Robot Learning

Deep Q-Networks

Prof. Songhwai Oh
ECE, SNU

DEEP Q-LEARNING

Review: Action-Utility Function

- $Q(s,a)$: the value of doing action a in state s.

$$U(s) = \max_a Q(s,a)$$

- **Q-learning** is a model-free method: TD agent that learns a **Q-function** does not need a model of the form $P(s' \mid s,a)$ for learning or for action selection.
- At equilibrium when Q-values are correct:

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q(s',a')$$

cf. Bellman equation: $U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s,a) U(s')$

- ADP Q-learning: (1) estimate the transition model; (2) update Q-values.
- TD Q-learning (does not require the transition model)

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Review: Q-Learning

- Update rule for Q-learning:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

function Q-LEARNING-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r'
  persistent: Q, a table of action values indexed by state and action, initially zero
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    s, a, r, the previous state, action, and reward, initially null

  if TERMINAL?(s) **then** $Q[s, None] \leftarrow r'$
  if s is not null **then**
    increment $N_{sa}[s, a]$
    $Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$
  $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a']), r'$
  return a

Deep Q-Learning: Q function is approximated by a deep neural network.

$$\theta_i \leftarrow \theta_i + \alpha \left[R(s) + \gamma \max_{a'} \hat{Q}_\theta(s',a') - \hat{Q}_\theta(s,a)\right] \frac{\partial \hat{Q}_\theta(s,a)}{\partial \theta_i}$$

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "**Playing Atari with deep reinforcement learning**." NIPS Deep Learning Workshop 2013

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves et al. "**Human-level control through deep reinforcement learning**." *Nature* 518, no. 7540 (2015): 529.

DEEP Q-NETWORK (DQN)

Q-Learning with DNNs

Bellman equation (optimality condition):

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') \middle| s,a\right]$$

-> value iteration

Q-network, $Q(s,a;\theta)$, approximates $Q^*(s,a)$ and is learned by minimizing:

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$$

$$y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | s,a\right]$$

$\rho(s,a)$: probability distribution over s and a.

Target value is **not** fixed (cf. supervised learning)

Stochastic gradient descent to learn θ:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

Issues: correlated data and non-stationary distribution -> **make Q-learning unstable**
Solution: experience replay

Notation

- $\mathcal{E}$: Atari emulator
- $a_t \in \mathcal{A} = \{1, \dots, k\}$: actions
- $x_t \in \mathbb{R}^d$: image (screen shot from $\mathcal{E}$)
- r_t: reward (changes in game score)

- $s_t = x_1, a_1, x_2, \dots, a_{t-1}, x_t$: state (cf. POMDP)
- $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$: future discounted return (T: terminal time)
- $Q^*(s, a) = \max_{\pi} \mathbb{E}(R_t | s_t = s, a_t = a, \pi)$

Deep Q-Learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

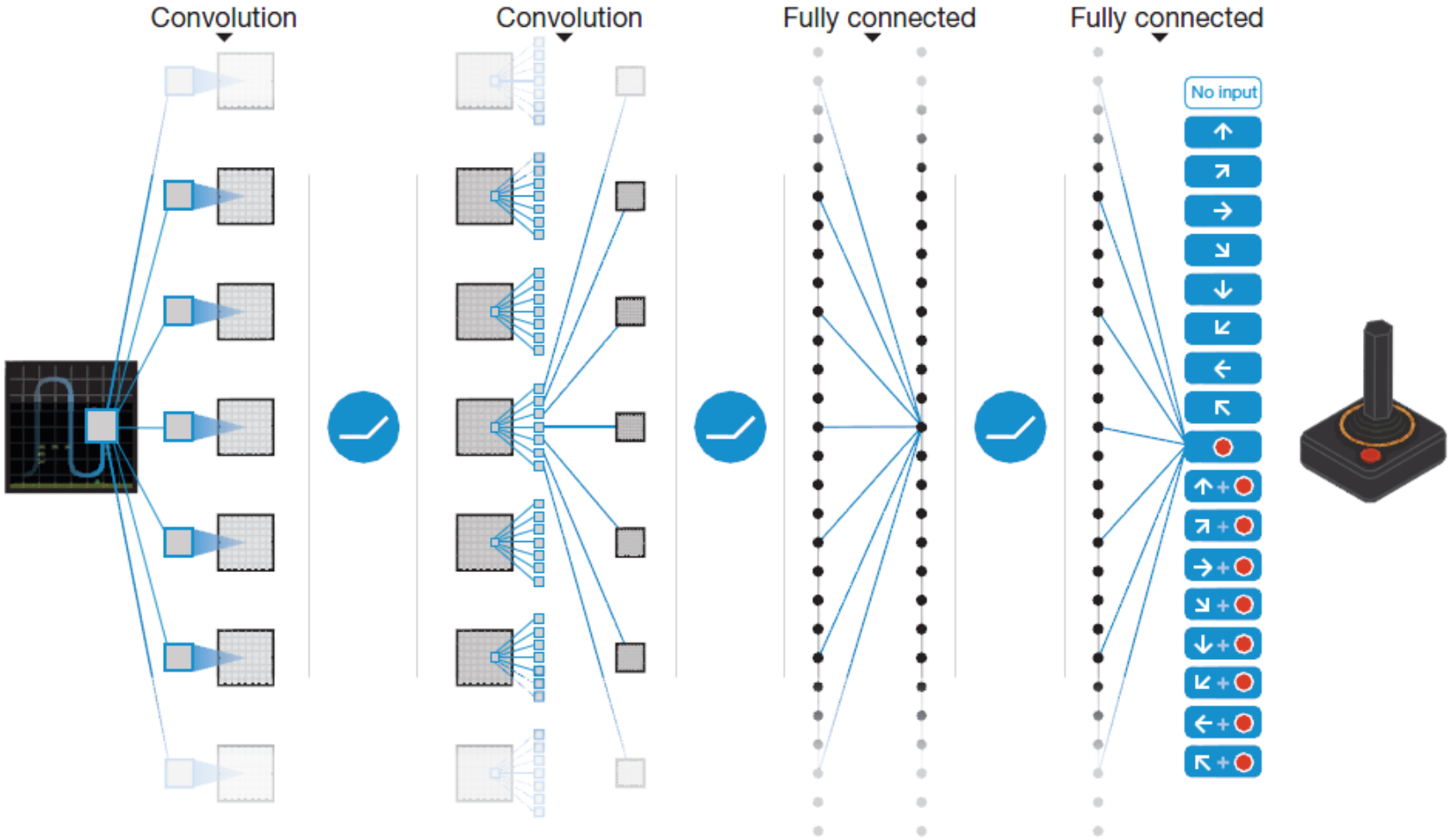
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Fixed length of sequence (last 4 frames)

ϵ-greedy

minibatch = 32

DQN



Input: 84 x 84 x 4 image, 3 convolutional layers, 2 fully connected layers

Target Network (Final Version)

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

    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 $\hat{Q} = Q$ ← Target network update

  End For

End For

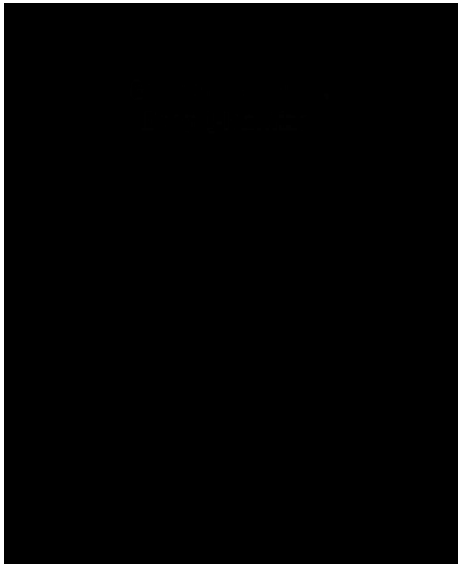
- Issues with Q-learning with function approximation: update that increases $Q(s_t, a_t)$ can also increase $Q(s_{t+1}, a_{t+1})$ for all a_{t+1} and increase target y_i, leading to oscillation or divergence.
- A separate network to generate the target values, y_i.
- A target network makes an algorithm becomes more stable.

*clipping the error term between [-1,1]

Training

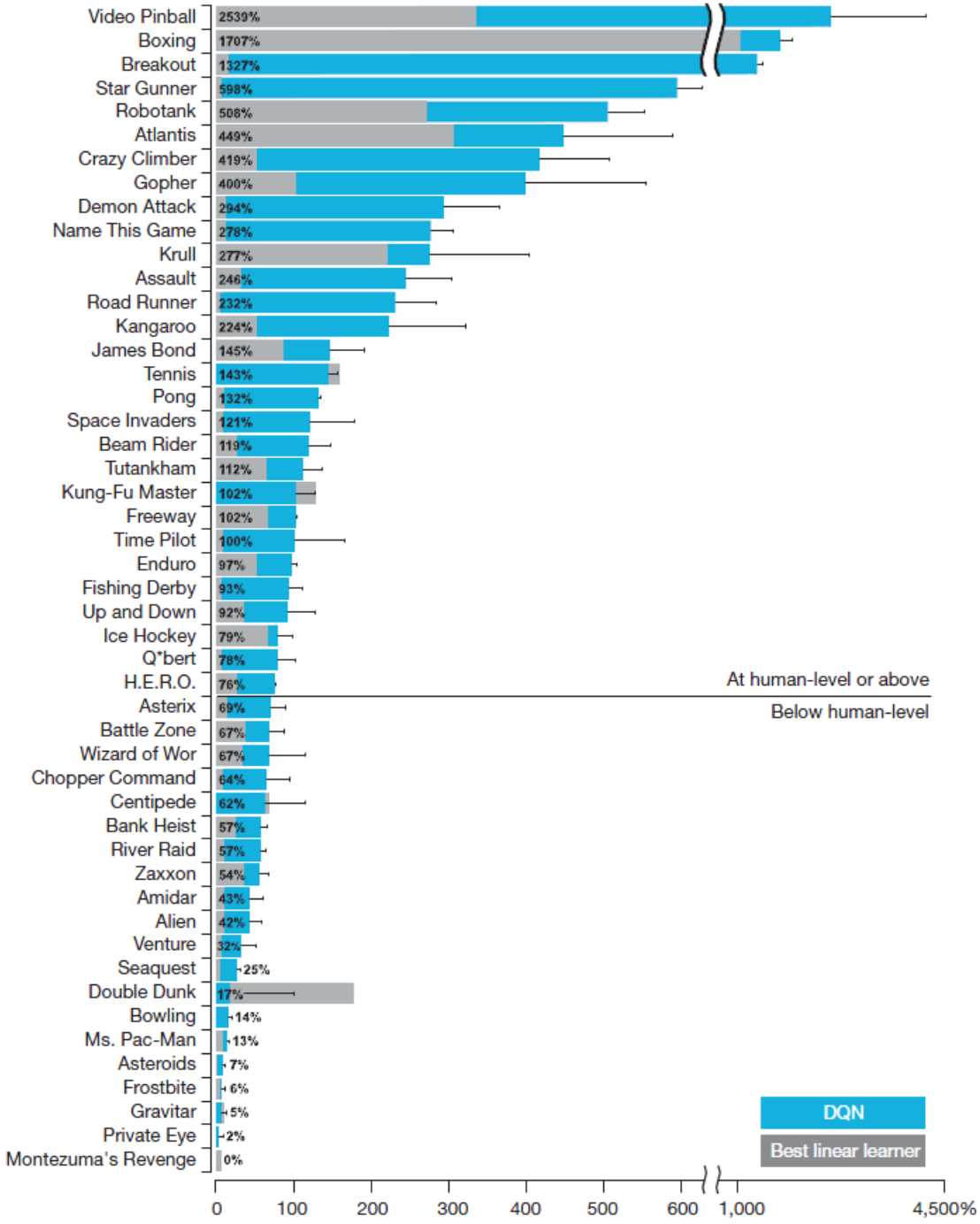
- Different networks for different games but the same architecture and hyperparameters
- RMSProp: divides the learning rate by a running average of recent gradient magnitudes
- Trained for 50 million frames (38 days of gaming)

Results



Breakout

Space Invaders



Key ingredients in DQN

- Experience replay
- Target network
- Error clipping

- and “a lot of engineering”

H. van Hasselt, A. Guez, and D. Silver, "**Deep reinforcement learning with double q-learning**," in Proc. of the AAAI Conference on Artificial Intelligence (AAAI), Feb, 2016.

DOUBLE Q-LEARNING

Double Q-Learning

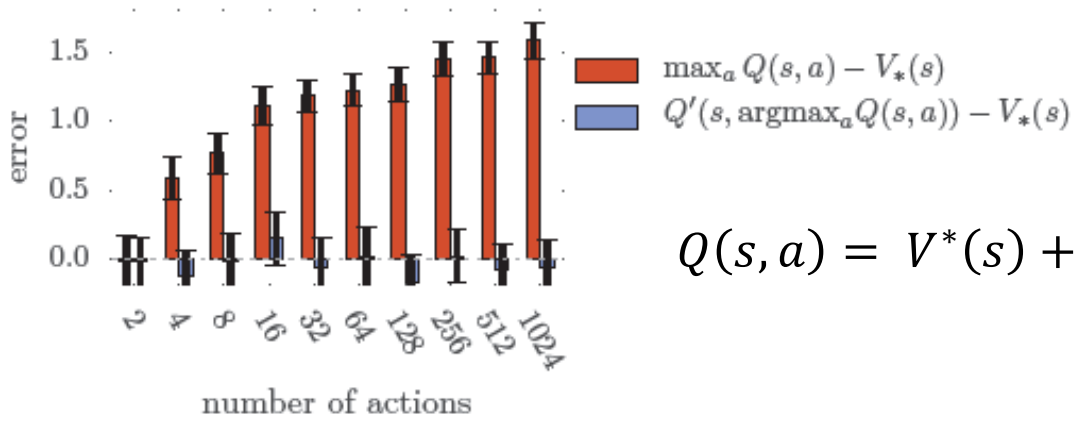
- DQN is likely to select overestimated values -> overoptimistic value estimates
- Decouple the selection from the evaluation

$$Y_t^{\text{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\text{argmax}}\, Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}'_t)$$

selection

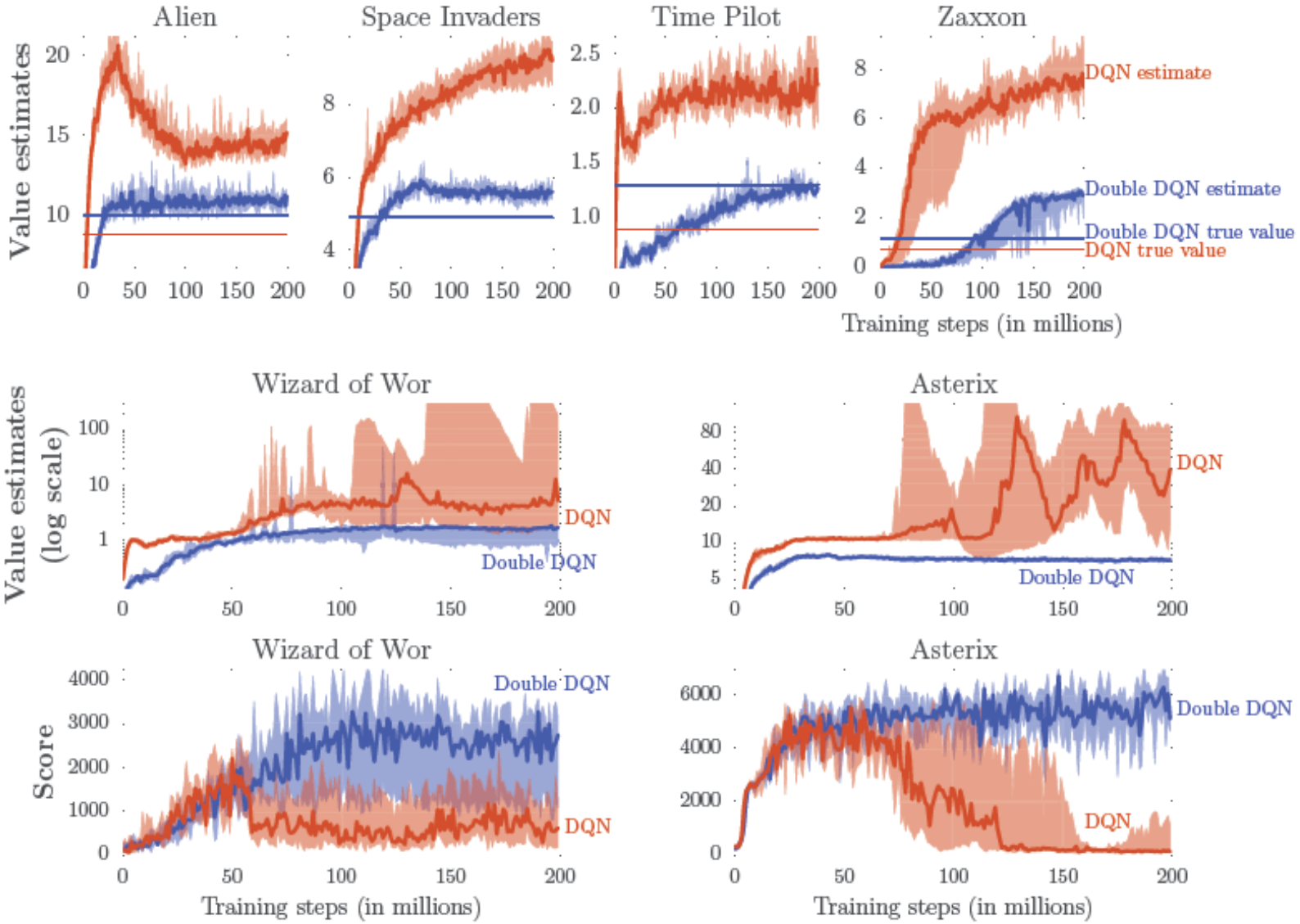
evaluation (uses the target network)

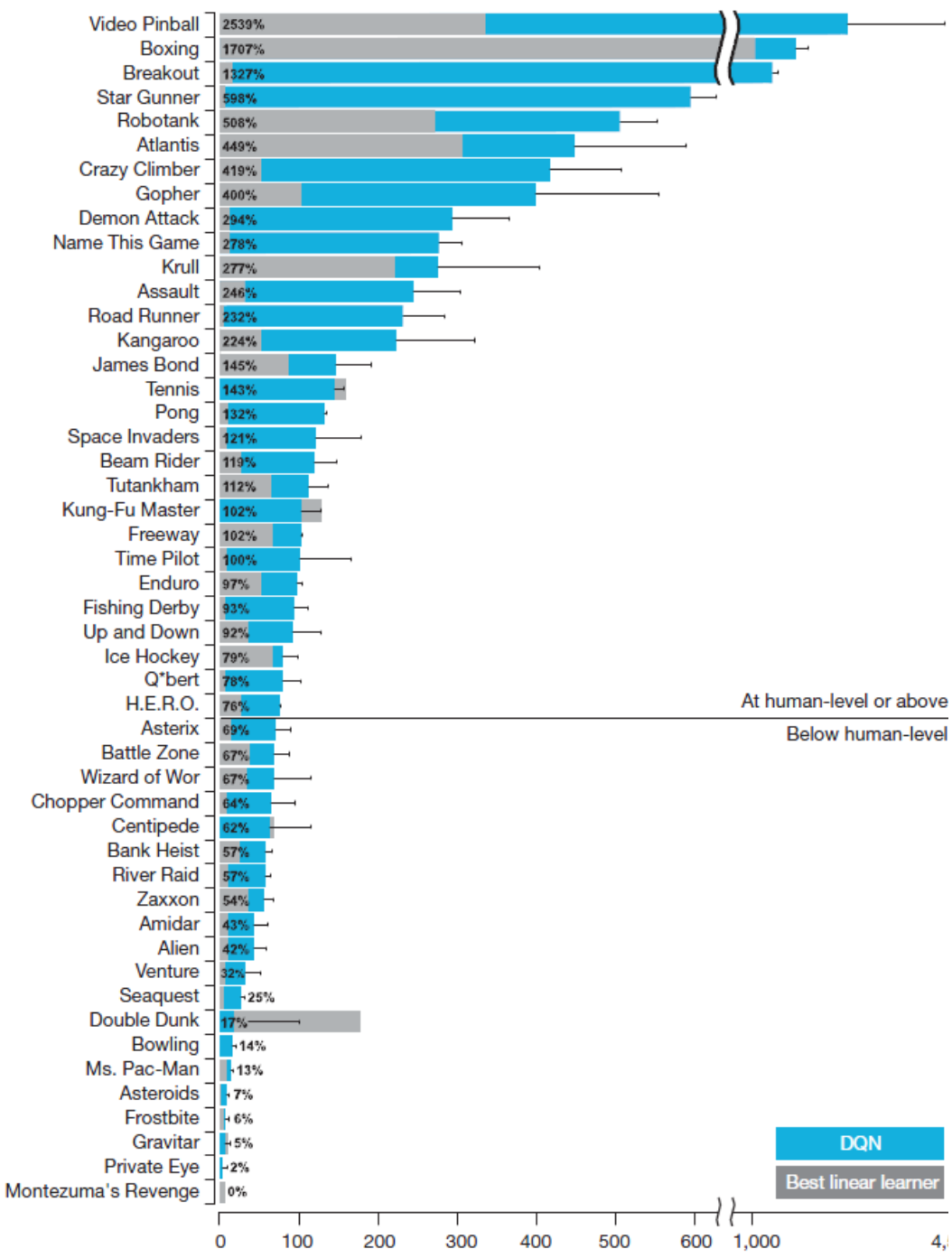
cf. DQN: $Y_t^{\text{Q}} = R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\text{argmax}}\, Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}_t)$

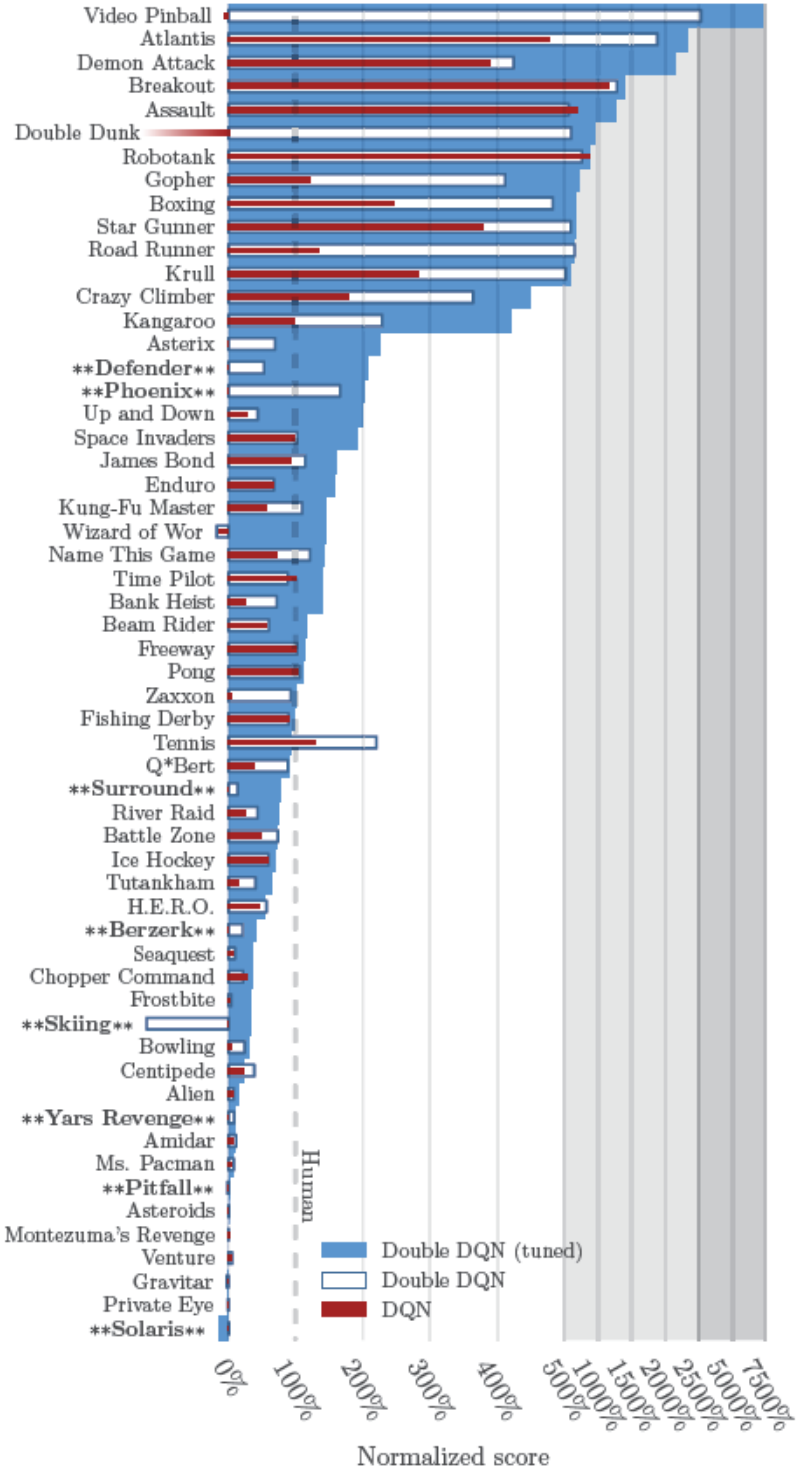


$$Q(s,a) = V^*(s) + \epsilon_a, \quad \epsilon_a \sim_{iid} \mathcal{N}(0,1)$$

Comparison


Alien
Space Invaders
Time Pilot
Zaxxon
Value estimates
DQN estimate
Double DQN estimate
Double DQN true value
DQN true value
Training steps (in millions)
Wizard of Wor
Asterix
Value estimates (log scale)
DQN
Double DQN
Wizard of Wor
Asterix
Score
Double DQN
DQN
Training steps (in millions)


Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%
At human-level or above
Below human-level
DQN
Best linear learner
0
100
200
300
400
500
600
1,000


Video Pinball
Atlantis
Demon Attack
Breakout
Assault
Double Dunk
Robotank
Gopher
Boxing
Star Gunner
Road Runner
Krull
Crazy Climber
Kangaroo
Asterix
Defender
Phoenix
Up and Down
Space Invaders
James Bond
Enduro
Kung-Fu Master
Wizard of Wor
Name This Game
Time Pilot
Bank Heist
Beam Rider
Freeway
Pong
Zaxxon
Fishing Derby
Tennis
Q*Bert
Surround
River Raid
Battle Zone
Ice Hockey
Tutankham
H.E.R.O.
Berzerk
Seaquest
Chopper Command
Frostbite
Skiing
Bowling
Centipede
Alien
Yars Revenge
Amidar
Ms. Pacman
Pitfall
Asteroids
Montezuma's Revenge
Venture
Gravitar
Private Eye
Solaris
Human
Double DQN (tuned)
Double DQN
DQN
0%
100%
200%
300%
400%
500%
1000%
1500%
2000%
2500%
5000%
7500%
Normalized score

T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "**Prioritized Experience Replay**," ICLR, 2016

PRIORITIZED EXPERIENCE REPLAY

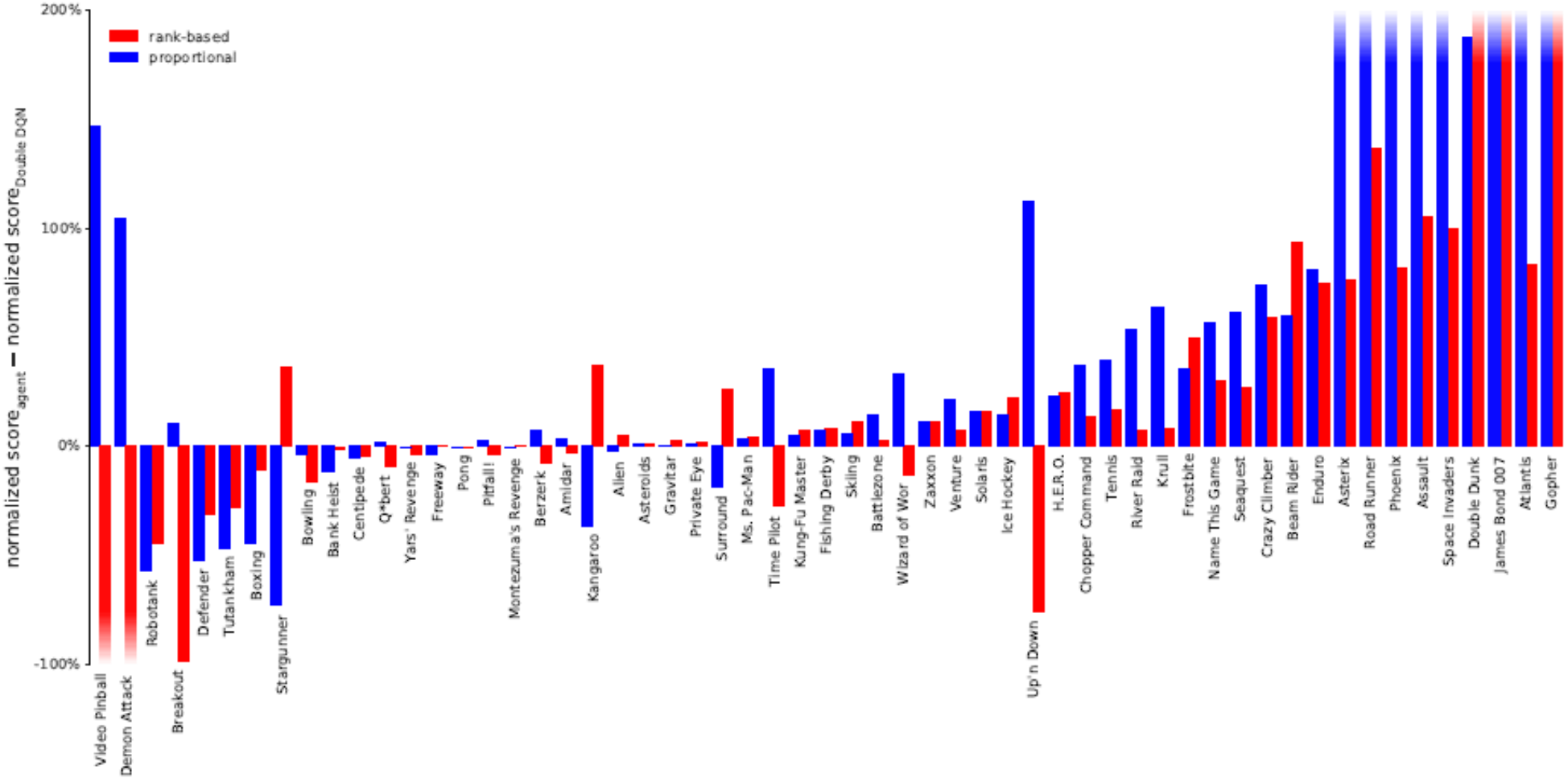
Prioritized Experience Replay

Algorithm 1 Double DQN with proportional prioritization

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T.
Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
Observe S_0 and choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ **to** T **do**
  Observe S_t, R_t, γ_t
  Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
  if $t \equiv 0 \mod K$ **then**
    for $j = 1$ **to** k **do**
      Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
      Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
      Compute TD-error $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
      Update transition priority $p_j \leftarrow |\delta_j|$
      Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
    end for
    Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
    From time to time copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
  end if
  Choose action $A_t \sim \pi_\theta(S_t)$
end for

Uniform if $\alpha = 0$; $\beta \uparrow 1$

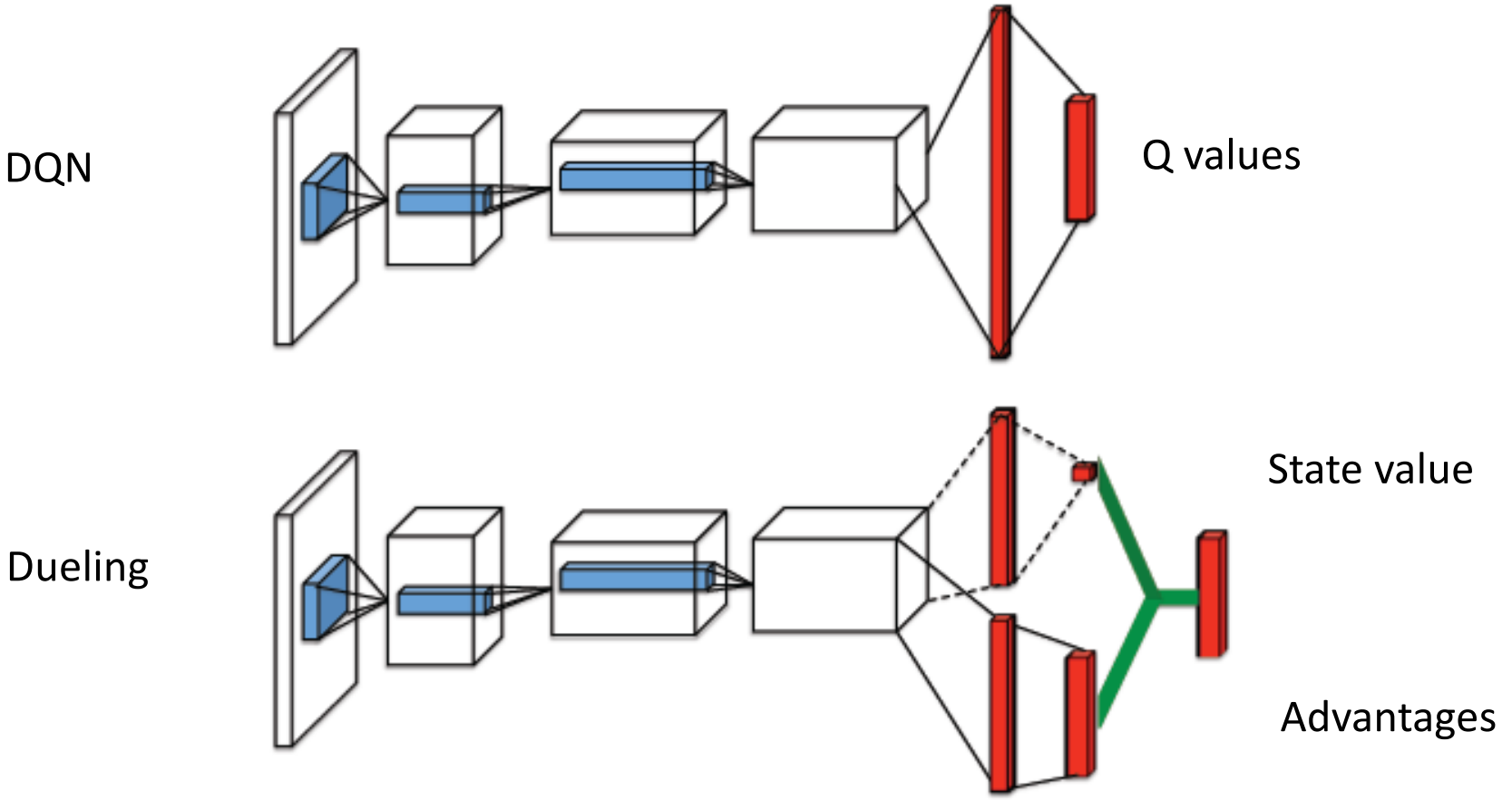
Comparison with Double DQN



Wang, Ziyu, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. "**Dueling network architectures for deep reinforcement learning**." in Proc. of the International Conference on Machine Learning (ICML), Jun, 2016.

DUELING DQN

Dueling Architecture



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

Dueling: Updates state values more often -> Better estimates

Forward Mapping for Q

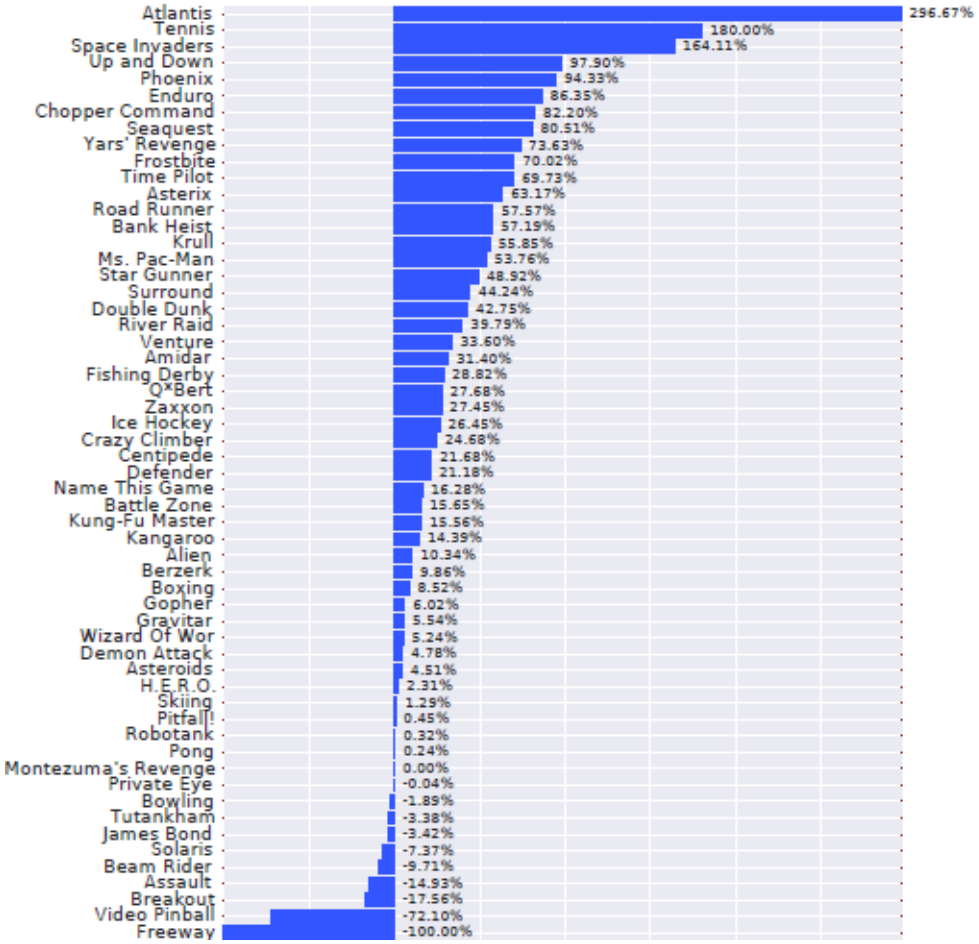
$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a' \in |\mathcal{A}|} A(s,a';\theta,\alpha)\right)$$

- Makes a greedy action have zero advantage
- Addresses the identifiability issue

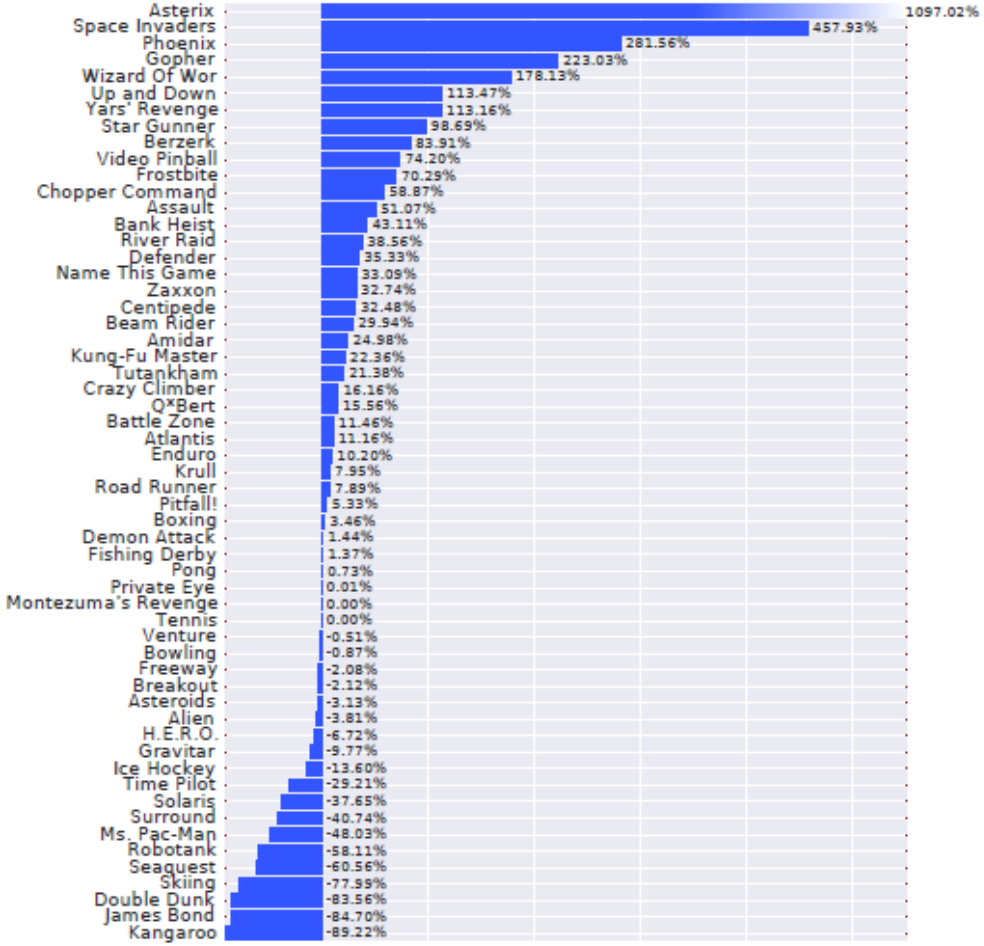
$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\theta,\alpha)\right)$$

- Increases the stability of the algorithm
- Also addresses the identifiability issue

Results

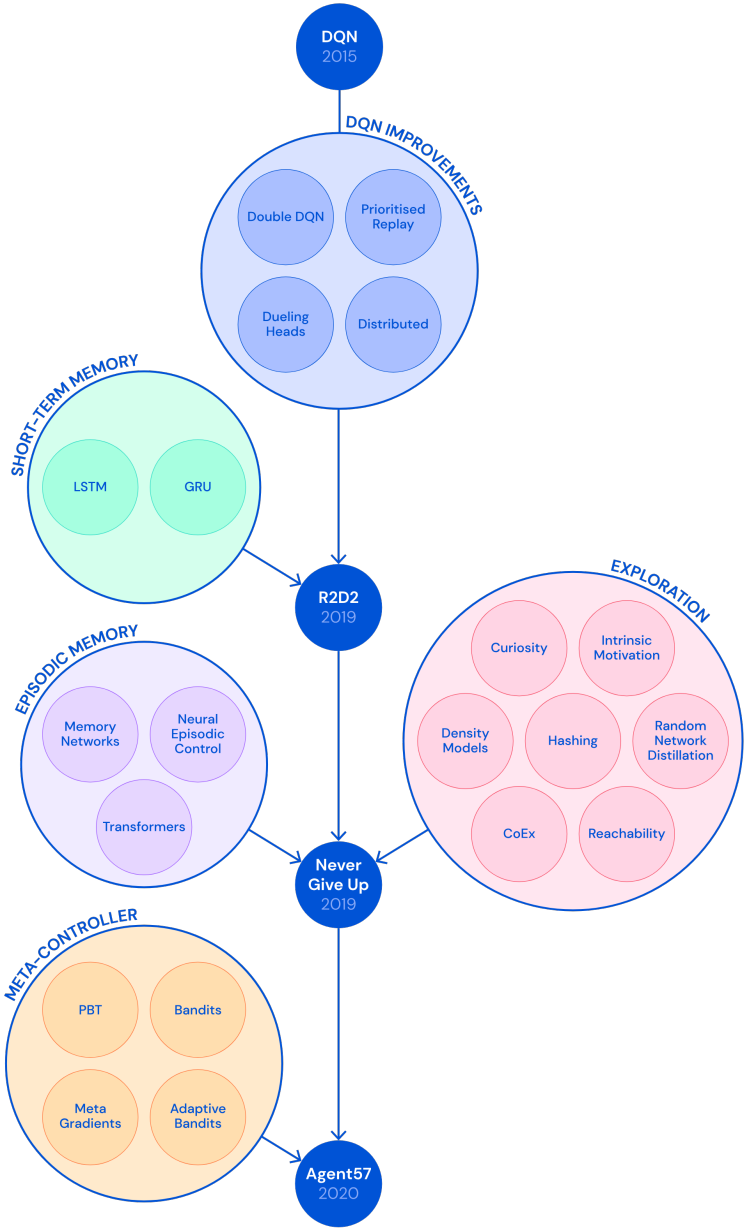


Dueling vs. Double DQN



Dueling vs. Prioritized Double DQN

Agent57



Source:
https://deepmind.google/discover/blog/agent57-outperforming-the-human-atari-benchmark/