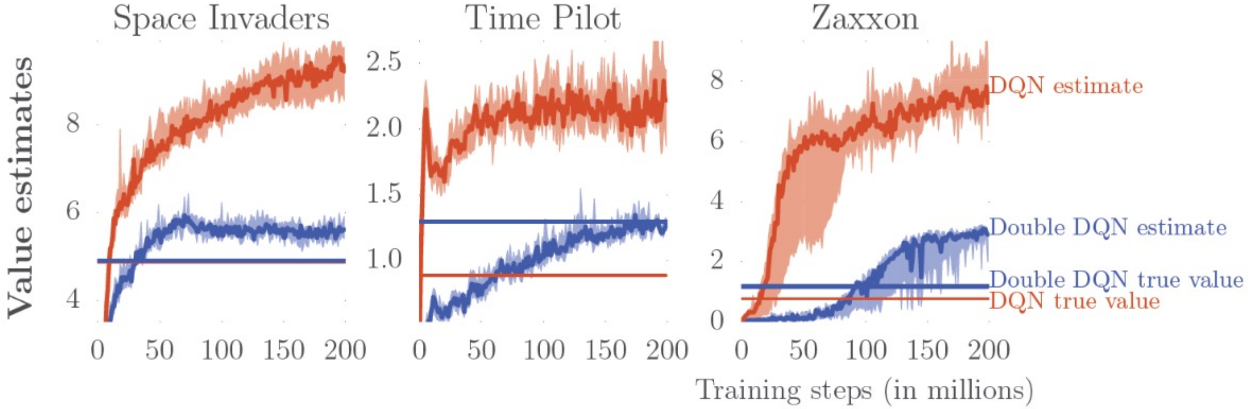
DQN $$y_t = r_t + \gamma Q_\theta(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a'))$$

Double DQN $$y_t = r_t + \gamma Q_{\theta'}(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a'))$$

"Double Reinforcement Learning with Double Q-Learning", van Hasselt et al. (2016)

Experimental results in the Atari environment

- Value estimation error

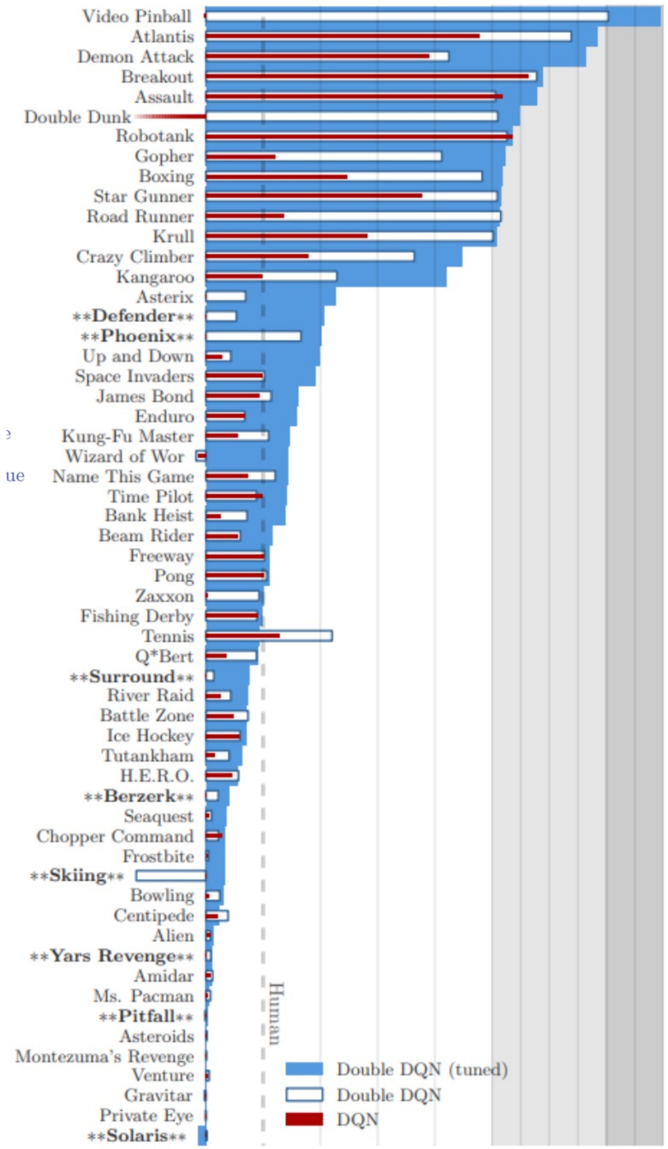


- Atari game performance

	no ops		human starts		
	DQN	DDQN	DQN	DDQN	DDQN (tuned)
Median	93%	**115%**	47%	88%	**117%**
Mean	241%	**330%**	122%	273%	**475%**

normalized performance

$$= \frac{\text{DQN score} - \text{random play score}}{\text{human score} - \text{random play score}}$$



Dueling DQN

- Assume the action-value function follows a distribution:

$$Q(s,a) \sim \mathcal{N}(\mu,\sigma)$$

- Then: $V(s) = \mathbb{E}[Q(s,a)] = \mu$ $\qquad Q(s,a) = \mu + \boxed{\varepsilon(s,a)}$

- How do we describe $\varepsilon(s,a)$?

$$\varepsilon(s,a) = Q(s,a) - V(s)$$

  - This term is also known as the Advantage function

"Dueling Network Architectures for Deep Reinforcement Learning", Wang et al. (2016)

Dueling DQN (cont.)

- Advantage function

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

$$Q^{\pi}(s,a) = \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$$

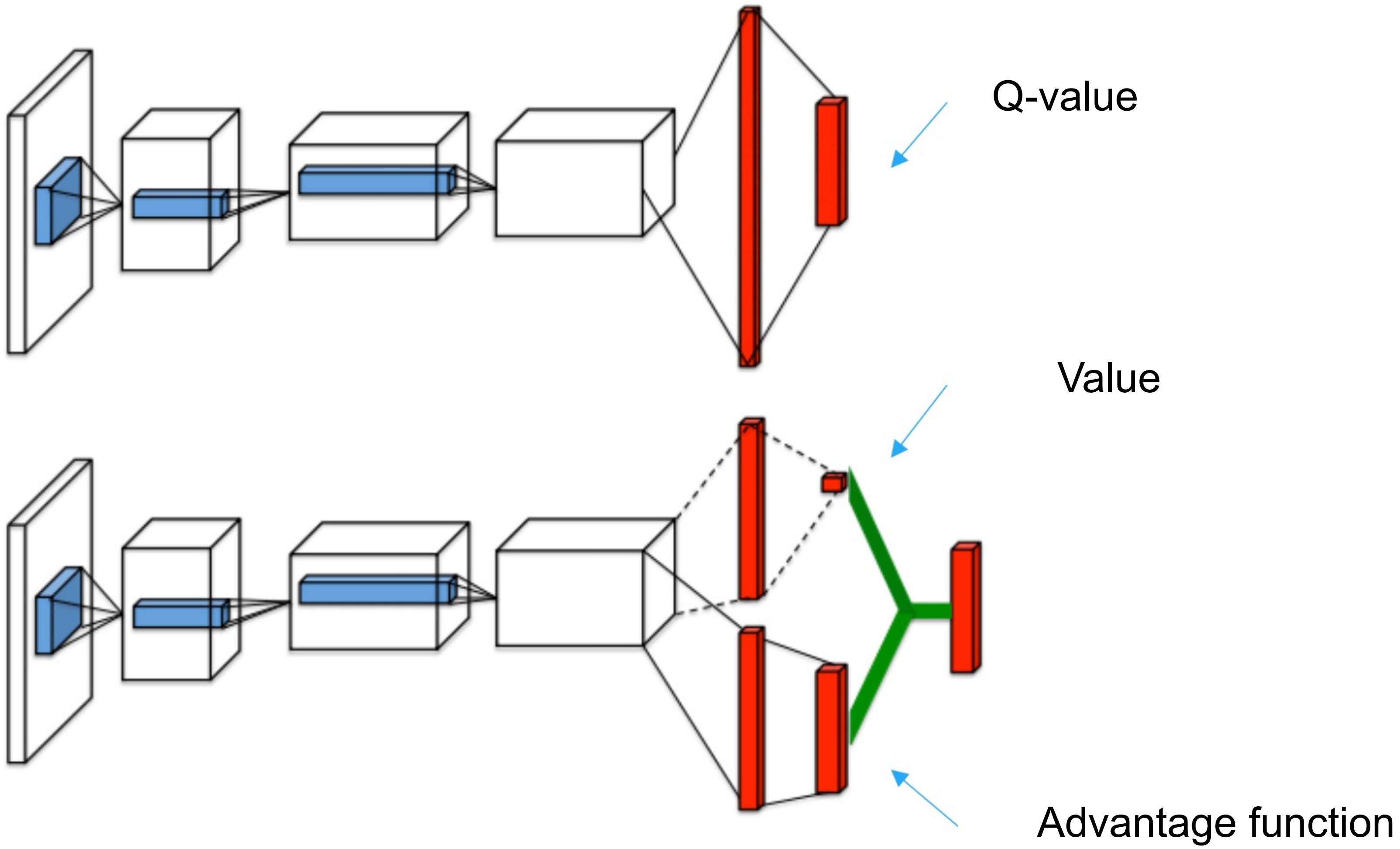
$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi(s)}[Q^{\pi}(s,a)]$$

- Different forms of advantage aggregation

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a'\in|A|} A(s,a';\theta,\alpha)\right)$$

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)\right)$$

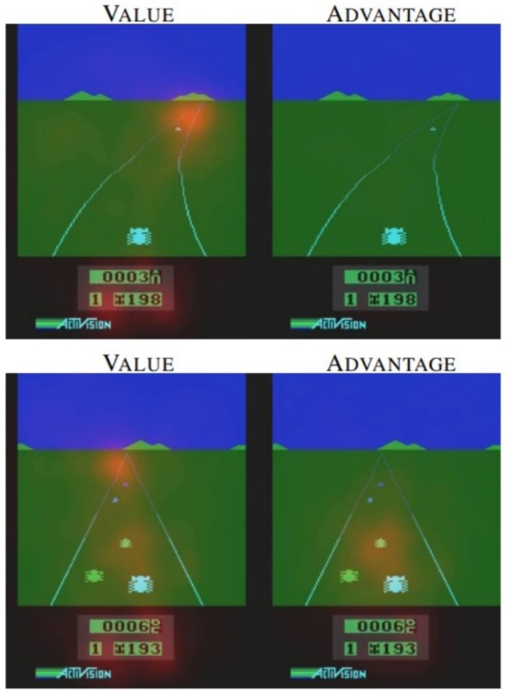
Network structure

- 

Advantages of Dueling DQN

- Effective for states weakly correlated with actions
- More efficient learning of the state-value function
  - The value stream V(s) is shared across all actions, allowing the network to generalize better across actions

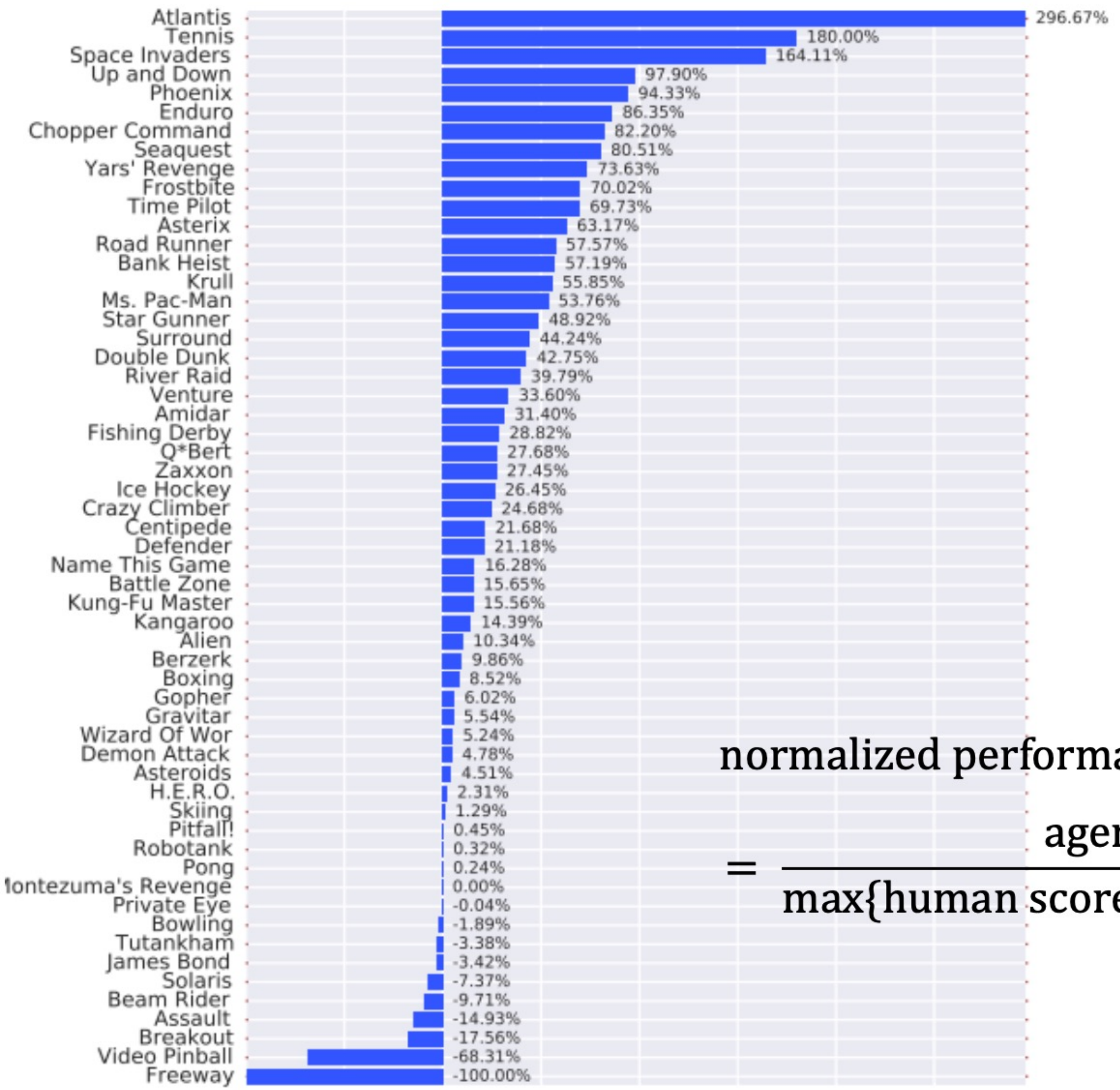
saliency maps



•The value stream allows the agent to evaluate how good a state is without considering the specific action taken.

•The advantage stream emphasizes action-specific importance: for instance, it can learn to focus more when an obstacle (e.g., a car) appears in front of the agent, thereby guiding more precise action selection.

Experimental results in the Atari environment I

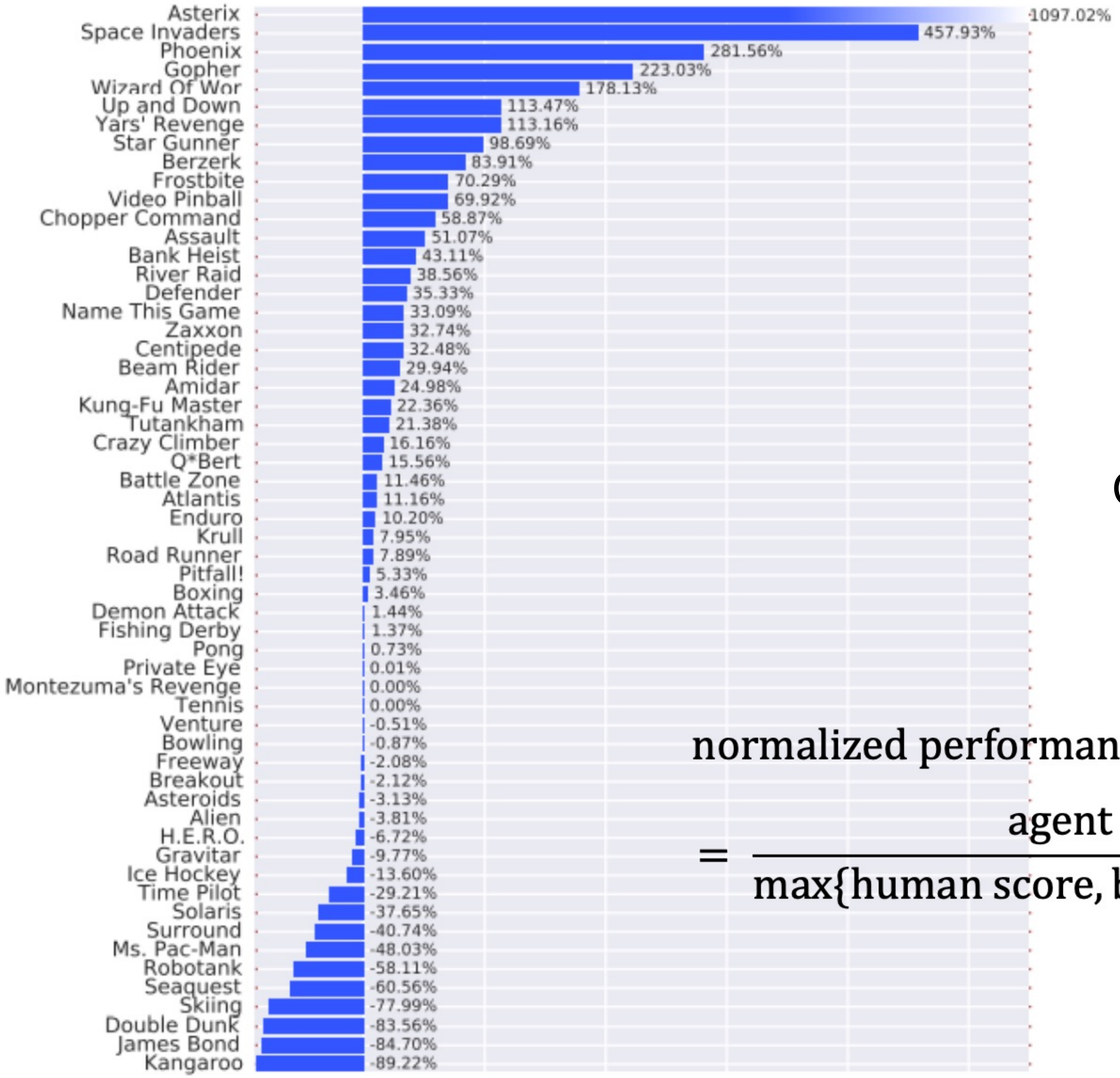


Compared with DQN

normalized performance

$$= \frac{\text{agent score} - \text{baseline score}}{\max\{\text{human score}, \text{baseline score}\} - \text{random play score}}$$

Experimental results in the Atari environment II



Compared with Double DQN

normalized performance

$$= \frac{\text{agent score} - \text{baseline score}}{\max\{\text{human score}, \text{baseline score}\} - \text{random play score}}$$

Deep RL – Policy-based methods

Review: The policy gradient theorem

- The policy gradient theorem generalizes the derivation of likelihood ratios to the multi-step MDP setting.
- It replaces the immediate reward r_t with the expected long-term return $Q^{\pi}(s,a)$.

$$\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_{\pi_\theta}\left[\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} Q^{\pi_\theta}(s,a)\right]$$

Policy Gradient in a Single-Step MDP

- Consider a simple single-step Markov Decision Process (MDP)
  - The initial state is drawn from a distribution: $s \sim d(s)$
  - The process terminates after one action, yielding a reward r_{sa}
- Expected Value of the Policy

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r] = \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(a|s) r_{sa}$$

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{s \in S} d(s) \sum_{a \in A} \frac{\partial \pi_\theta(a|s)}{\partial \theta} r_{sa}$$

Policy network gradient

- For stochastic policies, the probability of selecting an action is typically modeled using a softmax function:

$$\pi_\theta(a|s) = \frac{e^{f_\theta(s,a)}}{\sum_{a'} e^{f_\theta(s,a')}}$$

  - $f_\theta(s, a)$ is a score function (e.g., logits) for the state-action pair
  - Parameterized by θ, often realized via a neural network

- Gradient of the log-form

$$\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} = \frac{\partial f_\theta(s,a)}{\partial \theta} - \frac{1}{\sum_{a'} e^{f_\theta(s,a')}} \sum_{a''} e^{f_\theta(s,a'')} \frac{\partial f_\theta(s,a'')}{\partial \theta}$$

$$= \frac{\partial f_\theta(s,a)}{\partial \theta} - \mathbb{E}_{a' \sim \pi_\theta(a'|s)}\left[\frac{\partial f_\theta(s,a')}{\partial \theta}\right]$$

Policy network gradient (cont.)

- Gradient of the log-form

$$\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} = \frac{\partial f_\theta(s,a)}{\partial \theta} - \mathbb{E}_{a' \sim \pi_\theta(a'|s)}\left[\frac{\partial f_\theta(s,a')}{\partial \theta}\right]$$

- Gradient of the policy network

$$\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_{\pi_\theta}\left[\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} Q^{\pi_\theta}(s,a)\right]$$

$$= \mathbb{E}_{\pi_\theta}\left[\left(\underbrace{\frac{\partial f_\theta(s,a)}{\partial \theta}}_{\text{Back propagation}} - \mathbb{E}_{a' \sim \pi_\theta(a'|s)}\left[\underbrace{\frac{\partial f_\theta(s,a')}{\partial \theta}}_{\text{Back propagation}}\right]\right) Q^{\pi_\theta}(s,a)\right]$$

Back propagation Back propagation

Comparison: DQN v.s. Policy gradient

- Q-Learning:
  - Learns a Q-value function $Q_\theta(s, a)$ parameterized by θ
  - Objective: Minimize the TD error

$$J(\theta) = \mathbb{E}_{\pi'}\left[\frac{1}{2}\left(r_t + \gamma \max_{a'} Q_{\theta'}(s_{t+1}, a') - Q_\theta(s_t, a_t)\right)^2\right]$$

$$\theta \leftarrow \theta - \alpha \frac{\partial J(\theta)}{\partial \theta}$$

$$= \theta + \alpha \mathbb{E}_{\pi'}\left[\left(r_t + \gamma \max_{a'} Q_{\theta'}(s_{t+1}, a') - Q_\theta(s_t, a_t)\right) \frac{\partial Q_\theta(s, a)}{\partial \theta}\right]$$

Comparison: DQN v.s. Policy gradient

- Q-Learning:
  - Learns a Q-value function $Q_\theta(s, a)$ parameterized by θ
  - Objective: Minimize the TD error
- Policy gradient
  - Learns a policy $\pi_\theta(a \mid s)$ directly, parameterized by θ
  - Objective: Maximize the expected return directly

$$\max_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[Q^{\pi_\theta}(s, a)]$$

$$\theta \leftarrow \theta + \alpha \frac{\partial J(\theta)}{\partial \theta} = \theta + \alpha \mathbb{E}_{\pi_\theta}\left[\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} Q^{\pi_\theta}(s, a)\right]$$