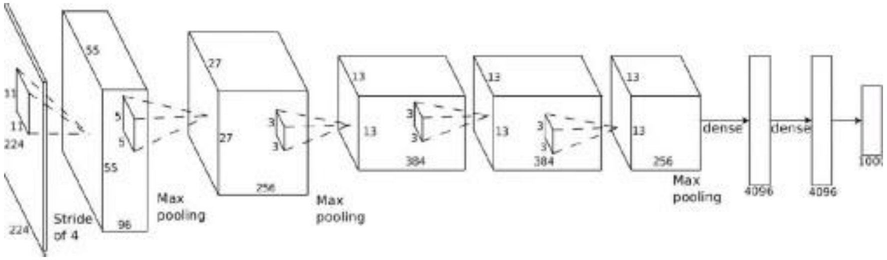
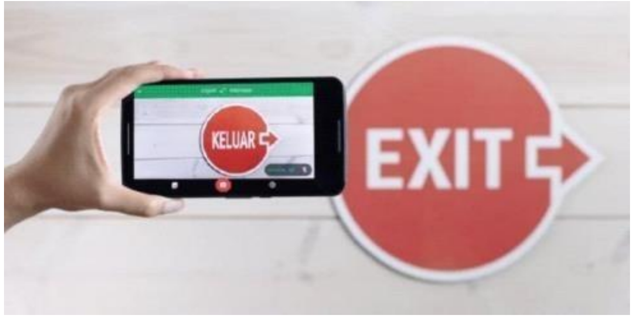
Lecture 5：Deep Reinforcement Learning

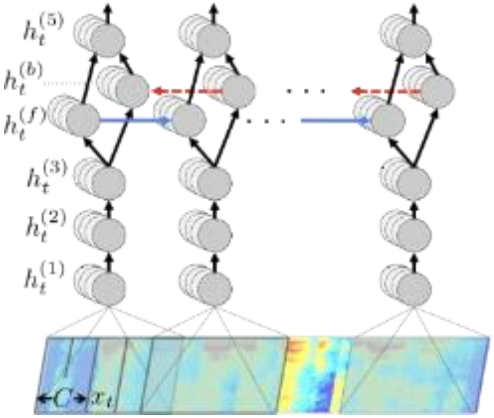
14th June. 2022

Deep Learning

- Deep Learning is part of a broader family of machine learning methods based on artificial neural networks with representation learning

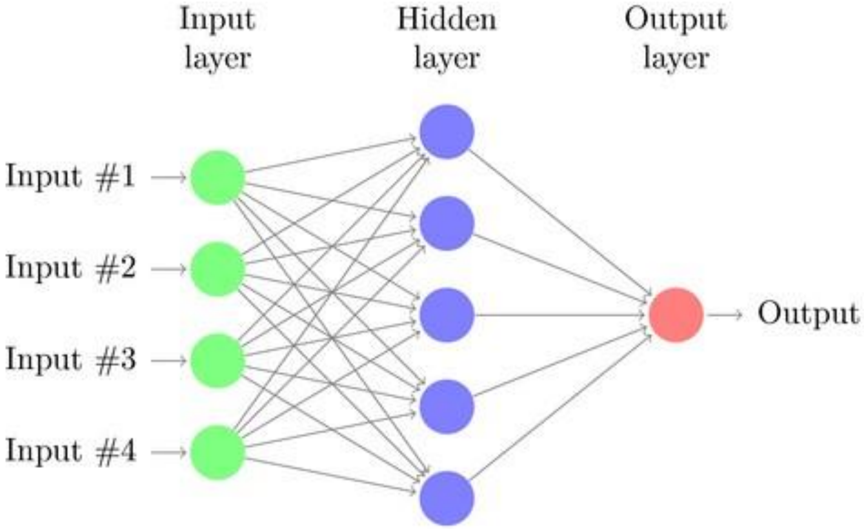






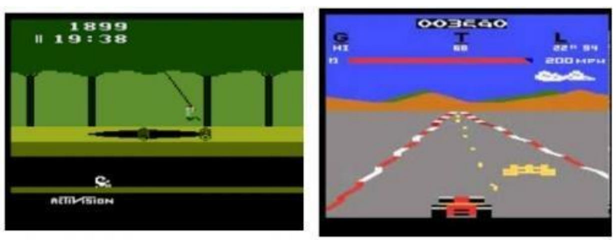
Deep Neural Networks(DNN)

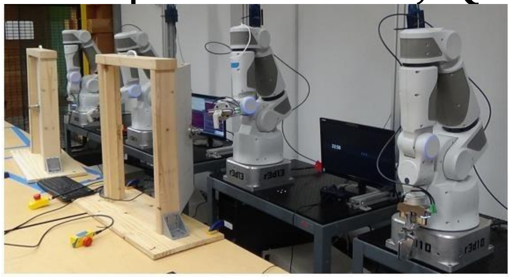
- Composition of multiple functions
- Can use the chain rule to backpropagate the gradient
- Generally combines both linear and non-linear transformations
- To fit the parameters, require a loss function(MSE, log likelihood, etc.)
- Major innovation: tools to automatically compute gradients for a DNN
- Deep Learning helps us handle unstructured environments

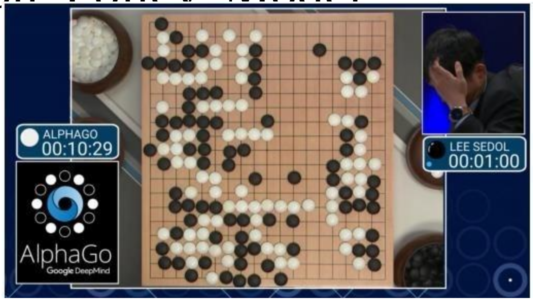


Deep Reinforcement Learning

- *What is deep RL, and why should we care?*
  - Deep learning helps us handle unstructured environments.
  - Deep models are what allow reinforcement learning algorithms to solve complex problems
    - Deep = can process complex sensory input
    - RL = can choose complex actions
  - Use deep neural networks to represent Value, Q function, Policy, Model







Atari games:

Q-learning:

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, et al. "Playing Atari with Deep Reinforcement Learning". (2013).

Policy gradients:

J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. "Trust Region Policy Optimization". (2015).
V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, et al. "Asynchronous methods for deep reinforcement learning". (2016).

Real-world robots:

Guided policy search:

S. Levine*, C. Finn*, T. Darrell, P. Abbeel. "End-to-end training of deep visuomotor policies". (2015).

Q-learning:

D. Kalashnikov et al. "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". (2018).

Beating Go champions:

Supervised learning + policy gradients + value functions + Monte Carlo tree search:

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, et al. "Mastering the game of Go with deep neural networks and tree search". Nature (2016).

Deep RL with Q-Functions

- Naïve deep Q-learning
  - Represent state-action value function by Q-network

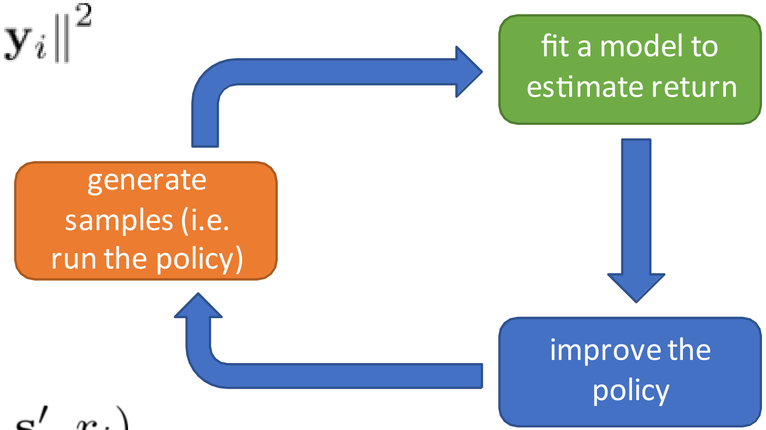
full fitted Q-iteration algorithm:

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy

$K\times$

2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

$Q_\phi(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}', \mathbf{a}')$



$\mathbf{a} = \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$

online Q iteration algorithm:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

Deep RL with Q-Functions

- Two of the issues:
  - Correlations between samples - sequential states are strongly correlated
  - Non-stationary targets - target value is always changing

online Q iteration algorithm:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}_i', r_i)$
2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}_i', \mathbf{a}_i')$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

these are correlated!

isn't this just gradient descent? that converges, right?

Q-learning is *not* gradient descent!

$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}_i', \mathbf{a}_i')])$$

no gradient through target value

Deep RL with Q-Functions

Solution: replay buffers

full Q-learning with replay buffer:

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$

$K\times$

2. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
3. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

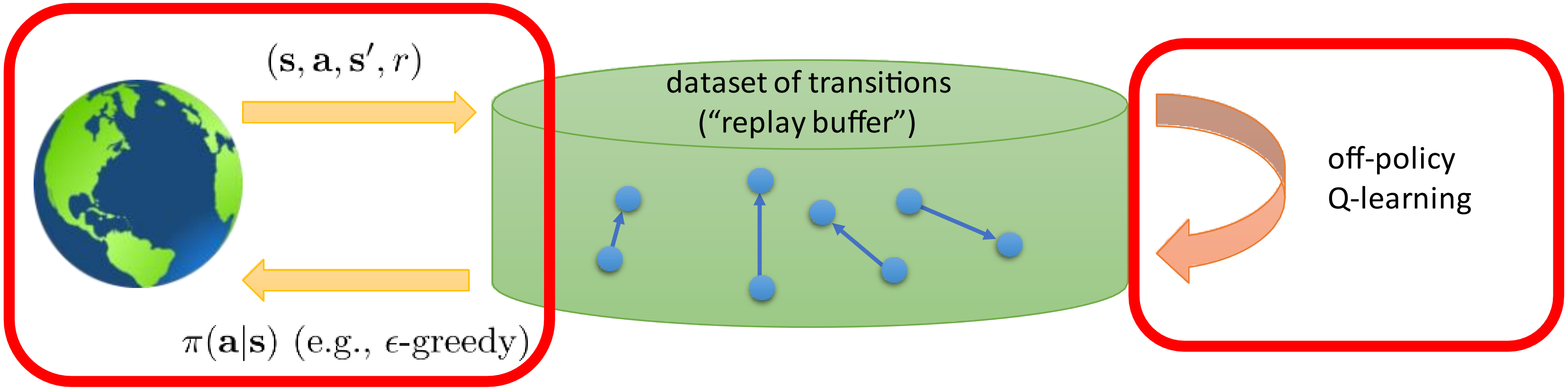
+ samples are no longer correlated

+ multiple samples in the batch (low-variance gradient)

K = 1 is common, though larger K more efficient

but where does the data come from?

need to periodically feed the replay buffer…



Deep RL with Q-Functions

Solution: Target Networks

online Q iteration algorithm:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

~~these are correlated!~~

use replay buffer

Q-learning is *not* gradient descent!

$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$$

no gradient through target value

This is still a problem!

Q-learning with Target Network:

Q-learning with replay buffer and target network:

1. save target network parameters: $\phi' \leftarrow \phi$

$N\times$

2. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$

$K\times$

3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$

targets don't change in inner loop!

supervised regression

Deep RL with Q-Functions

Deep Q-Network(DQN)

Q-learning with replay buffer and target network:

1. save target network parameters: $\phi' \leftarrow \phi$
   $N\times$
   2. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$
      $K\times$
      3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
      4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$

"classic" deep Q-learning algorithm:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. update ϕ': copy ϕ every N steps

$K = 1$

Deep RL with Q-Functions

Deep Q-Network(DQN) Summary

- Use experience replay and target network
- The target network is time-delayed
- Sample random mini-batch from replay buffer
- Use stochastic gradient descent

Q-learning with replay buffer and target network:

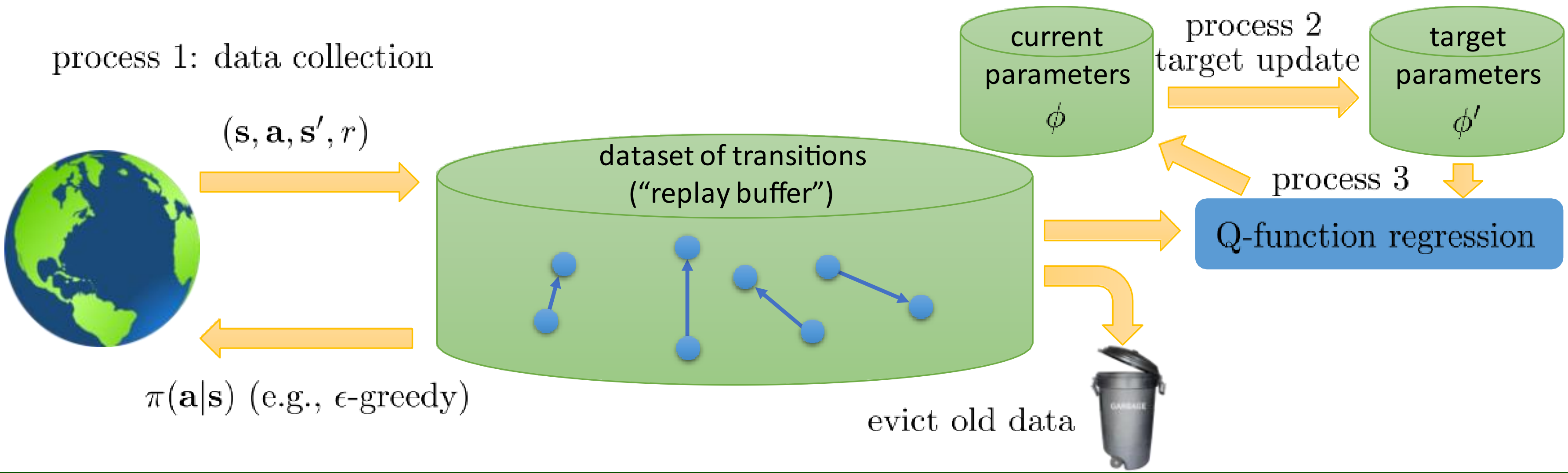
1. save target network parameters: $\phi' \leftarrow \phi$

$N\times$

2. collect M datapoints $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add them to $\mathcal{B}$

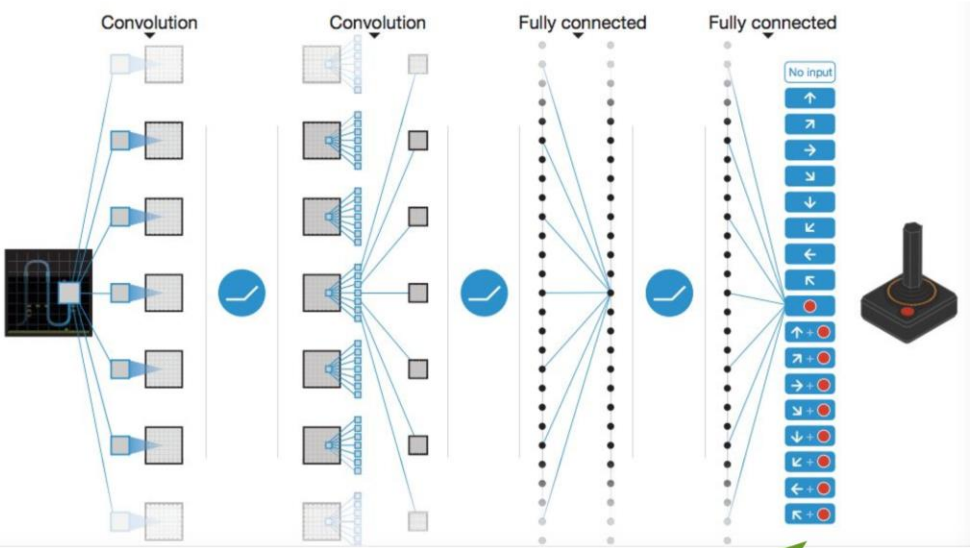
$K\times$

3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$

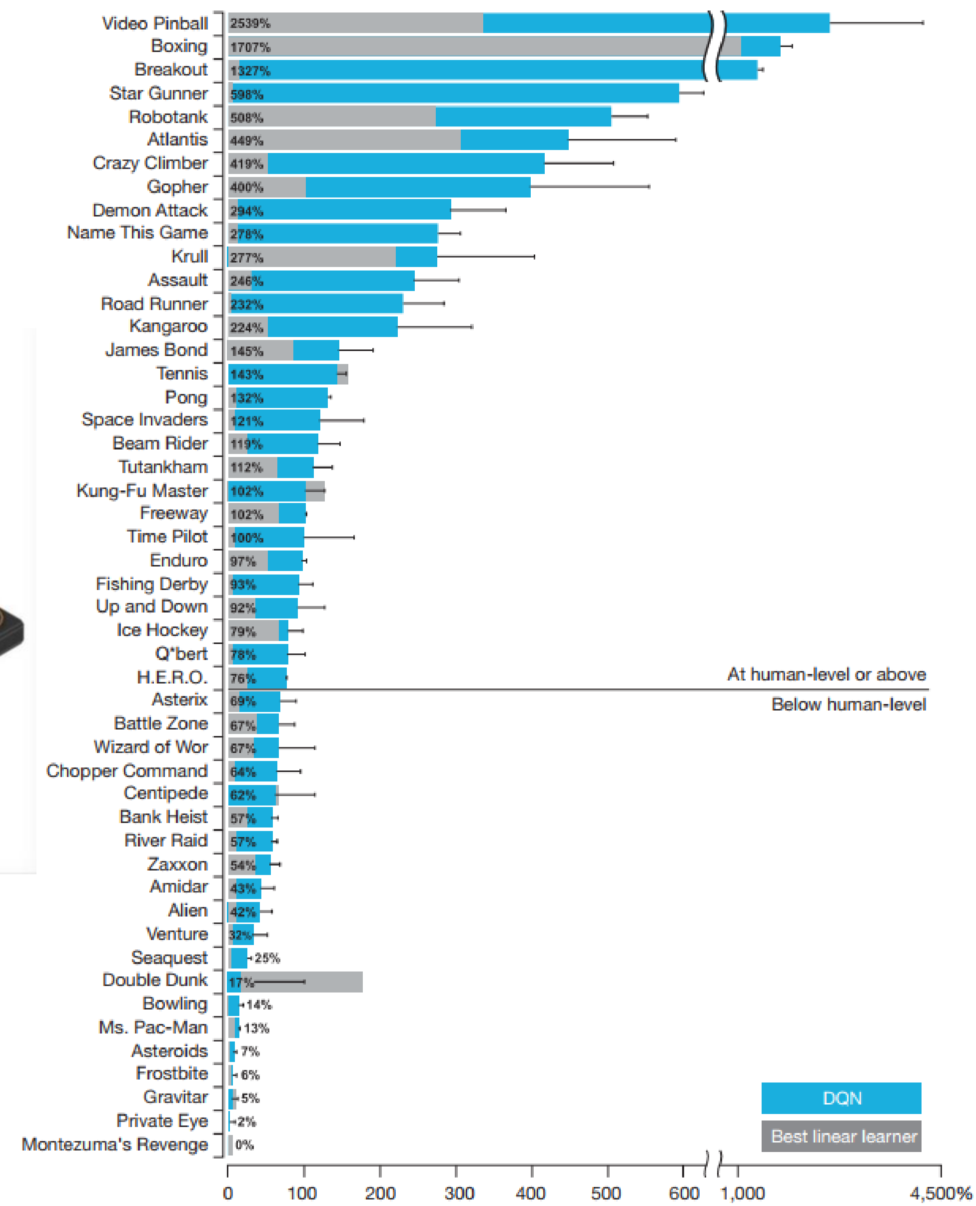


Deep RL with Q-Functions

☐ Network and Performance



1 network, outputs Q value for each action



Deep RL with Q-Functions

- Variant
  - Double DQN： solving overestimation in DQN

target value $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$

this last term is the problem

imagine we have two random variables: X_1 and X_2

$$E[\max(X_1, X_2)] \geq \max(E[X_1], E[X_2])$$

$Q_{\phi'}(\mathbf{s}', \mathbf{a}')$ is not perfect – it looks "noisy"

hence $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}')$ *overestimates* the next value!

idea: don't use the same network to choose the action and evaluate value!

"double" Q-learning: use two networks:

$$Q_{\phi_A}(\mathbf{s}, \mathbf{a}) \leftarrow r + \gamma Q_{\phi_B}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi_A}(\mathbf{s}', \mathbf{a}'))$$

$$Q_{\phi_B}(\mathbf{s}, \mathbf{a}) \leftarrow r + \gamma Q_{\phi_A}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi_B}(\mathbf{s}', \mathbf{a}'))$$

if the two Q's are noisy in *different* ways, there is no problem

Deep RL with Q-Functions

- Variant
  - Double DQN： solving overestimation in DQN

where to get two Q-functions?

just use the current and target networks!

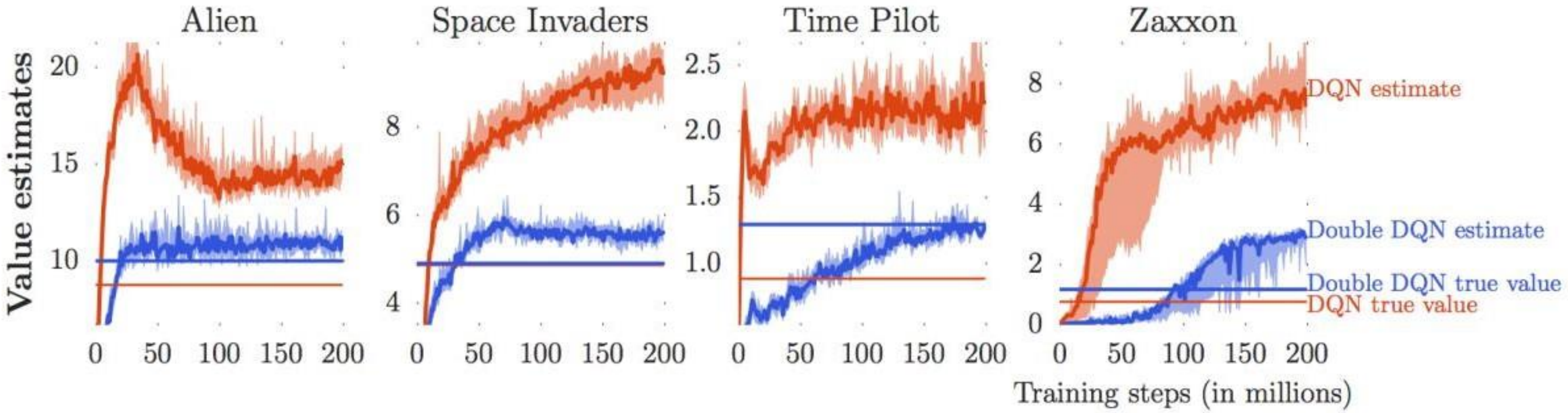
standard Q-learning: $y = r + \gamma Q_{\phi'}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}'))$

double Q-learning: $y = r + \gamma Q_{\phi'}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi}(\mathbf{s}', \mathbf{a}'))$

just use current network (not target network) to evaluate action

still use target network to evaluate value!

  - Value estimation in Atari



Deep RL with Q-Functions

Performance of Double DQN in Atari

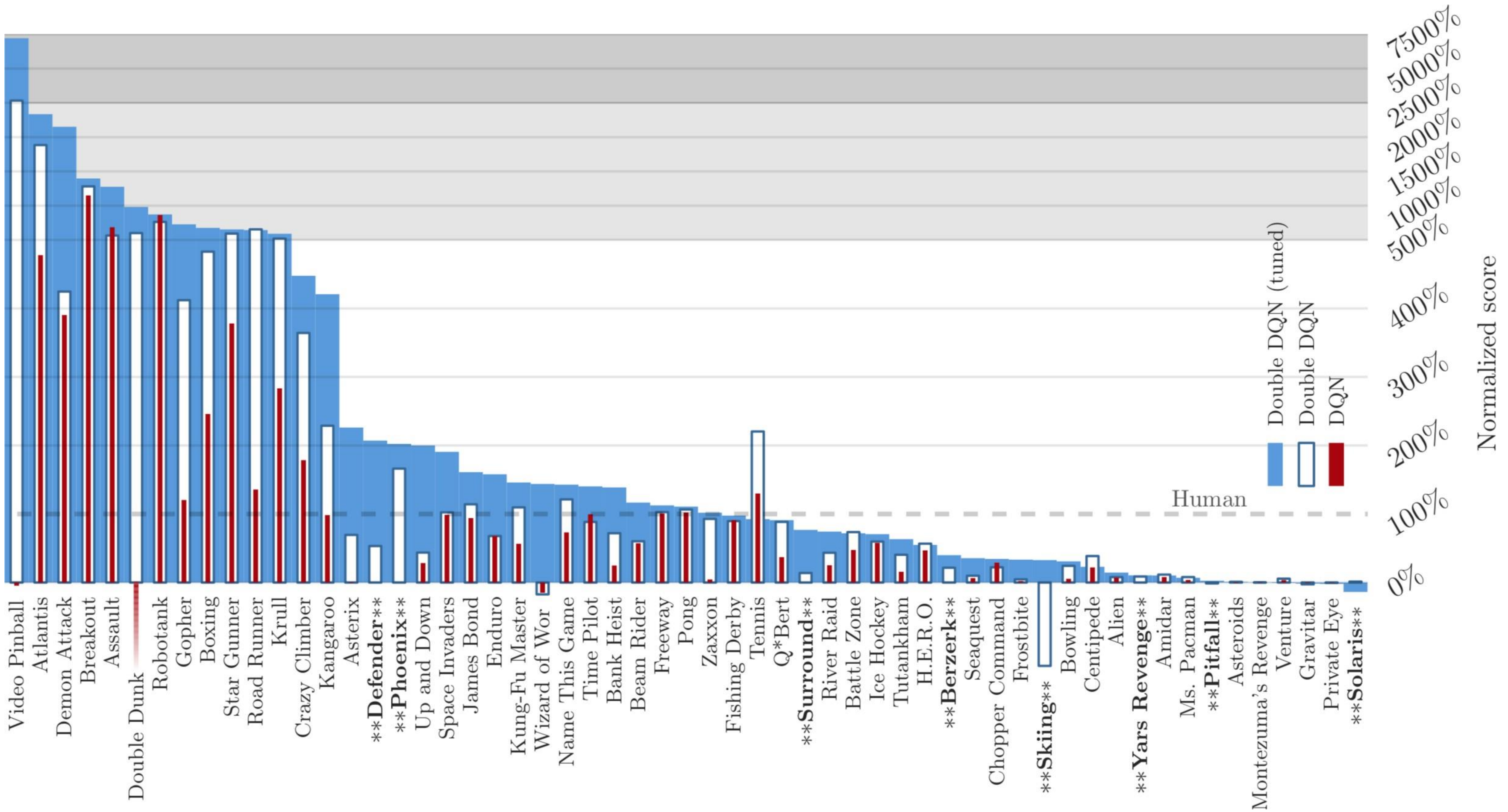


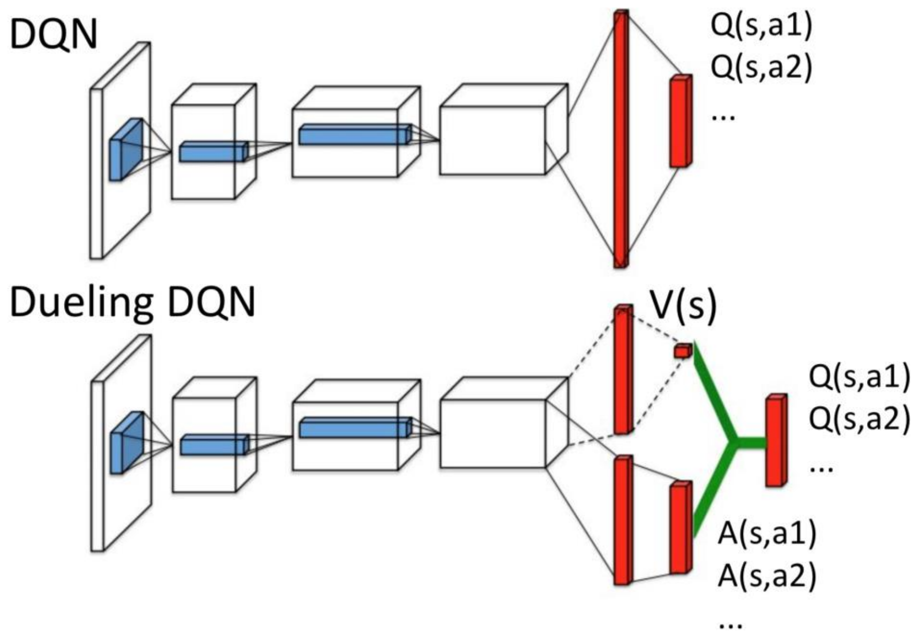
Figure: van Hasselt, Guez, Silver, 2015

Deep RL with Q-Functions

- Variant
  - Dueling DQN
    - Sometimes it is unnecessary to know the exact value of each action
    - Split the Q-values in two different parts, the value function V(s) and the advantage function A(s, a), Q(s, a) = V(s) + A(s, a)
    - Value function V(s) : how much reward we will collect from the state s
    - Advantage function A(s, a) : how much better one action is compared to the other actions.
  - Prioritized experience replay
    - Weigh the samples so that "important" ones are drawn more frequently for training
  - Rainbow
    - Combining improvements : Double DQN、Dueling DQN、Prioritized Replay Buffer、Multi-Step Learning、Distributional DQN（Categorical DQN）、NoisyNet

Deep RL with Q-Functions

Network and performance of Dueling DQN



Wang et.al., ICML, 2016

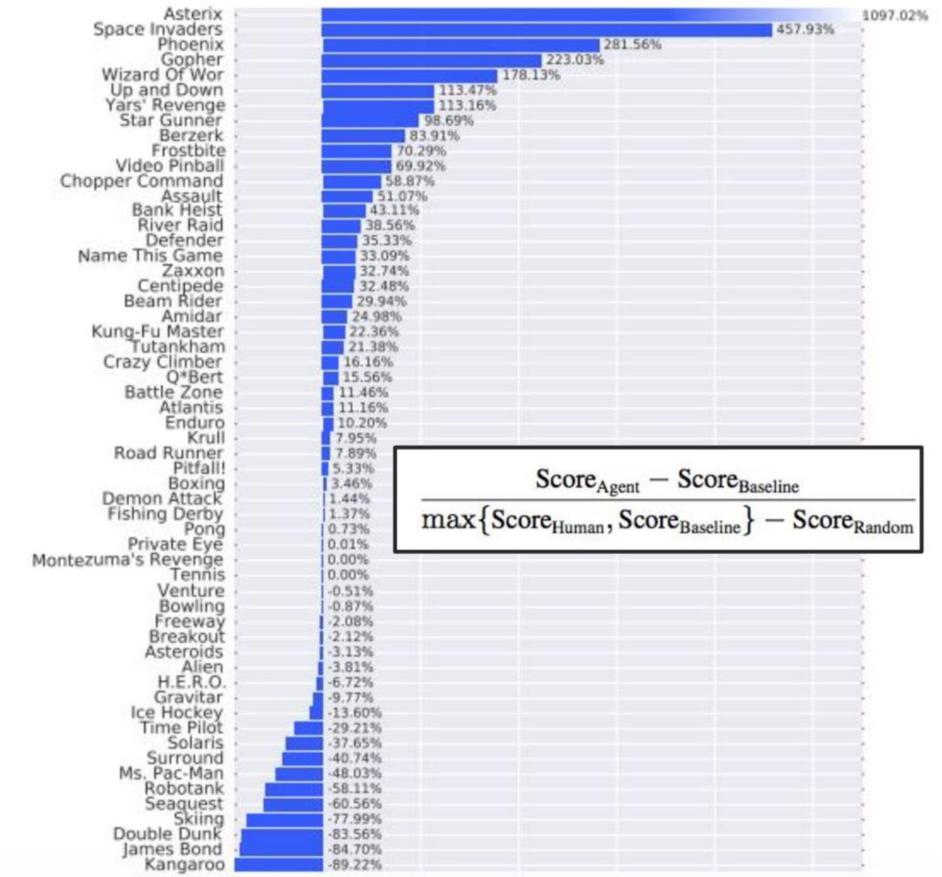


Figure: Wang et al, ICML 2016

Deep RL with Q-Functions

- Performance of Prioritized Experience Replay in Atari

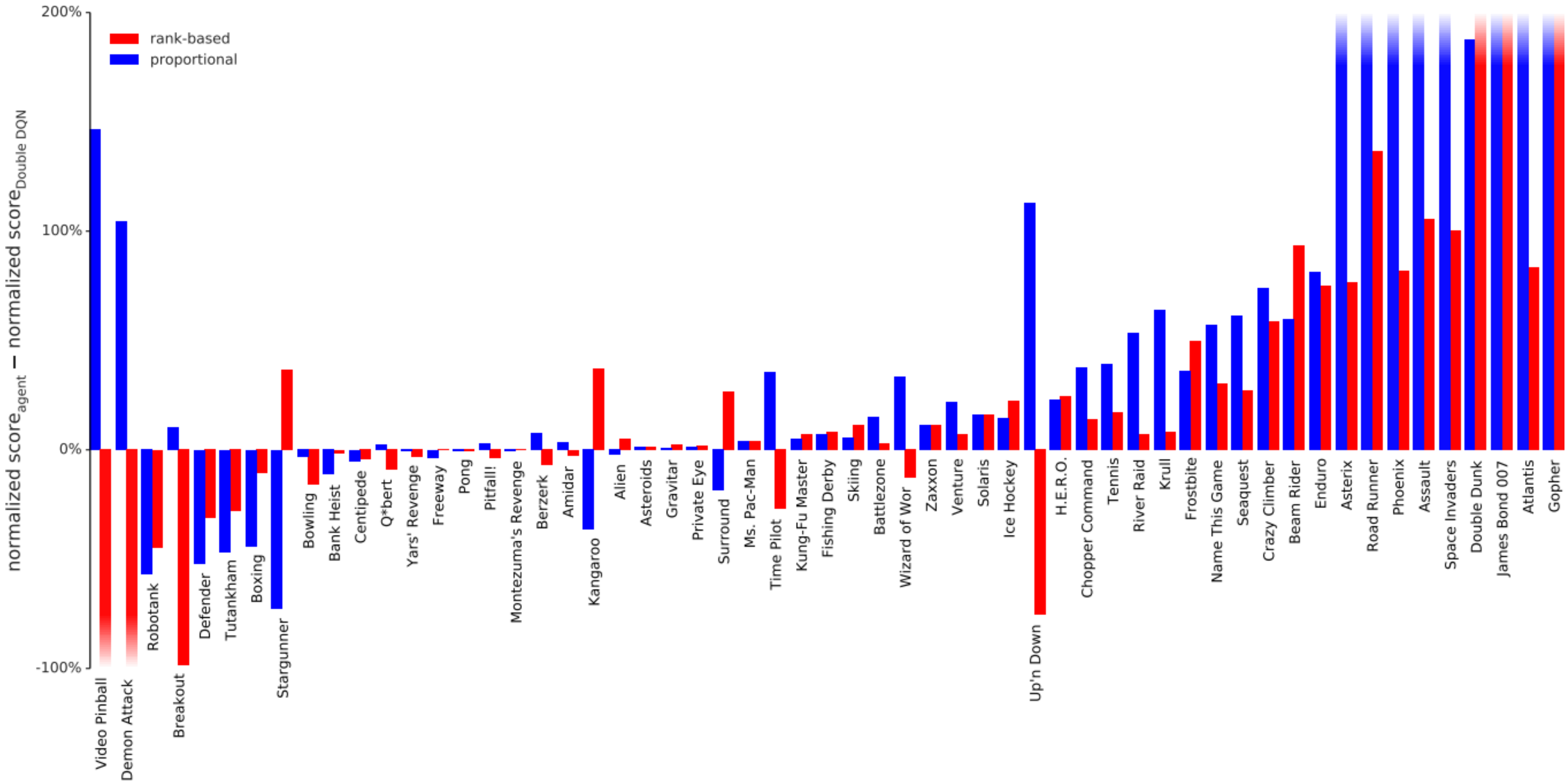


Figure: Schaul, Quan, Antonoglou, Silver ICLR 2016

Deep RL with Q-Functions

☐ Performance of Rainbow

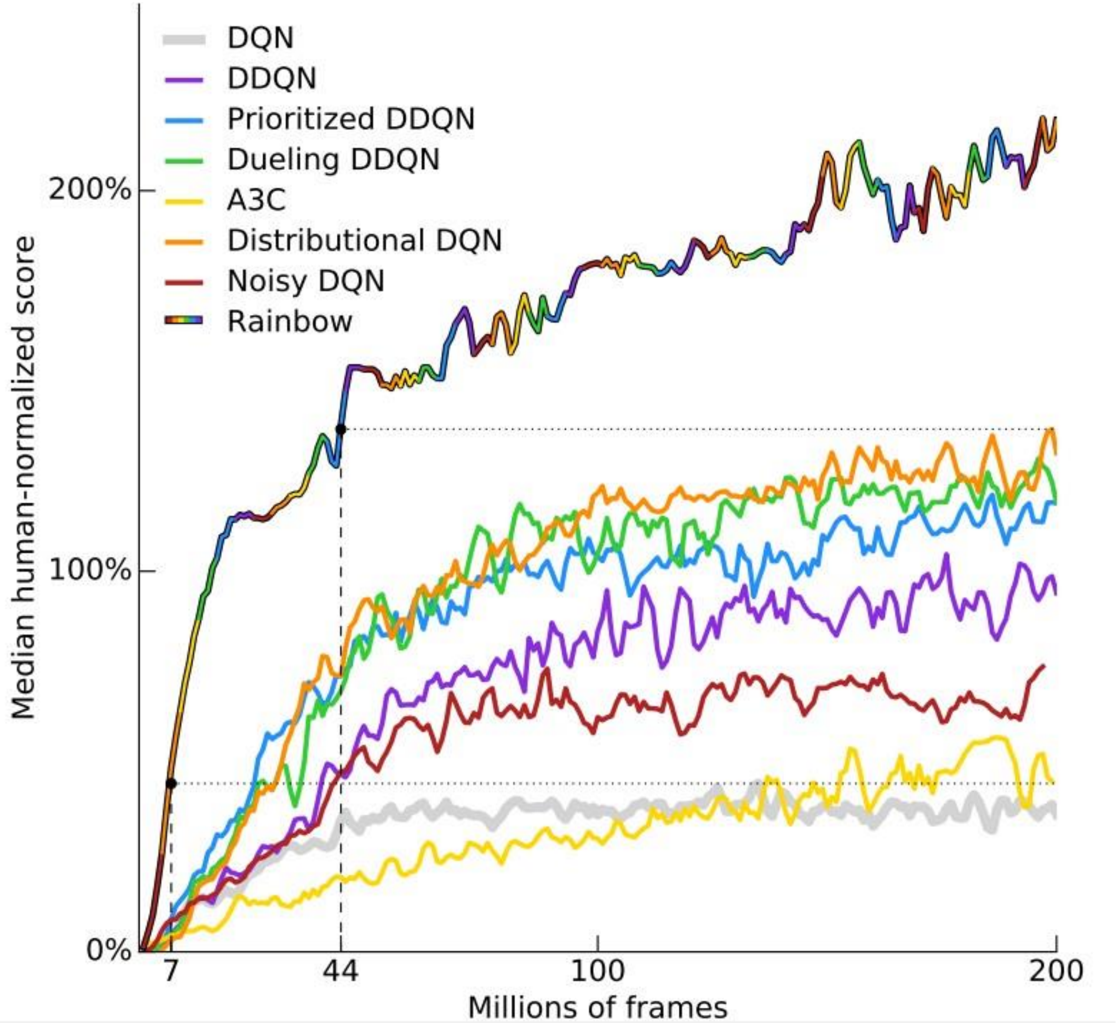


Figure: Hessel, Matteo, et al. "Rainbow: Combining Improvements in Deep Reinforcement Learning."

Deep RL with Q-Functions

- Q-learning with continuous actions
  - Problem

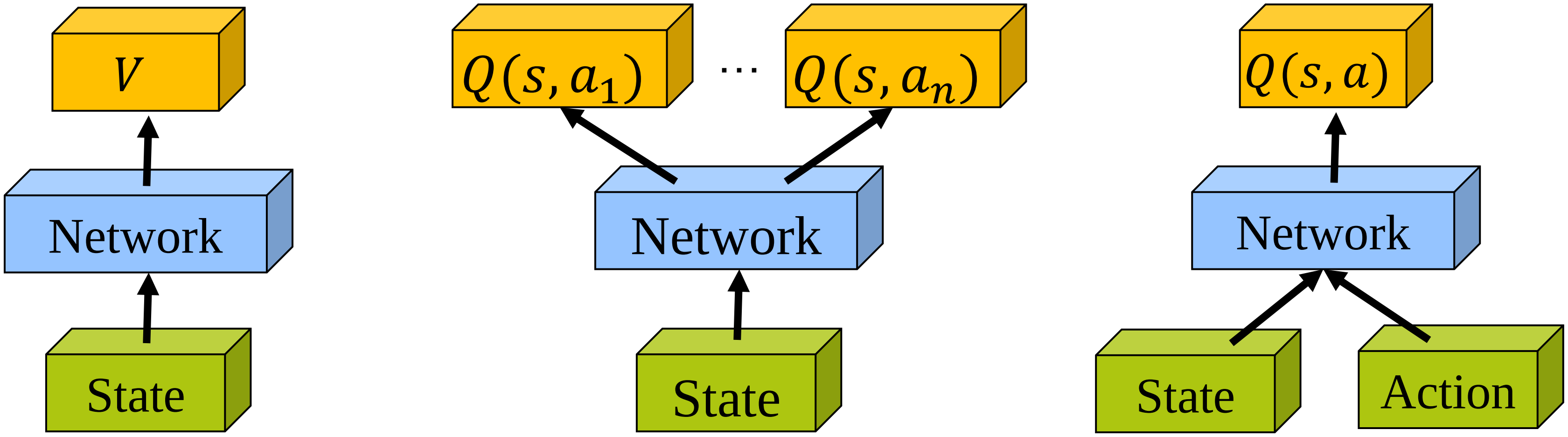
$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 \text{ otherwise} \end{cases}$$

$$\text{target value } y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$$

  - Solution
    - $\max_a Q(s,a) \approx \max\{Q(s,a_1),\ldots,Q(s,a_N)\}$, $(a_1,\ldots,a_N)$sampled from some distribution (e.g., uniform, Gaussian), but not very accurate.
    - Learn an approximate maximizer, Policy Gradient algorithm or DDPG ("deterministic" actor-critic, Lillicrap et al., ICLR 2016)

Deep RL with Q-Functions

- Q-learning with continuous actions
  - DDPG
    - Train actor network： $\mu_\theta(s) \approx argmax_a Q_\phi(s,a)$
    - Train critic network
    - Deterministic policy
  - A2C(Advantage Actor-Critic)
    - Train actor network with policy gradient theorem
    - Train critic network
    - Stochastic policy



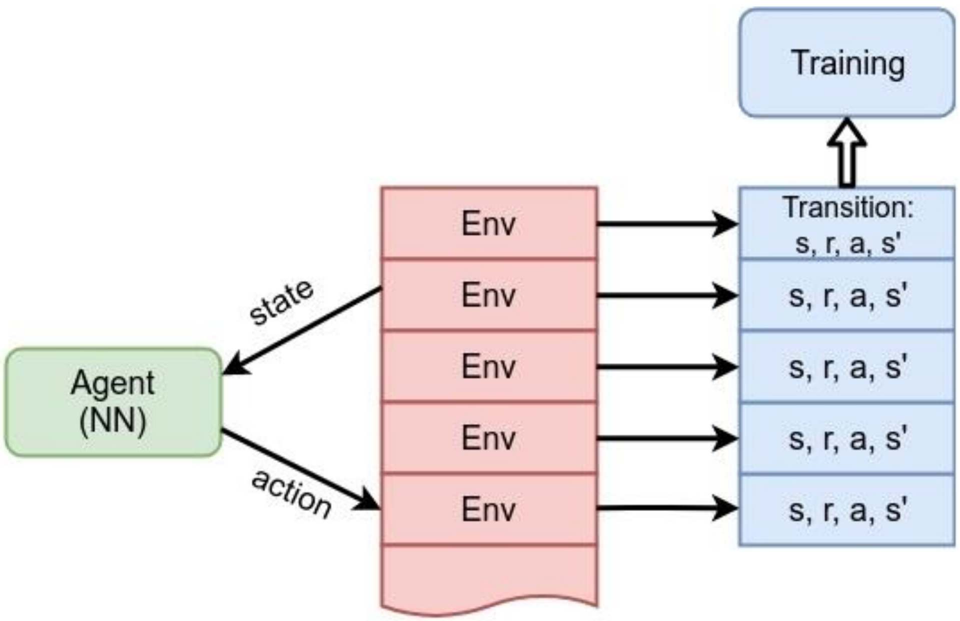
Deep RL with policy gradient

A3C

- Let π_θ denote a policy with parameters θ, and $J(\pi_\theta)$ denote the expected finite-horizon undiscounted return of the policy. The gradient of $J(\pi_\theta)$ is

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t)A^{\pi_\theta}(s_t,a_t)\right]$$

- Asynchronous A2C
  - Agents interact with their respective environments asynchronously, learning with each interaction. Each agent is controlled by a global network.

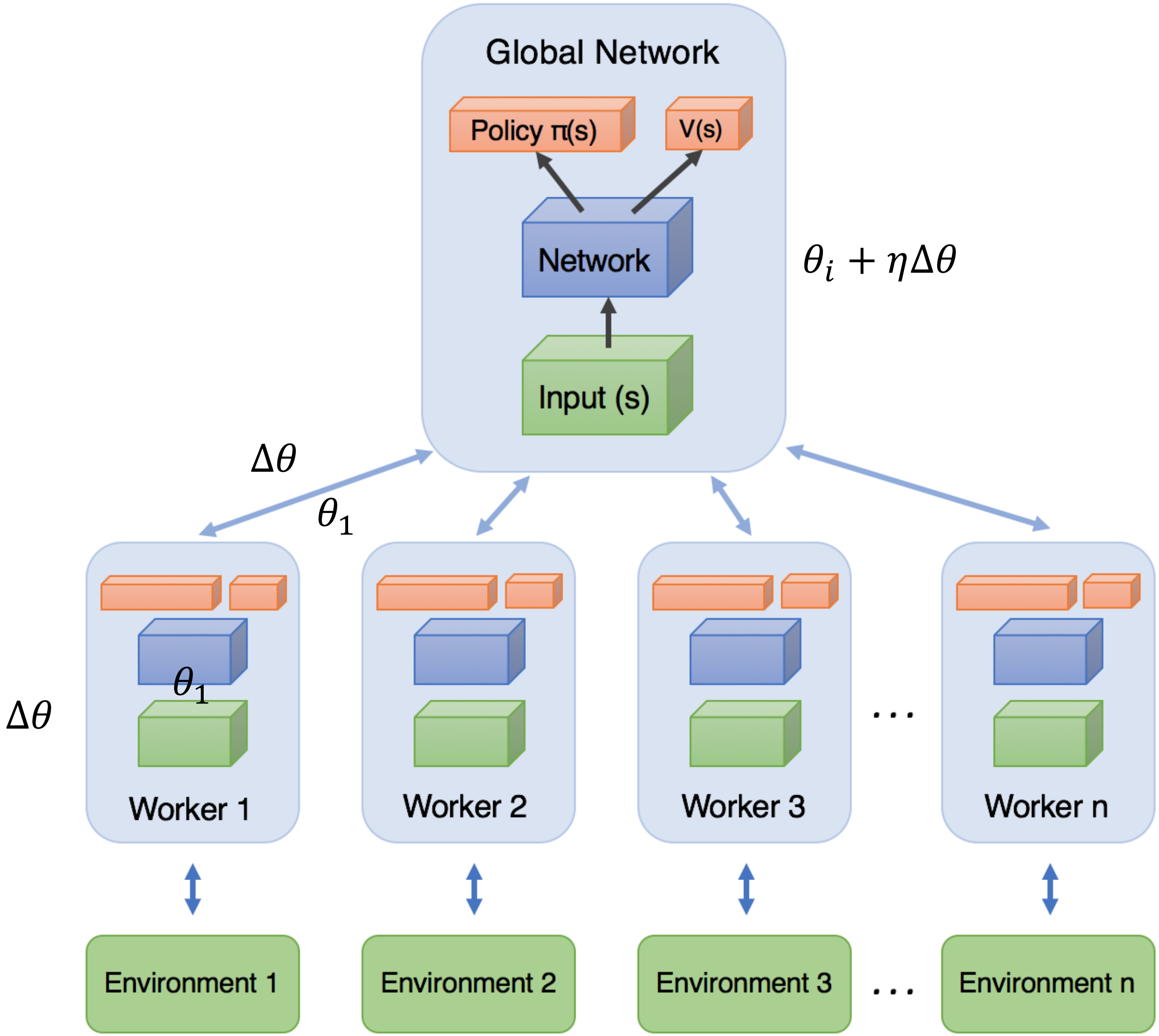


Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// *Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$*
// *Assume thread-specific parameter vectors θ' and θ'_v*
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t\text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \dots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial(R - V(s_i;\theta'_v))^2/\partial\theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

Deep RL with policy gradient

- A3C



Deep RL with policy gradient

DDPG(Deep Deterministic Policy Gradient)

- Idea: train actor network $\mu_\theta(s) \approx argmax_a Q_\phi(s,a)$

- Use four neural networks: a Q network, a deterministic policy network , a target q network, a target policy network

- The Q network and policy network is similar to actor-critic algorithm. But the Actor directly maps states to actions instead of outputting the probability distribution across a action space.

- Actor network:

$$\theta \leftarrow argmax_\theta Q_\phi(s, \mu_\theta(s)), \frac{dQ_\phi}{d\theta} = \frac{da}{d\theta}\frac{dQ_\phi}{da}$$

- Critic network: $y_j = r_j + \gamma Q_{\phi'}\left(s_j', \mu_{\theta'}(s_j')\right)$

$$\approx r_j + \gamma Q_{\phi'}\left(s_j', argmax_{a'} Q_{\phi'}(s_j', a_j')\right)$$

Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." *arXiv preprint arXiv:1509.02971* (2015).

Deep RL with policy gradient

- DDPG
  - **Pseudo Code**

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \mu_{\theta'}(\mathbf{s}'_j))$ using *target* nets $Q_{\phi'}$ and $\mu_{\theta'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. $\theta \leftarrow \theta + \beta \sum_j \frac{d\mu}{d\theta}(\mathbf{s}_j) \frac{dQ_\phi}{d\mathbf{a}}(\mathbf{s}_j, \mu(\mathbf{s}_j))$
6. update ϕ' and θ' (e.g., Polyak averaging)

  - Soft Updates(different with DQN)
    - Slowly track those of the learned networks via "soft updates"

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$
$$\phi' \leftarrow \tau\phi + (1-\tau)\phi'$$

Deep RL with policy gradient

◻ Network of DDPG