南方科技大学

MAT8034: Machine Learning

Deep Reinforcement Learning

Fang Kong

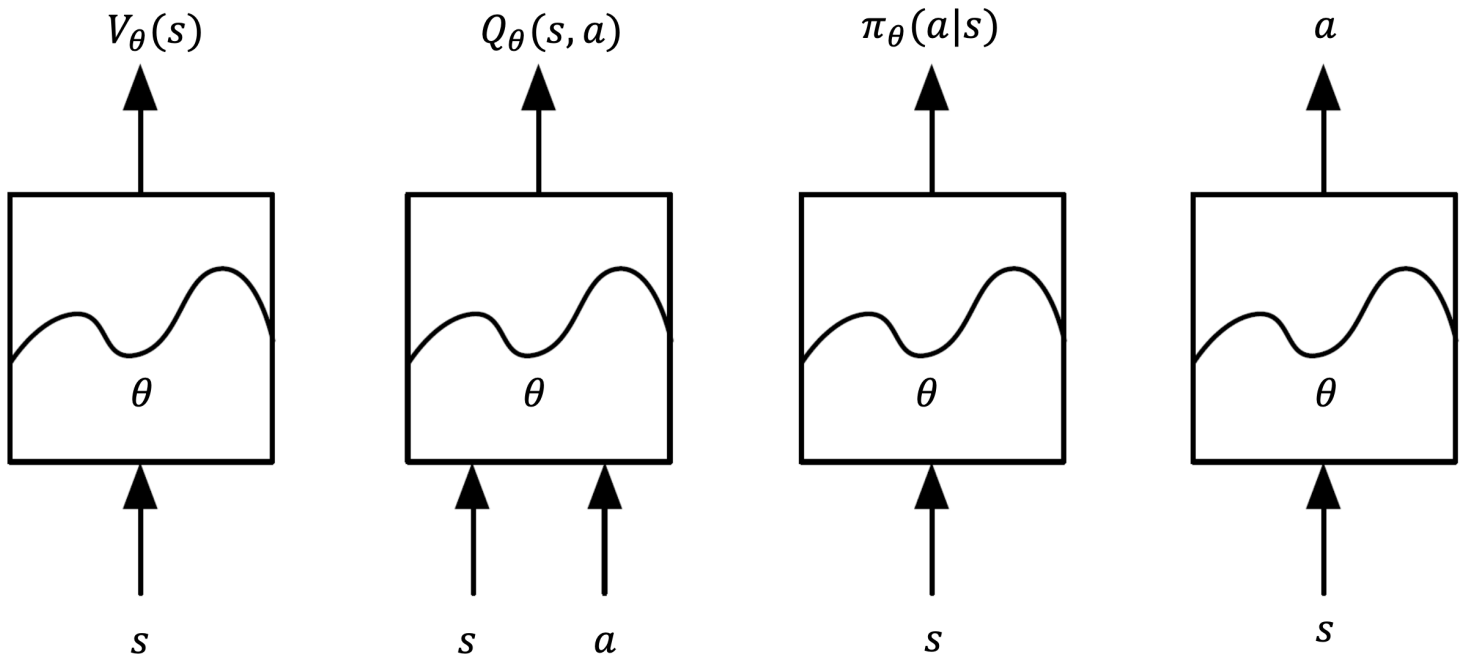
https://fangkongx.github.io/Teaching/MAT8034/Spring2026/index.html

Part of slide credits: Weinan Zhang

Outline

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

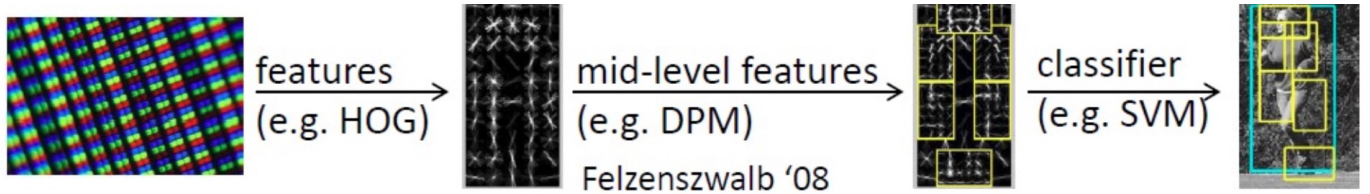
Function approximation for value and policy



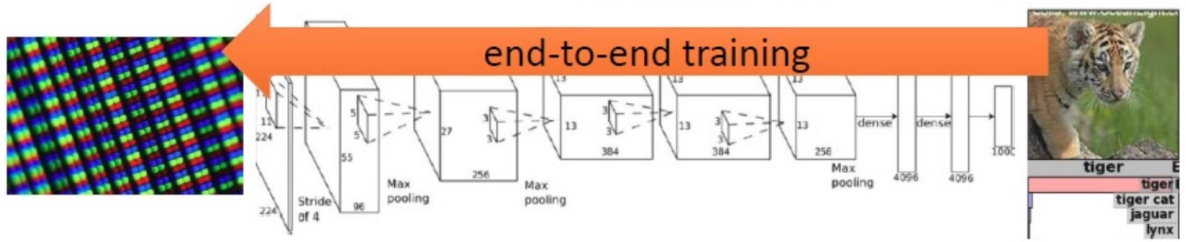
- What if we use deep neural networks directly to approximate these functions?

End-to-end reinforcement learning

Traditional computer vision

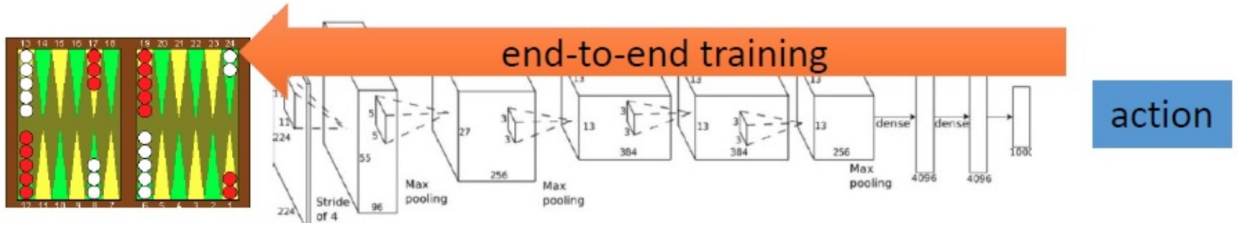


Deep learning



Traditional RL

Deep RL



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

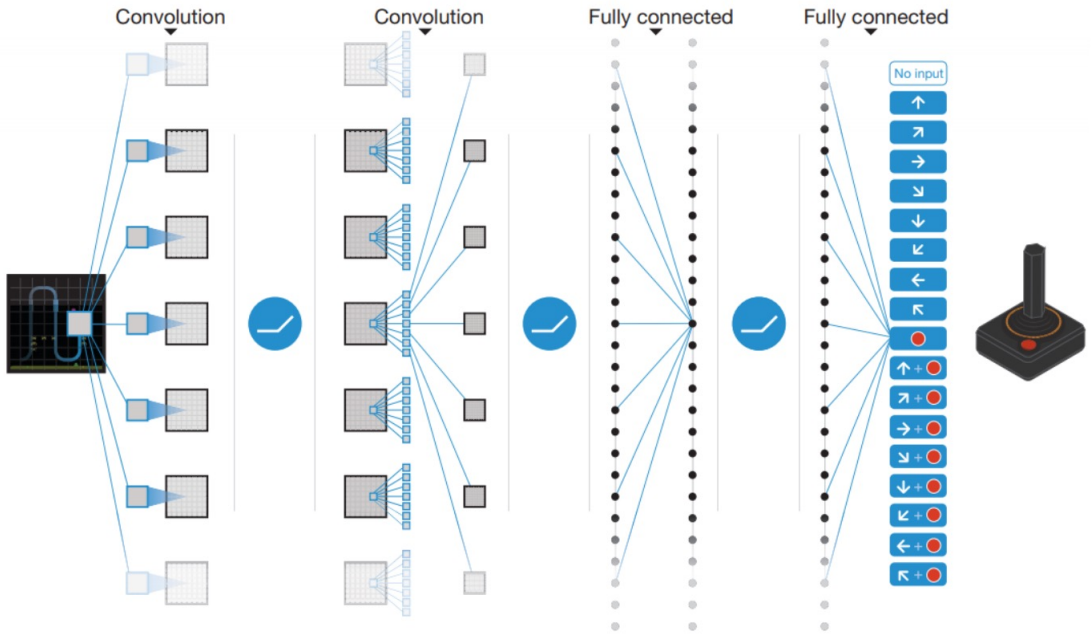
Slide from Sergey Levine. http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-1.pdf

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
  - Requires large amounts of data
  - Data dependence
  - High computational cost
  - ……

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: Loss function

- Intuition: Use a deep neural network to approximate Q(s,a)
- Loss Function?

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

DQN (cont.)

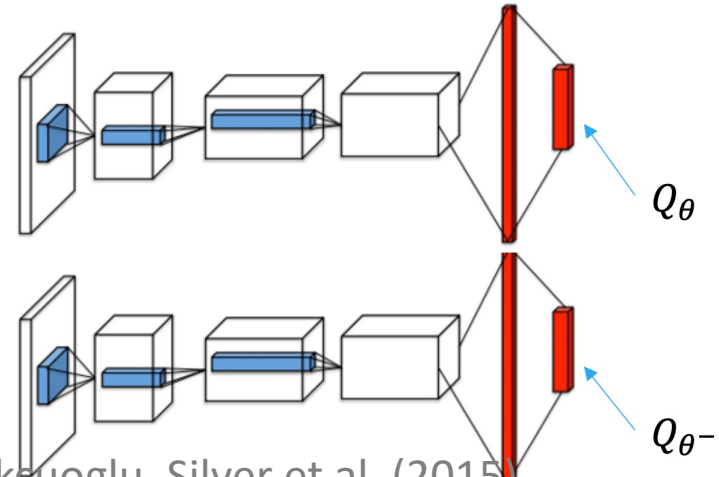
- Instability arises in the learning process
  - Samples $\{(s_t, a_t, s_{t+1}, r_t)\}$ are collected sequentially and do not satisfy the i.i.d. assumption
  - Frequent updates of Q(s,a) cause instability
- Solutions: Experience replay
  - Store transitions $e_t = (s_t, a_t, s_{t+1}, r_t)$ in a replay 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)

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



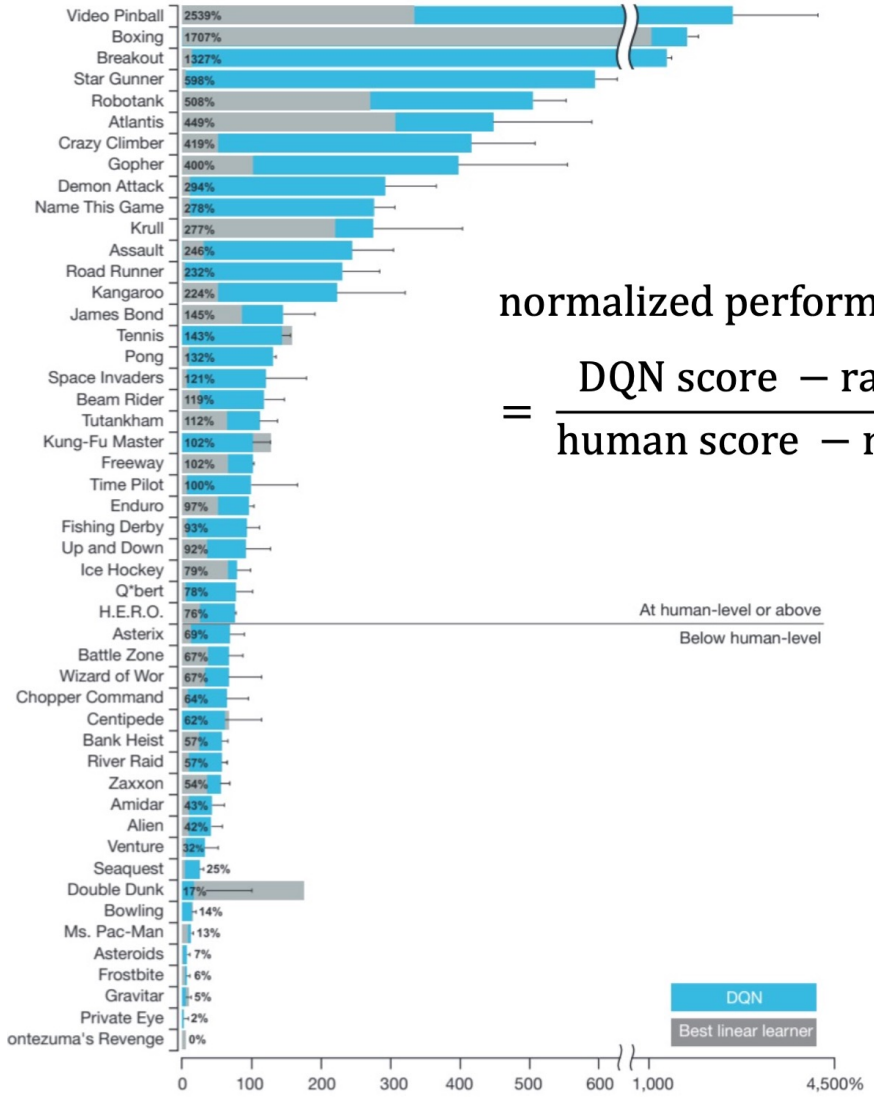
"Human-Level Control Through Deep Reinforcement Learning", Mnih, Kavukcuoglu, Silver et al. (2015)

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

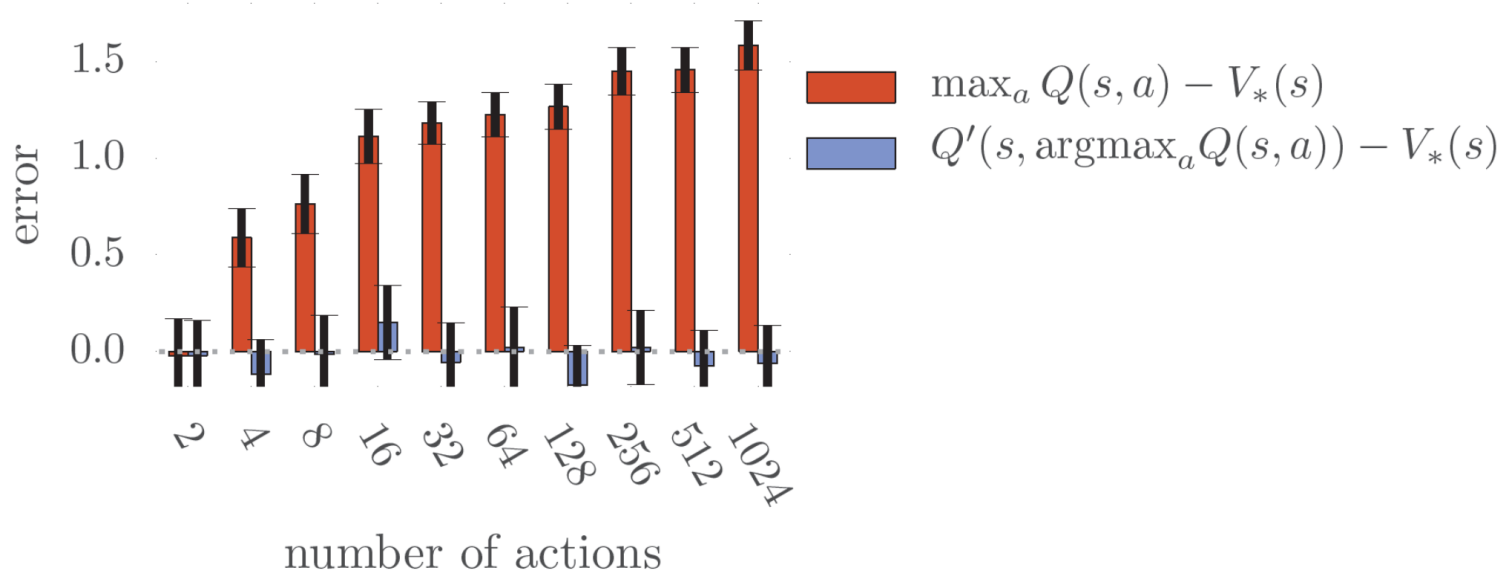
- Q-function overestimation
  - The target value is computed as: $y_t = r_t + \gamma \max_{a'} Q_\theta(s_{t+1}, a')$
  - The max operator leads to increasingly larger Q-values, potentially exceeding the true value
- Cause of overestimation

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

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

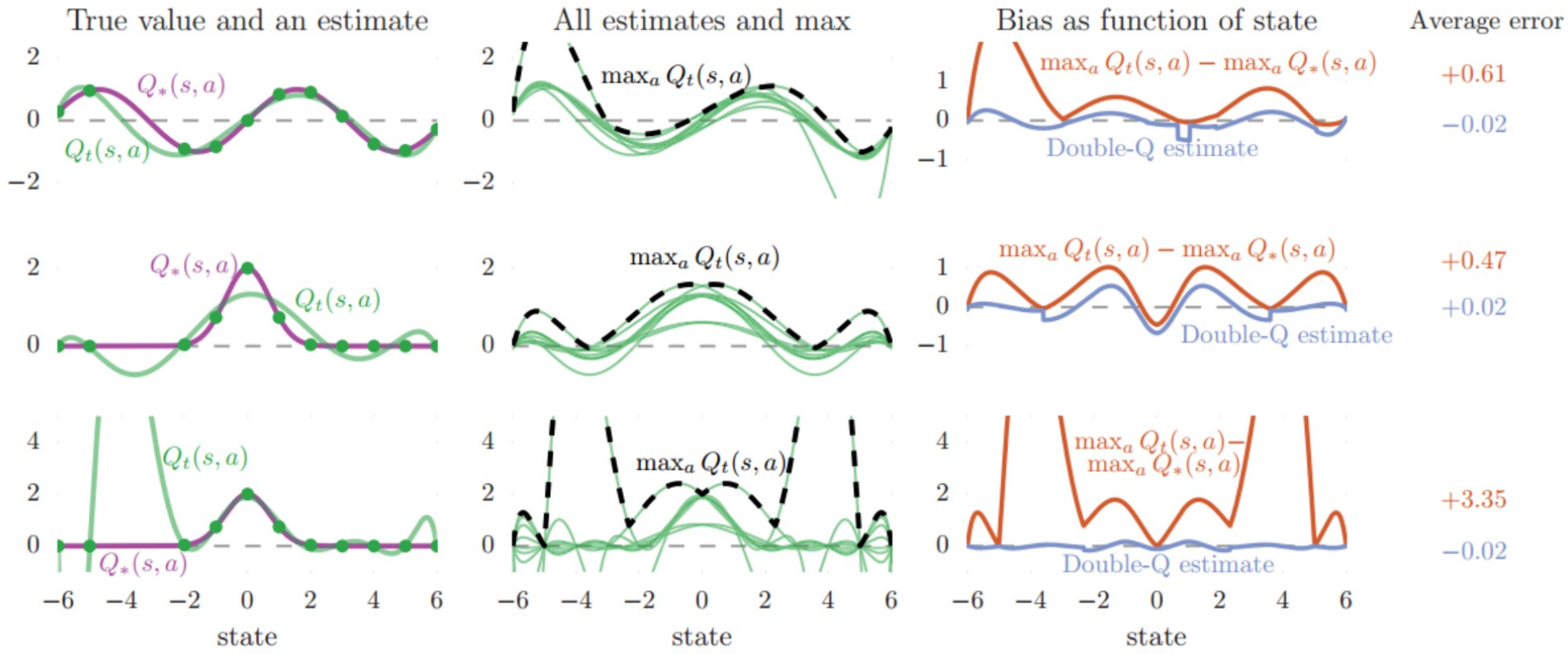
Degree of overestimation in DQN

- 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.

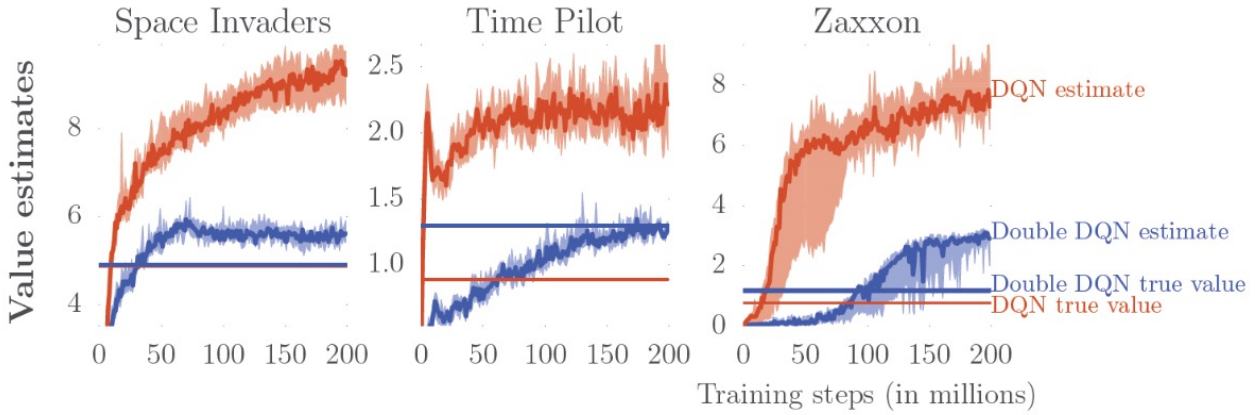
$$\text{DQN} \quad y_t = r_t + \gamma Q_\theta(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a'))$$

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

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

Experimental results in the Atari environment

- Value estimation error

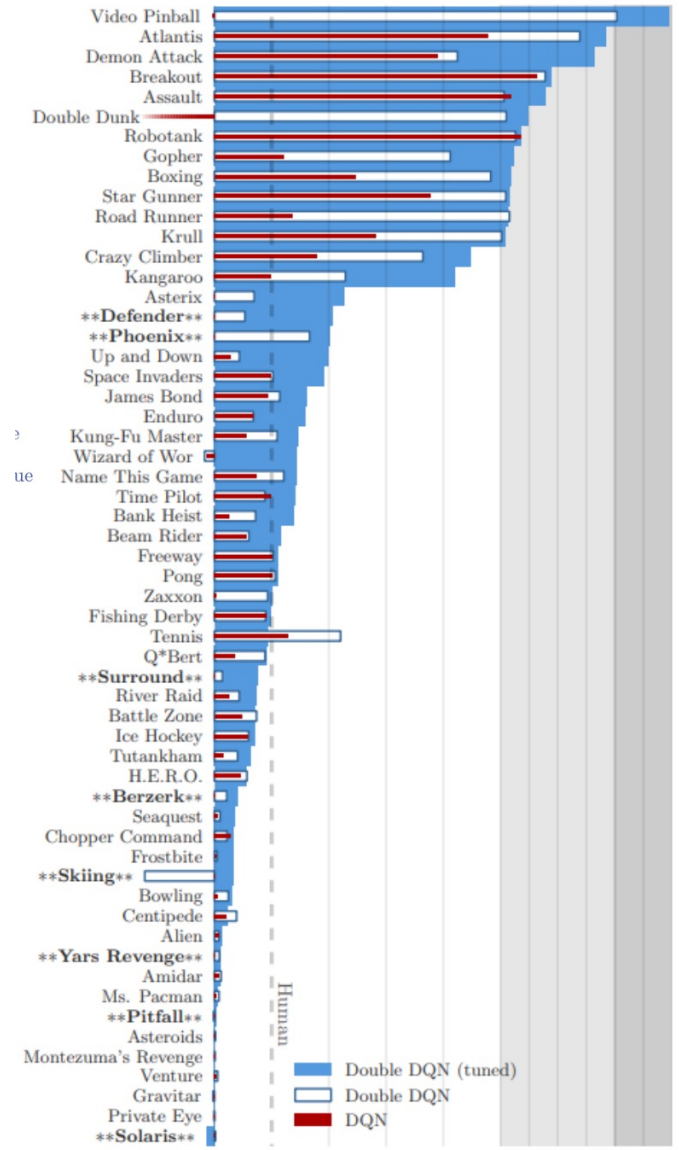


- Atari game performance

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

- 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

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

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

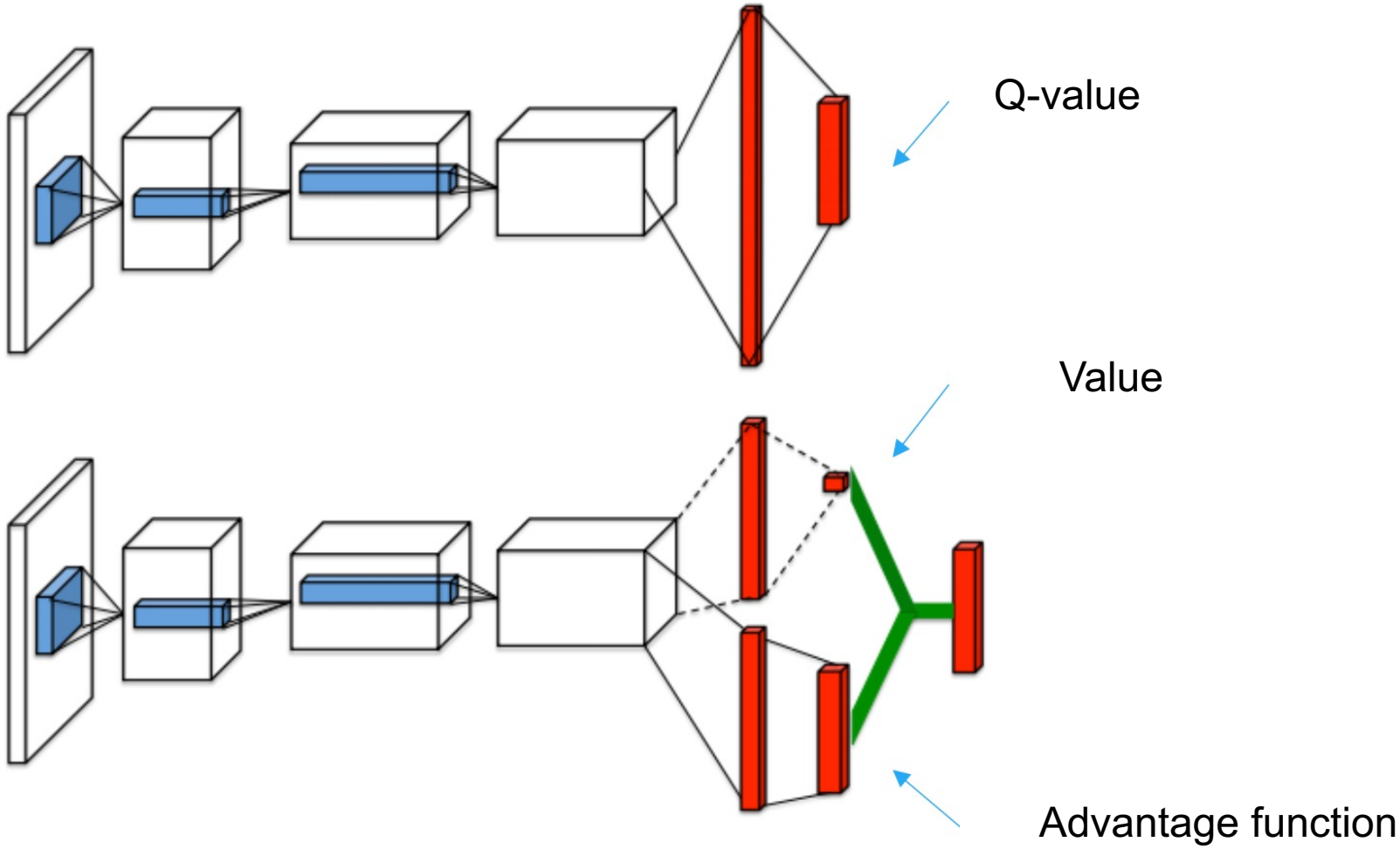
$$V^{\pi}(s) = \mathbb{E}_{a \sim \pi(s)}[Q^{\pi}(s,a)]$$

- Different forms of advantage aggregation

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

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

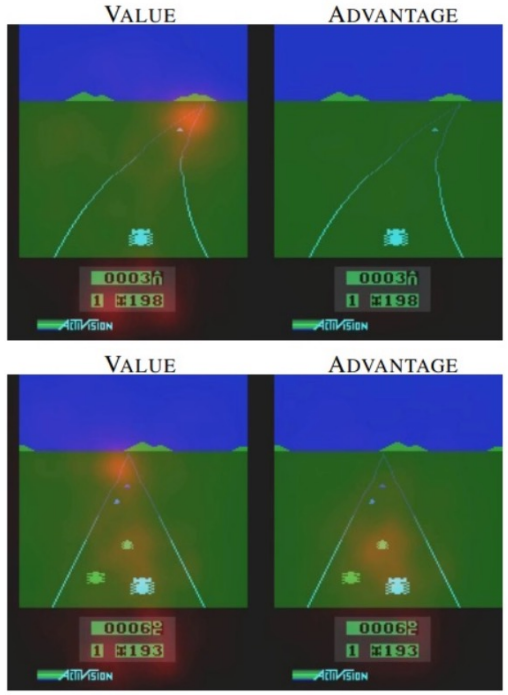
Network structure



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

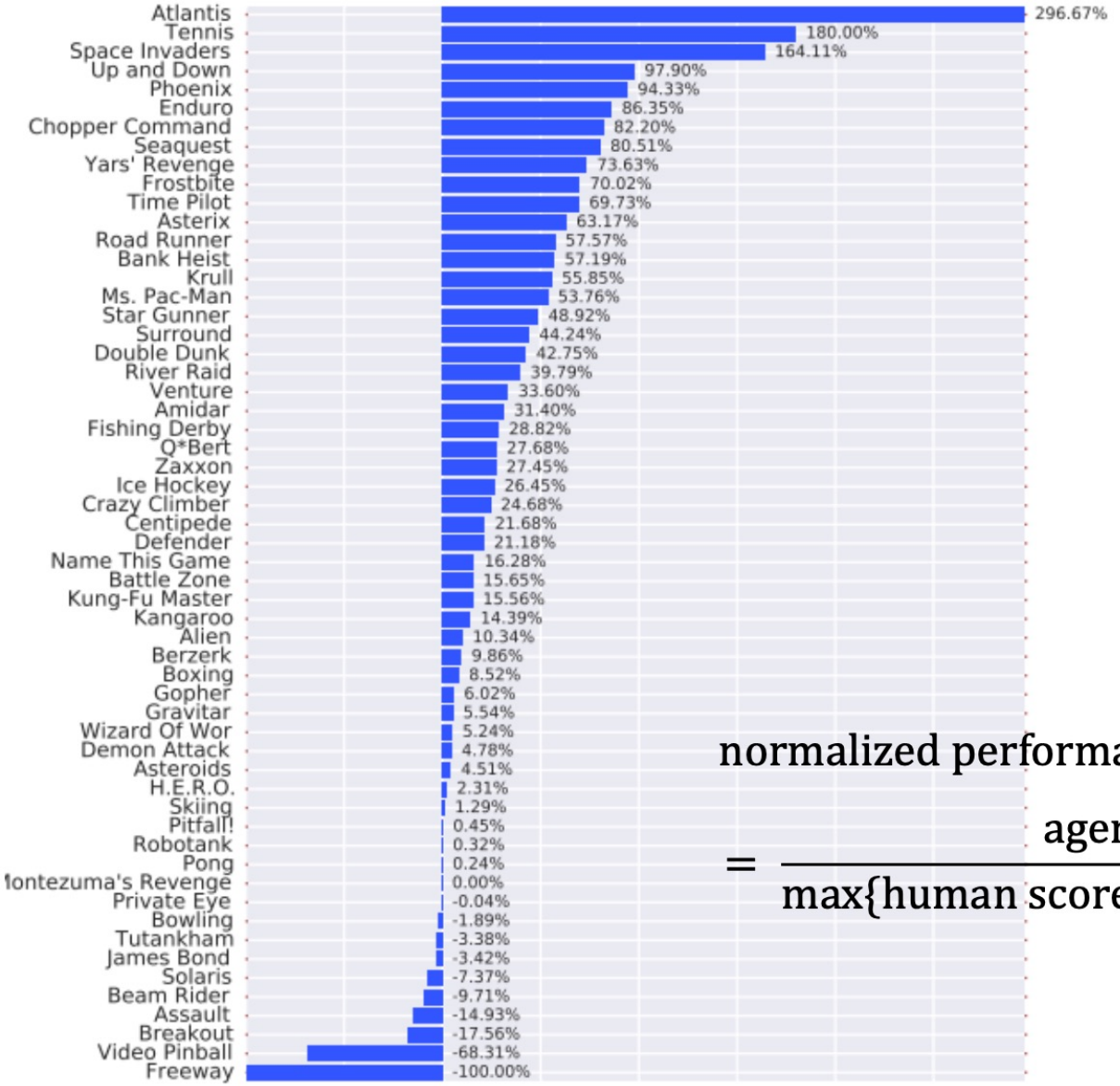
saliency maps



•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.

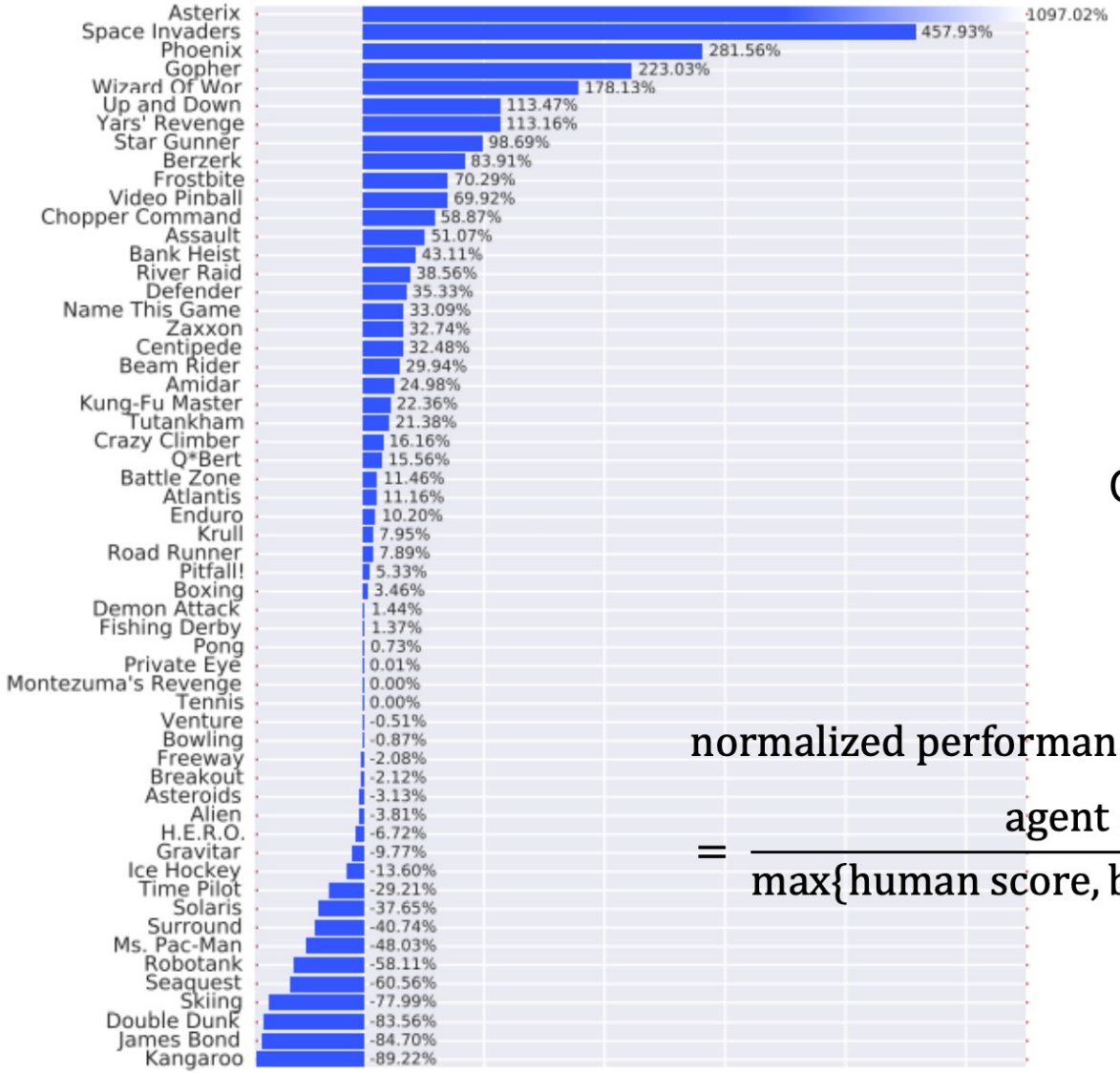
Experimental results in the Atari environment I



Compared with DQN

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

Experimental results in the Atari environment II



Compared with Double DQN

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

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 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

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

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

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

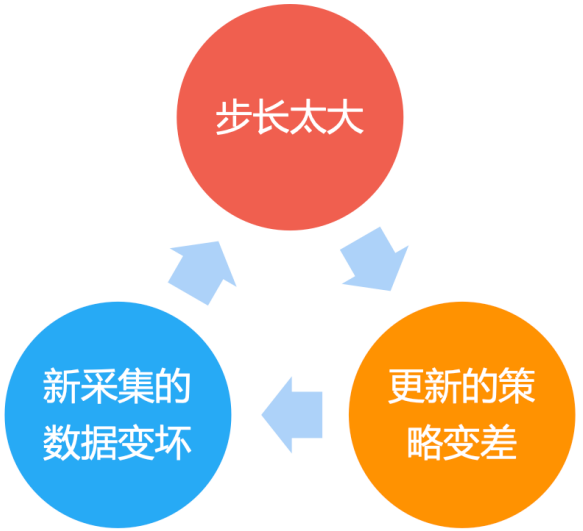
- Q-Learning:
  - Learns a Q-value function $Q_\theta(s, a)$ parameterized by θ
  - Objective: Minimize the TD error
- Policy gradient
  - Learns a policy $\pi_\theta(a \mid s)$ directly, parameterized by θ
  - Objective: Maximize the expected return directly

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

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

Limitations of policy gradient methods

- Learning rate (step size) selection is challenging in policy gradient algorithms
  - Since the data distribution changes as the policy updates, a previously good learning rate may become ineffective.
  - A poor choice of step size can significantly degrade performance:
    - Too large → policy diverges or collapses
    - Too small → slow convergence or stagnation



Trust Region Policy Optimization (TRPO)

- Two forms of the optimization objective
  - Form 1: Trajectory-based objective $J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[\sum_t \gamma^t r(s_t, a_t)]$
  - Form 2: State-value-based objective $J(\theta) = \mathbb{E}_{s_0 \sim p_\theta(s_0)}[V^{\pi_\theta}(s_0)]$

Optimization gap of the objective function

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[\sum_t \gamma^t r(s_t, a_t)]$$
$$J(\theta) = \mathbb{E}_{s_0 \sim p_\theta(s_0)}[V^{\pi_\theta}(s_0)]$$

- New policy θ' and old policy θ

$$J(\theta') - J(\theta) = J(\theta') - \mathbb{E}_{s_0 \sim p(s_0)}[V^{\pi_\theta}(s_0)]$$

Important:
Performance difference lemma

$$= \mathbb{E}_{\tau \sim p_{\theta'}(\tau)}[\sum_{t=0} \gamma^t A^{\pi_\theta}(s_t, a_t)]$$

Sampling inconvenience

$$A^{\pi_\theta}(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)$$

How to understand this lemma?

- Recall the policy improvement step in the PI algorithm

- Policy Improvement: For fixed values, get a better policy using policy extraction
  - One-step look-ahead:

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^{\pi_i}(s') \right]$$

  - We proved that
  - $Q^{\pi_i}\left(s, \pi_{i+1}(s)\right) \geq Q^{\pi_i}\left(s, \pi_i(s)\right), \forall s \implies V^{\pi_{i+1}}(s) \geq V^{\pi_i}(s), \forall s$

How to deal with sample inconvenience? Importance sampling

$$A^{\pi_\theta}(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)$$

$$J(\theta') - J(\theta)$$

$$= \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi_\theta}(s_t, a_t) \right]$$

$$= \sum_t \mathbb{E}_{s_t \sim p_{\theta'}(s_t)} [\mathbb{E}_{a_t \sim \pi_{\theta'}(a_t|s_t)} [\gamma^t A^{\pi_\theta}(s_t, a_t)]]$$

$$= \sum_t \mathbb{E}_{s_t \sim p_{\theta'}(s_t)} [\mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)} [\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \gamma^t A^{\pi_\theta}(s_t, a_t)]]$$

$p_{\theta'}$, approximation

Importance sampling

Ignoring the difference in state distributions

- When the change between the old policy π_θ and the new policy $\pi_{\theta'}$ is small, we can approximate $p_\theta(s_t) \approx p_{\theta'}(s_t)$
  - For deterministic policies:
    The probability that $\pi_{\theta'}(s_t) \neq \pi_\theta\ (s_t)$ is less than a small threshold ϵ.
  - For stochastic policies:
    The probability that $a' \sim \pi_{\theta'}(\cdot\,|s_t) \neq a \sim \pi_\theta(\cdot\,|s_t)$ is less than ϵ.

$$J(\theta') - J(\theta) \approx \sum_t \mathbb{E}_{s_t \sim p_\theta(s_t)}[\mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)}[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \gamma^t A^{\pi_\theta}(s_t, a_t)]]$$

TRPO Policy Constraint

- Use KL divergence to constrain policy update magnitude:

$$\theta' \leftarrow \arg\max_{\theta'} \sum_t \mathbb{E}_{s_t \sim p_\theta(s_t)}[\mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)}[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \gamma^t A^{\pi_\theta}(s_t, a_t)]]$$

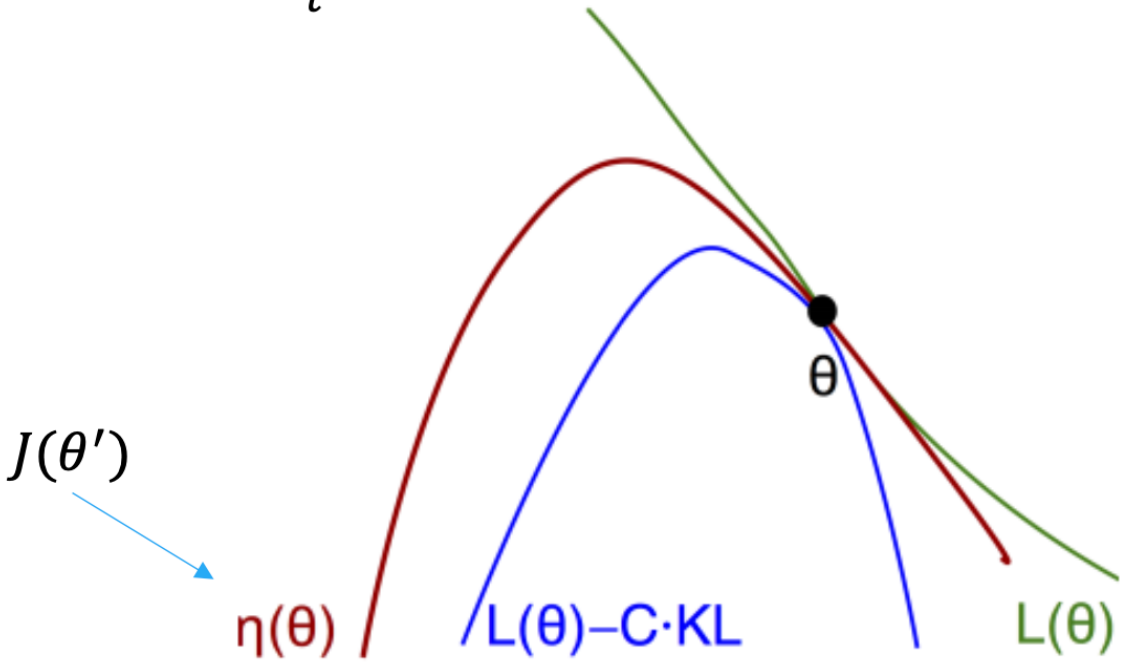
$$\text{such that } \mathbb{E}_{s_t \sim p_\theta(s_t)}[D_{KL}(\pi_{\theta'}(a_t|s_t) \parallel \pi_\theta(a_t|s_t))] \leq \epsilon$$

TRPO Monotonic Improvement Guarantee

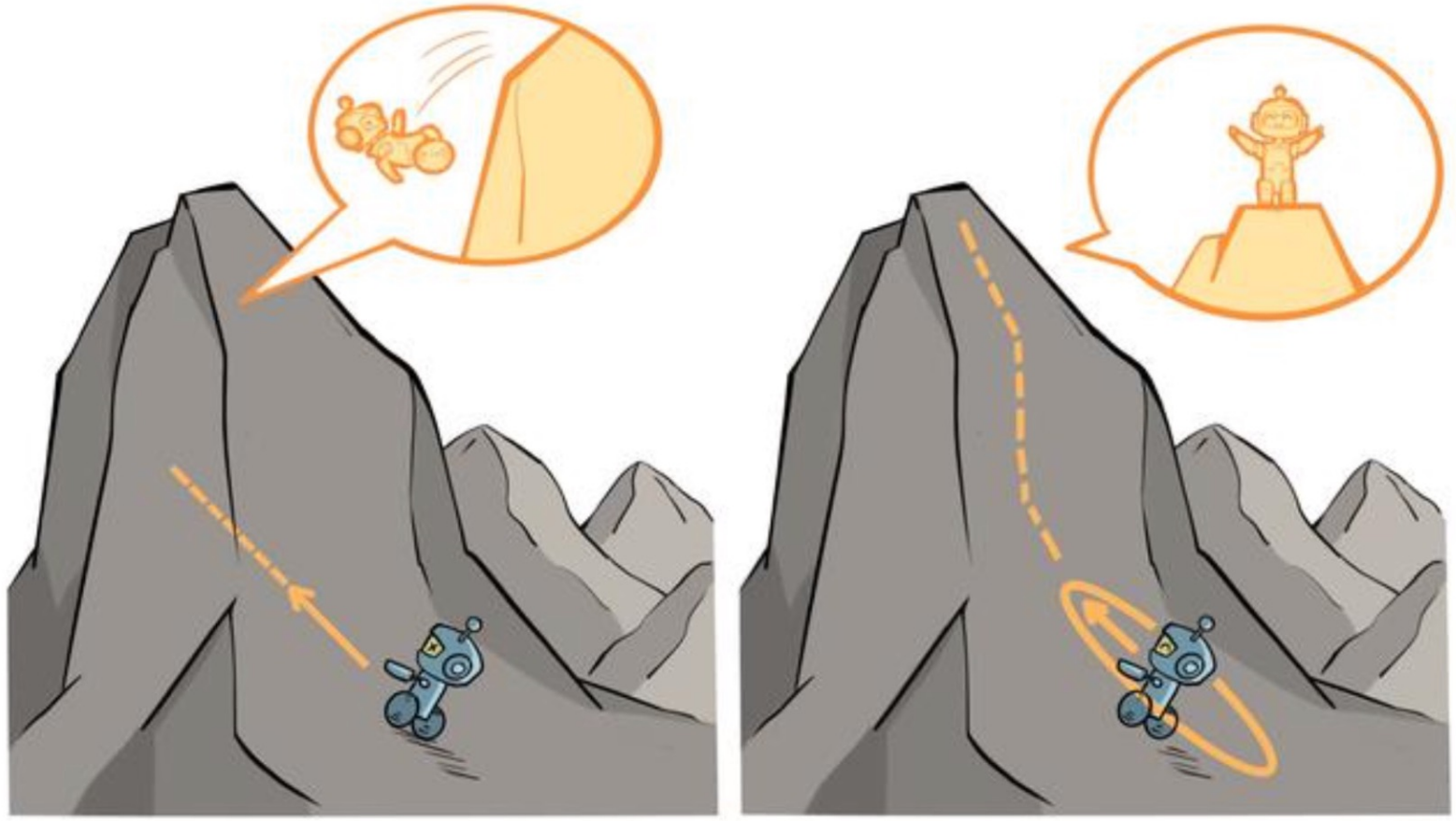
$$J(\theta') \geq L_\theta(\theta') - C \cdot D_{KL}^{max}(\theta, \theta'), where\ C = \frac{4\epsilon\gamma}{(1-\gamma)^2}, \epsilon = \max_{s,a} |A_\pi(s,a)|$$

$$L_\theta(\theta') = J(\theta) + \sum_t \mathbb{E}_{s_t \sim p_\theta(s_t)}[\mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)}[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \gamma^t A^{\pi_\theta}(s_t, a_t)]]$$

If policies θ and θ' are close in KL-divergence, then the approximation is good



Principle of TRPO



Gradient Ascent

Optimization in Trust Region

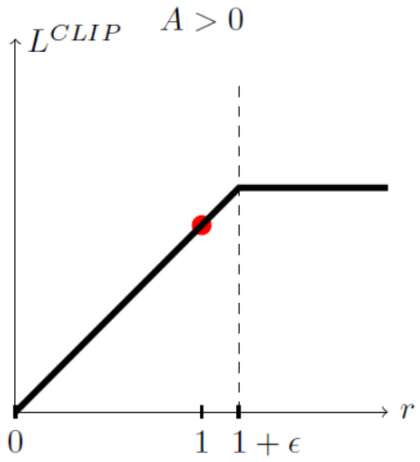
Proximal Policy Optimization (PPO)

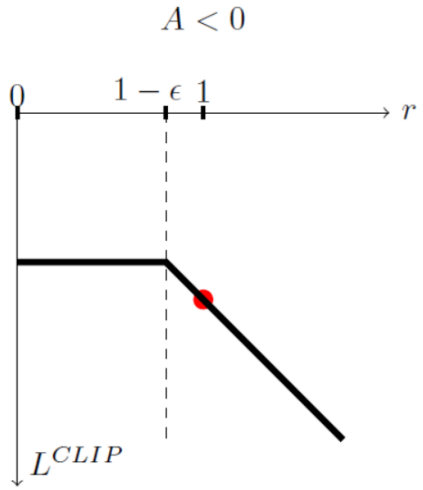
- Clipped Surrogate Objective

conservative policy iteration

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right]$$

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$





Construct the lower bound: $L^{CLIP}(\theta) \leq L^{CPI}(\theta)$

Equivalent at r=1: $L^{CLIP}(\theta) = L^{CPI}(\theta)$

PPO: Adaptive penalty version

- Another version: adaptive penalty version

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t - \beta \text{KL}[\pi_{\theta_{\text{old}}}(\cdot|s_t)|\pi_\theta(\cdot|s_t)] \right]$$

  - Adjust the penalty coefficient β dynamically:
    - Compute the KL value $d = \hat{\mathbb{E}}_t \left[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot|s_t)|\pi_\theta(\cdot|s_t)] \right]$
    - If d < target / 1.5 → β← β / 2
    - If d > target × 1.5 → β← β × 2

Note: Here, 1.5 and 2 are empirical parameters, and the algorithm performance is not very sensitive to them

PPO experimental comparison

No clipping or penalty: $L_t(\theta) = r_t(\theta)\hat{A}_t$

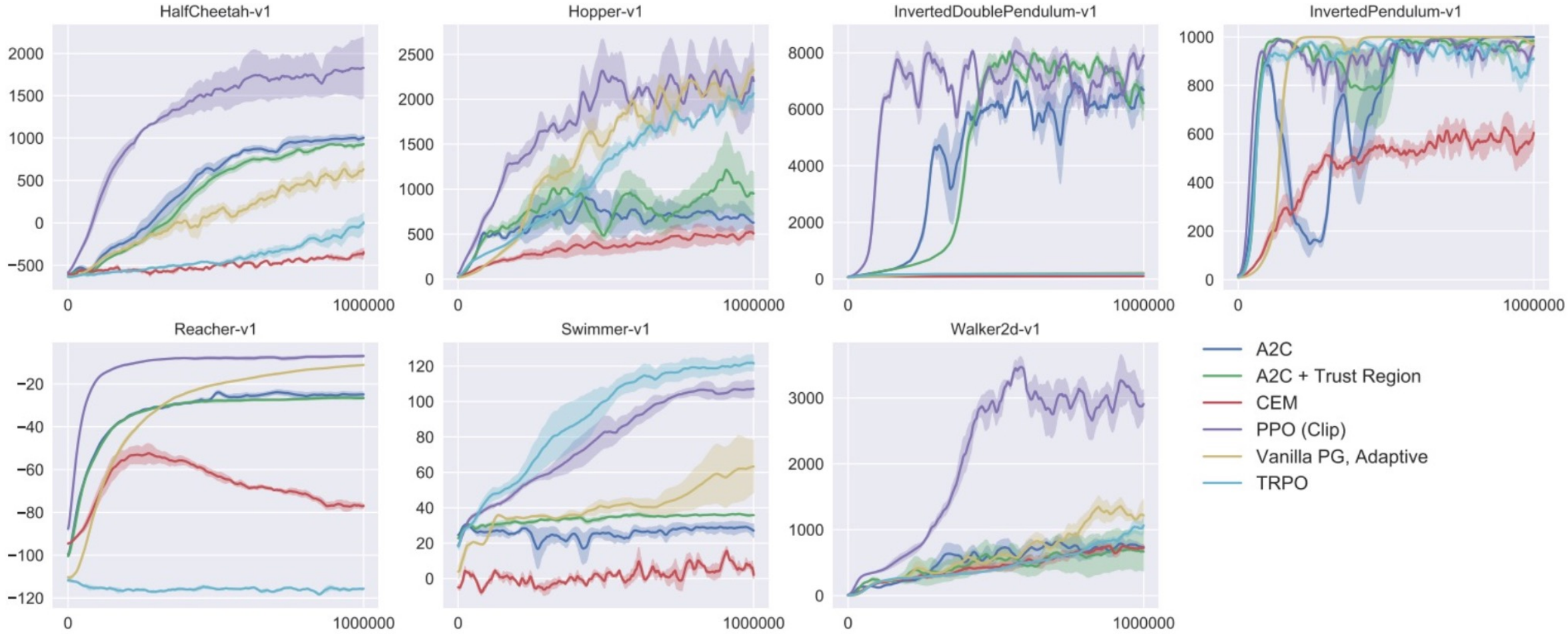
Clipping: $L_t(\theta) = \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta)), 1-\epsilon, 1+\epsilon)\hat{A}_t$

KL penalty (fixed or adaptive) $L_t(\theta) = r_t(\theta)\hat{A}_t - \beta\,\text{KL}[\pi_{\theta_{\text{old}}}, \pi_\theta]$

7 continuous control environments
3 random seeds
Each algorithm runs 100 episodes, repeated 21 times
Scores normalized to best model achieving 1.0

algorithm	avg. normalized score
No clipping or penalty	-0.39
Clipping, $\epsilon = 0.1$	0.76
Clipping, $\epsilon = 0.2$	**0.82**
Clipping, $\epsilon = 0.3$	0.70
Adaptive KL $d_{\text{targ}} = 0.003$	0.68
Adaptive KL $d_{\text{targ}} = 0.01$	0.74
Adaptive KL $d_{\text{targ}} = 0.03$	0.71
Fixed KL, $\beta = 0.3$	0.62
Fixed KL, $\beta = 1.$	0.71
Fixed KL, $\beta = 3.$	0.72
Fixed KL, $\beta = 10.$	0.69

PPO in MuJoCo



Summary

- 1. Value-based deep RL
  - DQN
  - Double DQN
  - Dueling DQN
- 2. Policy-based RL
  - Policy gradient
  - TRPO
  - PPO

- Next lecture: RL in LLMs