Rainbow

Milan Straka

October 31, 2022

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

unless otherwise stated

We will approximate value function v and/or state-value function q, selecting it from a family of functions parametrized by a weight vector $\boldsymbol{w} \in \mathbb{R}^d$.

We denote the approximations as

$$\hat{v}(s; \boldsymbol{w}),$$
$$\hat{q}(s, a; \boldsymbol{w}).$$

We utilize the *Mean Squared Value Error* objective, denoted $\overline{VE}$:

$$\overline{VE}(\boldsymbol{w}) \stackrel{\text{def}}{=} \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \boldsymbol{w})\right]^2,$$

where the state distribution $\mu(s)$ is usually on-policy distribution.

Gradient and Semi-Gradient Methods

The functional approximation (i.e., the weight vector $\boldsymbol w$) is usually optimized using gradient methods, for example as

$$\begin{aligned}\boldsymbol w_{t+1} &\leftarrow \boldsymbol w_t - \frac{1}{2}\alpha\nabla_{\boldsymbol w_t}\big(v_\pi(S_t) - \hat v(S_t; \boldsymbol w_t)\big)^2\\ &\leftarrow \boldsymbol w_t + \alpha\big(v_\pi(S_t) - \hat v(S_t; \boldsymbol w_t)\big)\nabla_{\boldsymbol w_t}\hat v(S_t; \boldsymbol w_t).\end{aligned}$$

As usual, the $v_\pi(S_t)$ is estimated by a suitable sample of a return:

- in Monte Carlo methods, we use episodic return G_t,
- in temporal difference methods, we employ bootstrapping and use one-step return

$$R_{t+1} + [\neg\mathrm{done}]\cdot\gamma\hat v(S_{t+1}; \boldsymbol w)$$

  or an n-step return.

Deep Q Network

Off-policy Q-learning algorithm with a convolutional neural network function approximation of action-value function.

Training can be extremely brittle (and can even diverge).

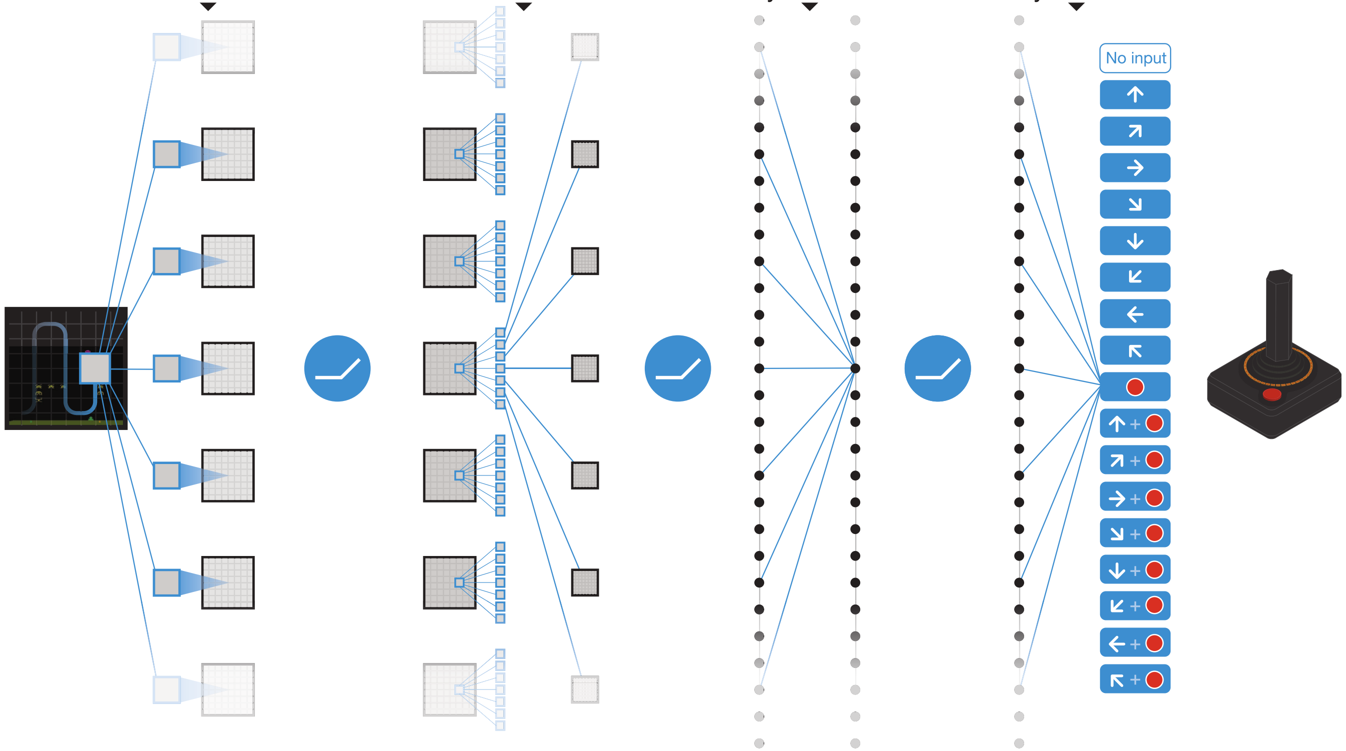

Figure 1 of "Human-level control through deep reinforcement learning" by Volodymyr Mnih et al.

Deep Q Networks

- Preprocessing: 210×160 128-color images are converted to grayscale and then resized to 84×84.
- Frame skipping technique is used, i.e., only every 4^{th} frame (out of 60 per second) is considered, and the selected action is repeated on the other frames.
- Input to the network are last 4 frames (considering only the frames kept by frame skipping), i.e., an image with 4 channels.
- The network is fairly standard, performing
  - 32 filters of size 8×8 with stride 4 and ReLU,
  - 64 filters of size 4×4 with stride 2 and ReLU,
  - 64 filters of size 3×3 with stride 1 and ReLU,
  - fully connected layer with 512 units and ReLU,
  - output layer with 18 output units (one for each action)

- Network is trained with RMSProp to minimize the following loss:

$$\mathcal{L} \stackrel{\text{def}}{=} \mathbb{E}_{(s,a,r,s')\sim\text{data}}\left[\left(r + [\neg\text{done}] \cdot \gamma \max_{a'} Q(s', a'; \bar{\boldsymbol\theta}) - Q(s, a; \boldsymbol\theta)\right)^2\right].$$

- An ε-greedy behavior policy is utilized (starts at $\varepsilon = 1$ and gradually decreases to 0.1).

Important improvements:

- **experience replay**: the generated episodes are stored in a buffer as (s, a, r, s') quadruples, and for training a transition is sampled uniformly (off-policy training);
- separate **target network** $\bar{\boldsymbol\theta}$: to prevent instabilities, a separate *target network* is used to estimate one-step returns. The weights are not trained, but copied from the trained network after a fixed number of gradient updates;
- reward clipping: because rewards have wildly different scale in different games, all positive rewards are replaced by $+1$ and negative by -1; life loss is used as end of episode.
  - furthermore, $(r + [\neg\text{done}] \cdot \gamma \max_{a'} Q(s', a'; \bar{\boldsymbol\theta}) - Q(s, a; \boldsymbol\theta))$ is also clipped to $[-1, 1]$ (i.e., a smooth_{L_1} loss or Huber loss).

Deep Q Networks Hyperparameters

Hyperparameter	Value
minibatch size	32
replay buffer size	1M
target network update frequency	10k
discount factor	0.99
training frames	50M
RMSProp learning rate and momentum	0.00025, 0.95
initial ε, final ε (linear decay) and frame of final ε	1.0, 0.1, 1M
replay start size	50k
no-op max	30

There have been many suggested improvements to the DQN architecture. In the end of 2017, the *Rainbow: Combining Improvements in Deep Reinforcement Learning* paper combines 6 of them into a single architecture they call **Rainbow**.

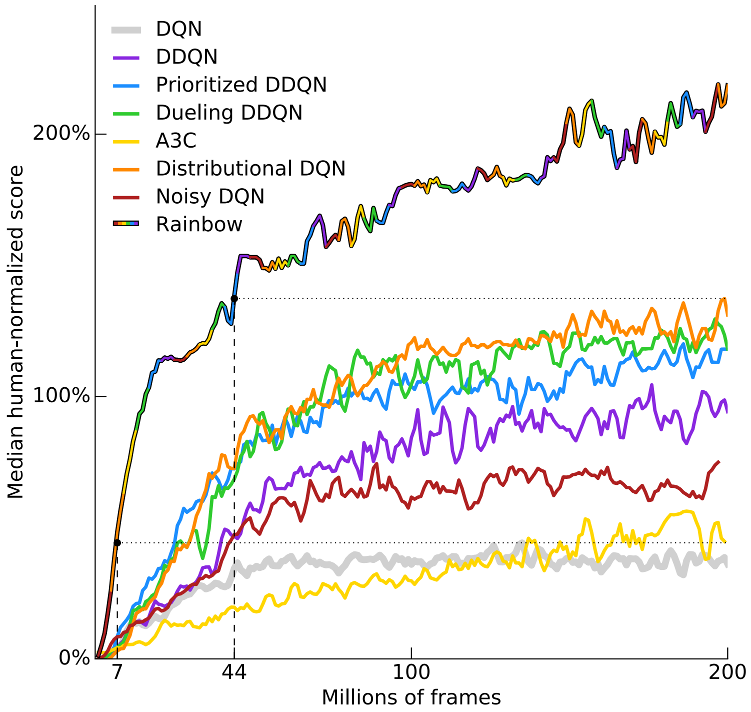

Figure 1 of "Rainbow: Combining Improvements in Deep Reinforcement Learning" by Matteo Hessel et al.

Because behaviour policy in Q-learning is ε-greedy variant of the target policy, the same samples (up to ε-greedy) determine both the maximizing action and estimate its value.

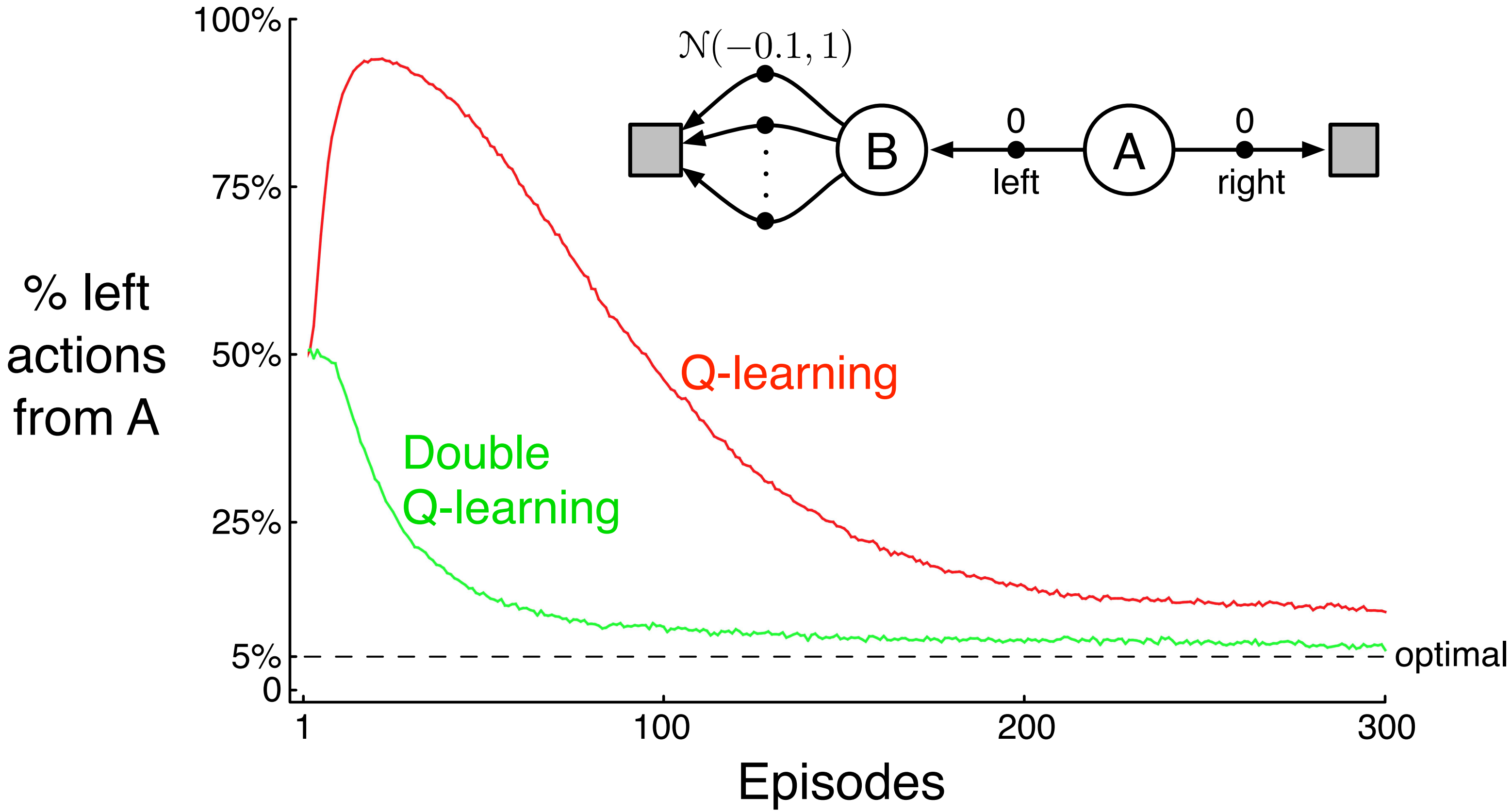

Figure 6.5 of "Reinforcement Learning: An Introduction, Second Edition".

Double Q-learning

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, such that $Q(\textit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:
      $$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$
    else:
      $$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S', a)\big) - Q_2(S, A)\Big)$$
    $S \leftarrow S'$
  until S is terminal

Modification of Algorithm 6.7 of "Reinforcement Learning: An Introduction, Second Edition" (replacing S+ by S).

Double Q-learning

Similarly to double Q-learning, instead of

$$r + \gamma \max_{a'} Q(s', a'; \bar{\boldsymbol\theta}) - Q(s, a; \boldsymbol\theta),$$

we minimize

$$r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \boldsymbol\theta); \bar{\boldsymbol\theta}) - Q(s, a; \boldsymbol\theta).$$

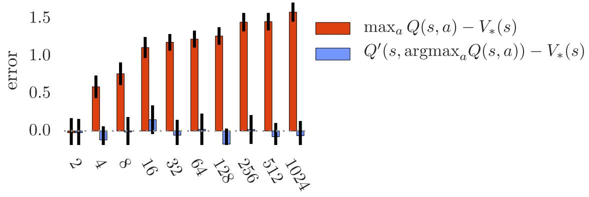

Figure 1: The orange bars show the bias in a single Q-learning update when the action values are $Q(s,a) = V_*(s) + \epsilon_a$ and the errors $\{\epsilon_a\}_{a=1}^m$ are independent standard normal random variables. The second set of action values Q', used for the blue bars, was generated identically and independently. All bars are the average of 100 repetitions.

Figure 1 of "Deep Reinforcement Learning with Double Q-learning" by Hado van Hasselt et al.

Double Q-learning

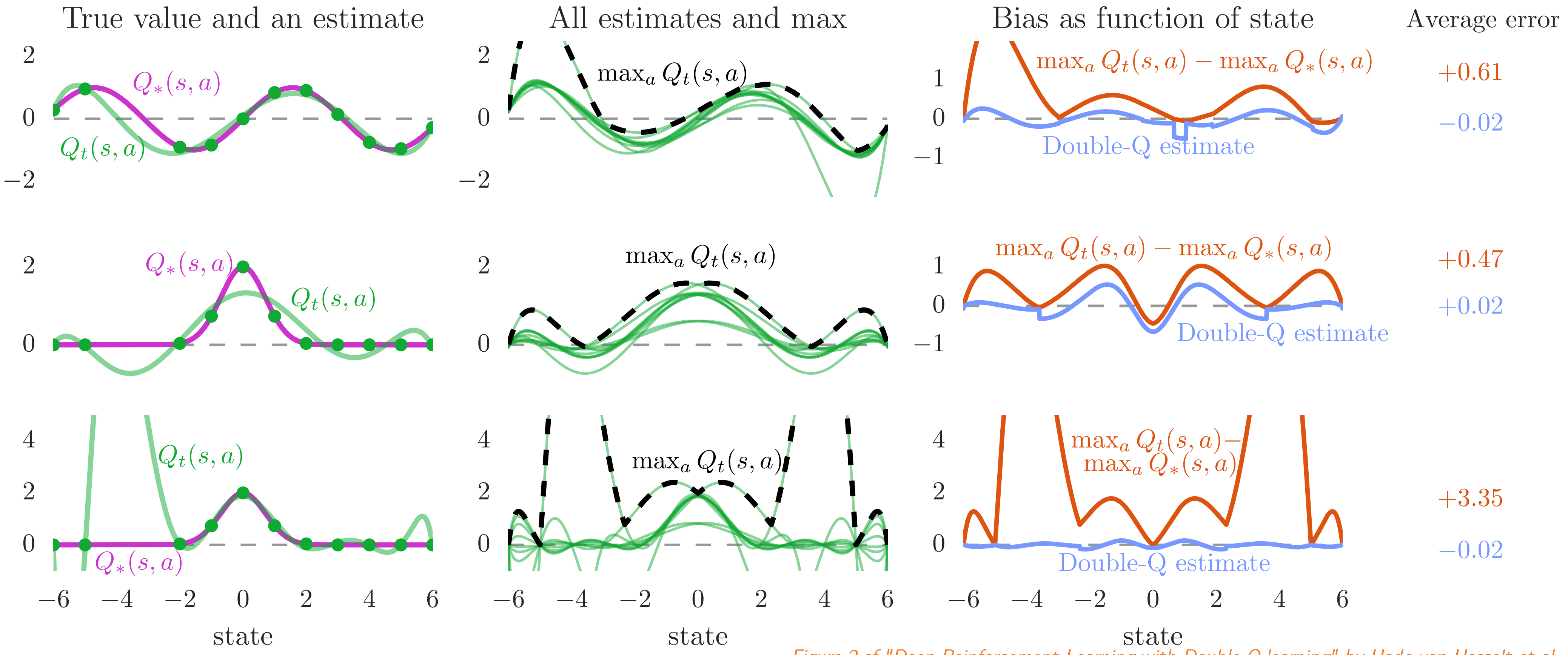

Figure 2 of "Deep Reinforcement Learning with Double Q-learning" by Hado van Hasselt et al.

Double Q-learning

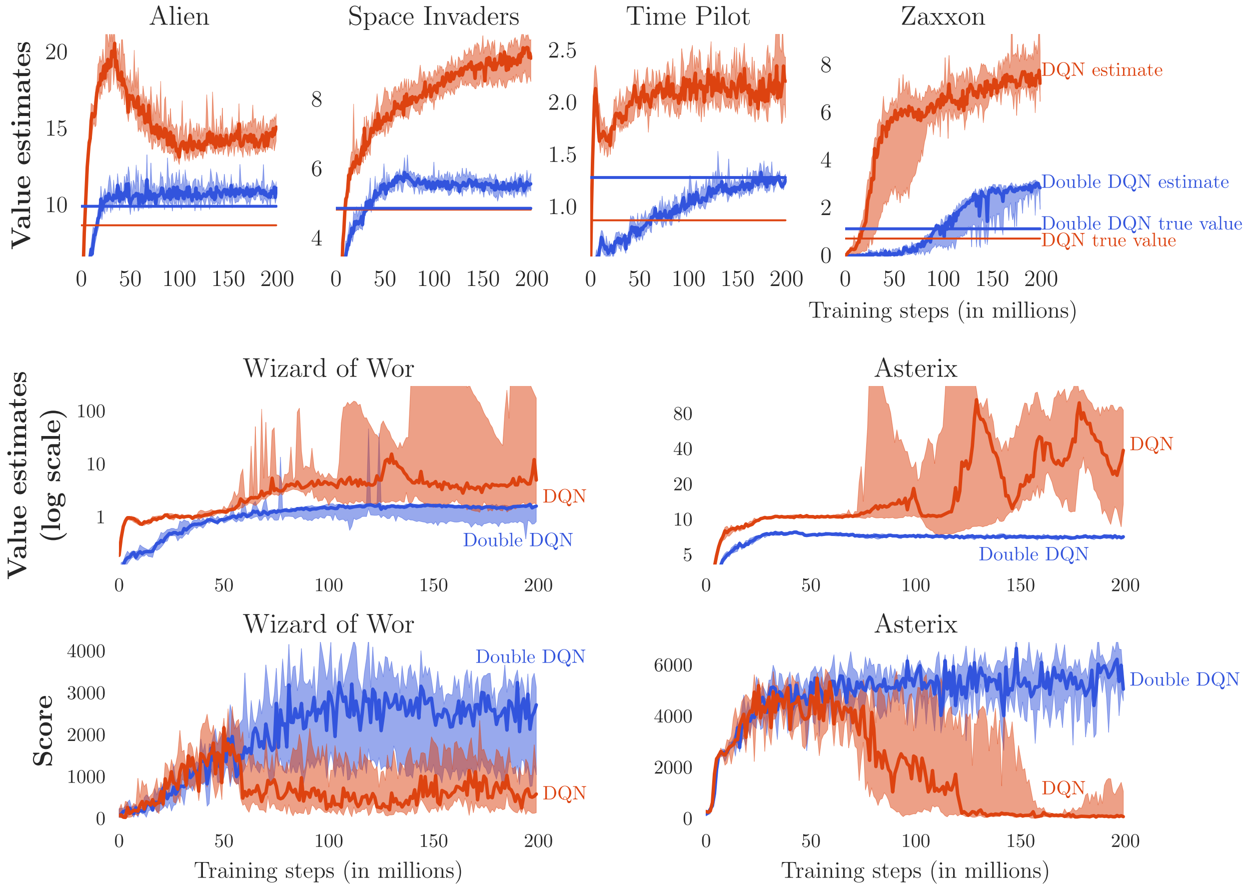

Figure 3 of "Deep Reinforcement Learning with Double Q-learning" by Hado van Hasselt et al.

Double Q-learning

Performance on episodes taking at most 5 minutes and no-op starts on 49 games:

	DQN	Double DQN
Median	93.5%	114.7%
Mean	241.1%	330.3%

Table 1 of "Deep Reinforcement Learning with Double Q-learning" by Hado van Hasselt et al.

Performance on episodes taking at most 30 minutes and using human starts on 49 games:

	DQN	Double DQN	Double DQN (tuned)
Median	47.5%	88.4%	116.7%
Mean	122.0%	273.1%	475.2%

Table 2 of "Deep Reinforcement Learning with Double Q-learning" by Hado van Hasselt et al.

Prioritized Replay

Instead of sampling the transitions uniformly from the replay buffer, we instead prefer those with a large TD error. Therefore, we sample transitions according to their probability

$$p_t \propto \Big|r + \gamma \max_{a'} Q(s', a'; \bar{\boldsymbol\theta}) - Q(s, a; \boldsymbol\theta)\Big|^\omega,$$

where ω controls the shape of the distribution (which is uniform for $\omega = 0$ and corresponds to TD error for $\omega = 1$).

New transitions are inserted into the replay buffer with maximum probability to support exploration of all encountered transitions.

When combined with DDQN, the probabilities are naturally computed as

$$p_t \propto \Big|r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \boldsymbol\theta); \bar{\boldsymbol\theta}) - Q(s, a; \boldsymbol\theta)\Big|^\omega,$$

Prioritized Replay

Because we now sample transitions according to p_t instead of uniformly, on-policy distribution and sampling distribution differ. To compensate, we therefore utilize importance sampling with ratio

$$\rho_t = \left(\frac{1/N}{p_t}\right)^\beta.$$

The authors utilize in fact "for stability reasons"

$$\rho_t / \max_i \rho_i.$$

Prioritized 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

Algorithm 1 of "Prioritized Experience Replay" by Tom Schaul et al.

Dueling Networks

Instead of computing directly $Q(s, a; \boldsymbol\theta)$, we compose it from the following quantities:

- value function for a given state s,
- advantage function computing an **advantage** of using action a in state s.

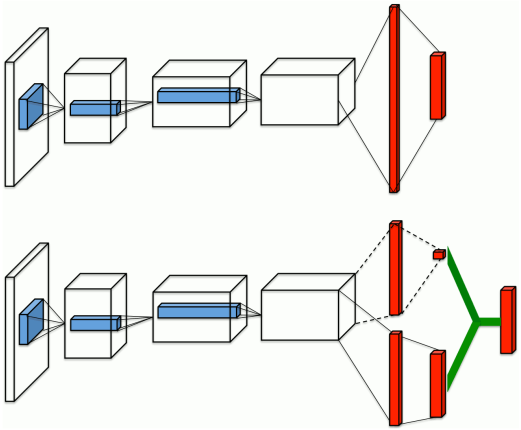

Figure 1 of "Dueling Network Architectures for Deep Reinforcement Learning" by Ziyu Wang et al.

$$Q(s, a) \stackrel{\text{def}}{=} V\big(f(s;\zeta);\eta\big) + A\big(f(s;\zeta), a; \psi\big) - \frac{\sum_{a' \in \mathcal{A}} A(f(s;\zeta), a'; \psi)}{|\mathcal{A}|}$$

Dueling Networks

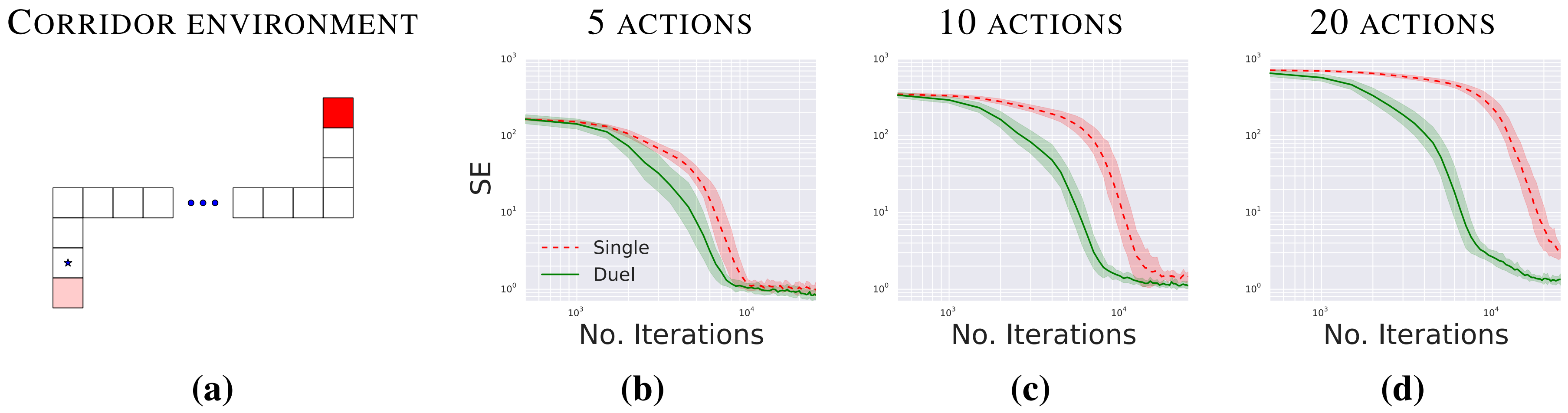

Figure 3. **(a)** The corridor environment. The star marks the starting state. The redness of a state signifies the reward the agent receives upon arrival. The game terminates upon reaching either reward state. The agent's actions are going up, down, left, right and no action. Plots **(b), (c)** and **(d)** shows squared error for policy evaluation with 5, 10, and 20 actions on a log-log scale. The dueling network (Duel) consistently outperforms a conventional single-stream network (Single), with the performance gap increasing with the number of actions.

Figure 3 of "Dueling Network Architectures for Deep Reinforcement Learning" by Ziyu Wang et al.

Dueling Networks

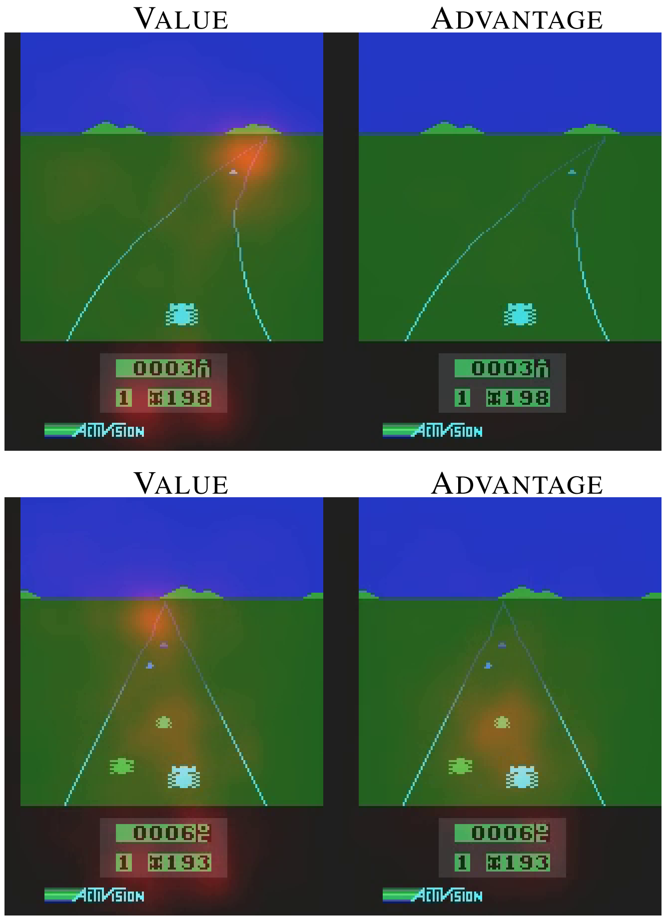

Figure 2 of "Dueling Network Architectures for Deep Reinforcement Learning" by Ziyu Wang et al.

Dueling Networks

Results on all 57 games (retraining the original DQN on the 8 missing games). `Single` refers to DDQN with a direct computation of $Q(s, a; \boldsymbol\theta)$, `Clip` corresponds to gradient clipping to norm at most 10.

	30 no-ops		Human Starts	
	Mean	**Median**	**Mean**	**Median**
Prior. Duel Clip	**591.9%**	**172.1%**	**567.0%**	**115.3%**
Prior. Single	434.6%	123.7%	386.7%	112.9%
Duel Clip	**373.1%**	**151.5%**	**343.8%**	**117.1%**
Single Clip	341.2%	132.6%	302.8%	114.1%
Single	307.3%	117.8%	332.9%	110.9%
Nature DQN	227.9%	79.1%	219.6%	68.5%

Table 1 of "Dueling Network Architectures for Deep Reinforcement Learning" by Ziyu Wang et al.

Multi-step Learning

Instead of Q-learning, we use n-step variant of Q-learning, which estimates return as

$$\sum_{i=1}^{n} \gamma^{i-1} R_i + \gamma^n \max_{a'} Q(s', a'; \bar{\boldsymbol\theta}).$$

This changes the off-policy algorithm to on-policy (because the "inner" actions are sampled from the behaviour distribution, but should follow the target distribution); however, it is not discussed in any way by the authors.

Noisy Nets

Noisy Nets are neural networks whose weights and biases are perturbed by a parametric function of a noise.

The parameters $\boldsymbol\theta$ of a regular neural network are in Noisy nets represented as

$$\boldsymbol\theta \approx \boldsymbol\mu + \boldsymbol\sigma \odot \boldsymbol\varepsilon,$$

where $\boldsymbol\varepsilon$ is zero-mean noise with fixed statistics. We therefore learn the parameters $(\boldsymbol\mu, \boldsymbol\sigma)$.

A fully connected layer with parameters $(\boldsymbol w, \boldsymbol b)$,

$$\boldsymbol y = \boldsymbol w \boldsymbol x + \boldsymbol b,$$

is represented in the following way in Noisy nets:

$$\boldsymbol y = (\boldsymbol\mu_w + \boldsymbol\sigma_w \odot \boldsymbol\varepsilon_w)\boldsymbol x + (\boldsymbol\mu_b + \boldsymbol\sigma_b \odot \boldsymbol\varepsilon_b).$$

Noisy Nets

The noise ε can be for example independent Gaussian noise. However, for performance reasons, factorized Gaussian noise is used to generate a matrix of noise. If $\varepsilon_{i,j}$ is noise corresponding to a layer with i inputs and j outputs, we generate independent noise ε_i for input neurons, independent noise ε_j for output neurons, and set

$$\varepsilon_{i,j} = f(\varepsilon_i) f(\varepsilon_j)$$

for $f(x) = \operatorname{sign}(x)\sqrt{|x|}$.

The authors generate noise samples for every batch, sharing the noise for all batch instances.

Deep Q Networks

When training a DQN, ε-greedy is no longer used and all policies are greedy, and all fully connected layers are parametrized as noisy nets.

Noisy Nets

	Baseline Mean	Baseline Median	NoisyNet Mean	NoisyNet Median	Improvement (On median)
DQN	319	83	**379**	**123**	48%
Dueling	524	132	**633**	**172**	30%
A3C	293	80	**347**	**94**	18%

Table 1 of "Noisy Networks for Exploration" by Meire Fortunato et al.

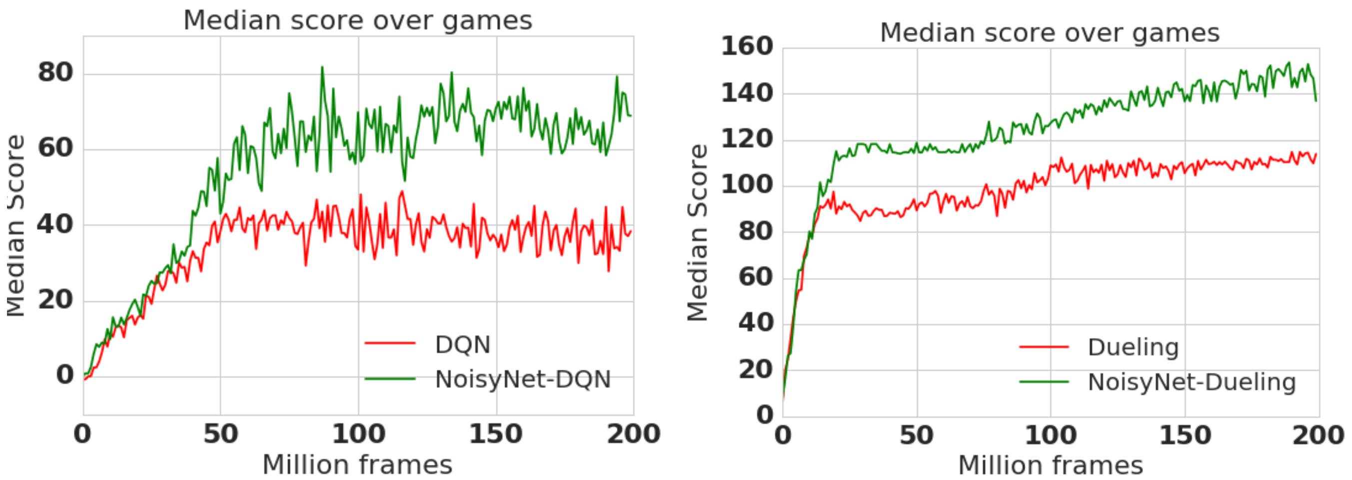

Figure 2 of "Noisy Networks for Exploration" by Meire Fortunato et al.

Noisy Nets

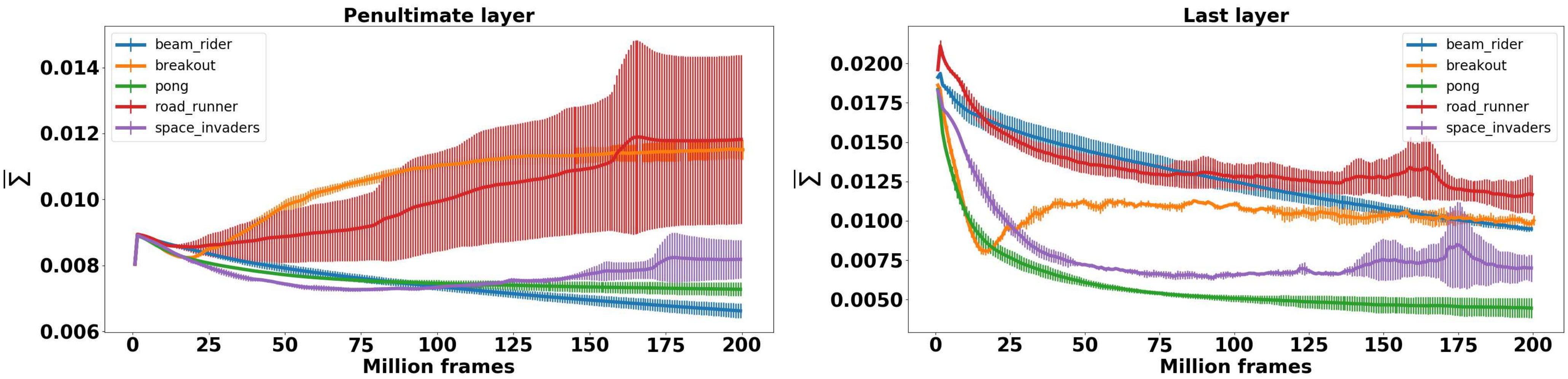

Figure 3: Comparison of the learning curves of the average noise parameter $\bar{\Sigma}$ across five Atari games in NoisyNet-DQN. The results are averaged across 3 seeds and error bars (+/- standard deviation) are plotted.

Figure 3 of "Noisy Networks for Exploration" by Meire Fortunato et al.

Distributional RL

Instead of an expected return $Q(s, a)$, we could estimate the distribution of expected returns $Z(s, a)$.

These distributions satisfy a distributional Bellman equation:

$$Z_\pi(s, a) = R(s, a) + \gamma \mathbb{E}_{s', a'} Z(s', a').$$

The authors of the paper prove similar properties of the distributional Bellman operator compared to the regular Bellman operator, mainly being a contraction under a suitable metric (Wasserstein metric).

Distributional RL

The distribution of returns is modeled as a discrete distribution parametrized by the number of atoms $N \in \mathbb{N}$ and by $V_\textrm{MIN}, V_\textrm{MAX} \in \mathbb{R}$. Support of the distribution are atoms

$$\{z_i \stackrel{\textrm{def}}{=} V_\textrm{MIN} + i\Delta z : 0 \le i < N\} \quad \textrm{for } \Delta z \stackrel{\textrm{def}}{=} \frac{V_\textrm{MAX} - V_\textrm{MIN}}{N-1}.$$

The atom probabilities are predicted using a $\operatorname{softmax}$ distribution as

$$Z_{\boldsymbol\theta}(s, a) = \left\{ z_i \textrm{ with probability } p_i = \frac{e^{f_i(s,a;\boldsymbol\theta)}}{\sum_j e^{f_j(s,a;\boldsymbol\theta)}} \right\}.$$

Distributional RL

After the Bellman update, the support of the distribution $R(s,a) + \gamma Z(s', a')$ is not the same as the original support. We therefore project it to the original support by proportionally mapping each atom of the Bellman update to immediate neighbors in the original support.

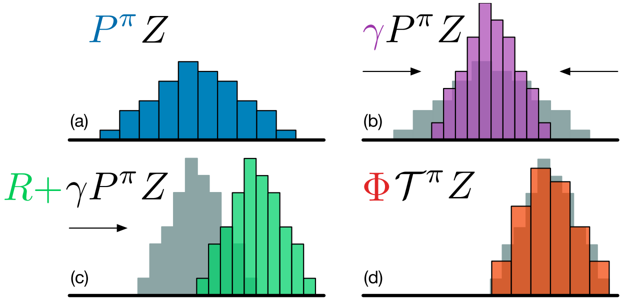

Figure 1 of "A Distributional Perspective on Reinforcement Learning" by Marc G. Bellemare et al.

$$\Phi\big(R(s,a) + \gamma Z(s', a')\big)_i \stackrel{\text{def}}{=} \sum_{j=1}^N \left[1 - \frac{\left|[r + \gamma z_j]_{V_\text{MIN}}^{V_\text{MAX}} - z_i\right|}{\Delta z}\right]_0^1 p_j(s', a').$$

The network is trained to minimize the Kullbeck-Leibler divergence between the current distribution and the (mapped) distribution of the one-step update

$$D_\text{KL}\Big(\Phi\big(R + \gamma Z_{\bar{\boldsymbol\theta}}\big(s', \arg\max_{a'} \mathbb{E} Z_{\bar{\boldsymbol\theta}}(s', a')\big)\big) \Big\| Z_{\boldsymbol\theta}\big(s, a\big)\Big).$$

Distributional RL

Algorithm 1 Categorical Algorithm

input A transition $x_t, a_t, r_t, x_{t+1}, \gamma_t \in [0, 1]$
  $Q(x_{t+1}, a) := \sum_i z_i p_i(x_{t+1}, a)$
  $a^* \leftarrow \arg\max_a Q(x_{t+1}, a)$
  $m_i = 0, \quad i \in 0, \dots, N-1$
  for $j \in 0, \dots, N-1$ **do**
    # Compute the projection of $\hat{\mathcal{T}} z_j$ onto the support $\{z_i\}$
    $\hat{\mathcal{T}} z_j \leftarrow [r_t + \gamma_t z_j]_{V_{\text{MIN}}}^{V_{\text{MAX}}}$
    $b_j \leftarrow (\hat{\mathcal{T}} z_j - V_{\text{MIN}})/\Delta z$ # $b_j \in [0, N-1]$
    $l \leftarrow \lfloor b_j \rfloor, u \leftarrow \lceil b_j \rceil$
    # 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

Algorithm 1 of "A Distributional Perspective on Reinforcement Learning" by Marc G. Bellemare et al.

Distributional RL

	Mean	Median	> H.B.	> DQN
DQN	228%	79%	24	0
DDQN	307%	118%	33	43
DUEL.	373%	151%	37	50
PRIOR.	434%	124%	39	48
PR. DUEL.	592%	172%	39	44
C51	**701%**	**178%**	**40**	**50**

Figure 6 of "A Distributional Perspective on Reinforcement Learning" by Marc G. Bellemare et al.

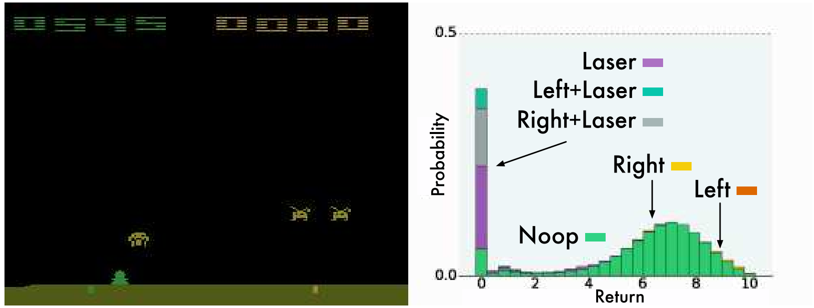

Figure 4. Learned value distribution during an episode of SPACE INVADERS. Different actions are shaded different colours. Returns below 0 (which do not occur in SPACE INVADERS) are not shown here as the agent assigns virtually no probability to them.

Figure 4 of "A Distributional Perspective on Reinforcement Learning" by Marc G. Bellemare et al.

Distributional RL

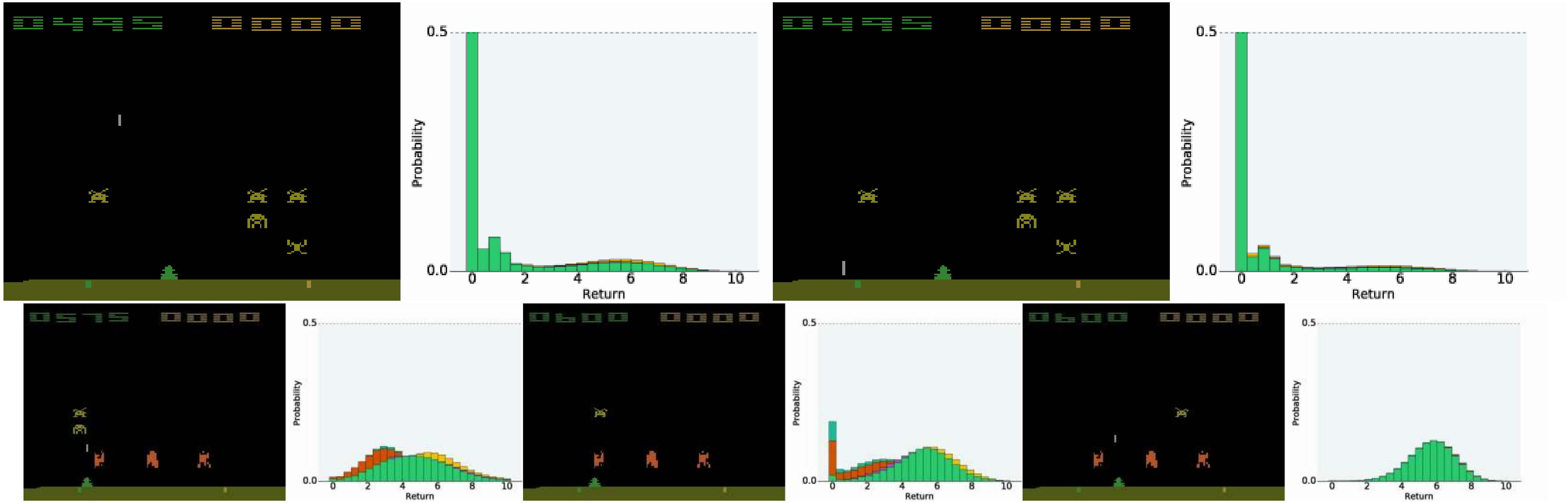

Figure 18. SPACE INVADERS: Top-Left: Multi-modal distribution with high uncertainty. Top-Right: Subsequent frame, a more certain demise. Bottom-Left: Clear difference between actions. Bottom-Middle: Uncertain survival. Bottom-Right: Certain success.

Figure 18 of "A Distributional Perspective on Reinforcement Learning" by Marc G. Bellemare et al.

Distributional RL

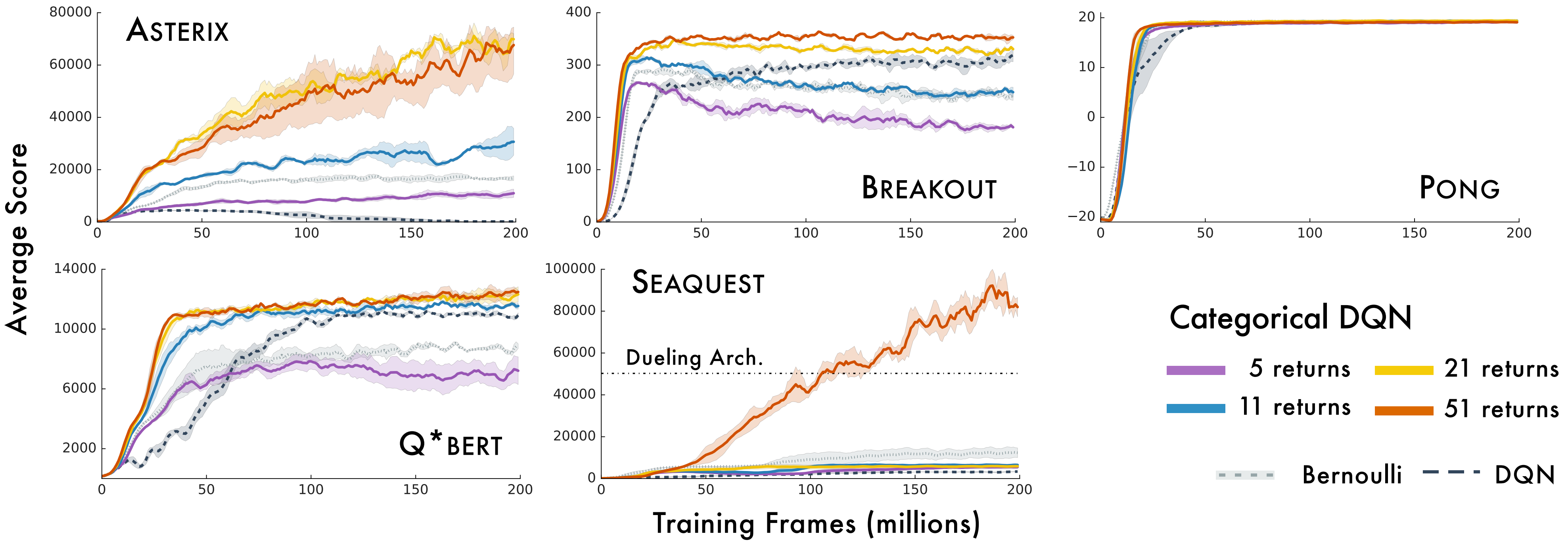

Figure 3. Categorical DQN: Varying number of atoms in the discrete distribution. Scores are moving averages over 5 million frames.

Figure 3 of "A Distributional Perspective on Reinforcement Learning" by Marc G. Bellemare et al.

Rainbow combines all described DQN extensions. Instead of 1-step updates, n-step updates are utilized, and KL divergence of the current and target return distribution is minimized:

$$D_{\mathrm{KL}}\Big(\Phi\big(\textstyle\sum_{i=0}^{n-1}\gamma^i R_{t+i+1} + \gamma^n Z_{\bar{\boldsymbol\theta}}\big(S_{t+n}, \arg\max_{a'} \mathbb{E} Z_{\boldsymbol\theta}(S_{t+n}, a')\big)\big)\Big\| Z(S_t, A_t)\Big).$$

The prioritized replay chooses transitions according to the probability

$$p_t \propto D_{\mathrm{KL}}\Big(\Phi\big(\textstyle\sum_{i=0}^{n-1}\gamma^i R_{t+i+1} + \gamma^n Z_{\bar{\boldsymbol\theta}}\big(S_{t+n}, \arg\max_{a'} \mathbb{E} Z_{\boldsymbol\theta}(S_{t+n}, a')\big)\big)\Big\| Z(S_t, A_t)\Big)^w.$$

Network utilizes dueling architecture feeding the shared representation $f(s;\zeta)$ into value computation $V(f(s;\zeta);\eta)$ and advantage computation $A_i(f(s;\zeta), a;\psi)$ for atom z_i, and the final probability of atom z_i in state s and action a is computed as

$$p_i(s,a) \stackrel{\text{def}}{=} \frac{e^{V_i(f(s;\zeta);\eta)+A_i(f(s;\zeta),a;\psi)-\sum_{a'\in\mathcal{A}} A_i(f(s;\zeta),a';\psi)/|\mathcal{A}|}}{\sum_j e^{V_j(f(s;\zeta);\eta)+A_j(f(s;\zeta),a;\psi)-\sum_{a'\in\mathcal{A}} A_j(f(s;\zeta),a';\psi)/|\mathcal{A}|}}.$$

Rainbow Hyperparameters

Finally, we replace all linear layers by their noisy equivalents.

Parameter	Value
Min history to start learning	80K frames
Adam learning rate	0.0000625
Exploration ϵ	0.0
Noisy Nets σ_0	0.5
Target Network Period	32K frames
Adam ϵ	1.5×10^{-4}
Prioritization type	proportional
Prioritization exponent ω	0.5
Prioritization importance sampling β	$0.4 \rightarrow 1.0$
Multi-step returns n	3
Distributional atoms	51
Distributional min/max values	$[-10, 10]$

Table 1 of "Rainbow: Combining Improvements in Deep Reinforcement Learning" by Matteo Hessel et al.

Rainbow Results

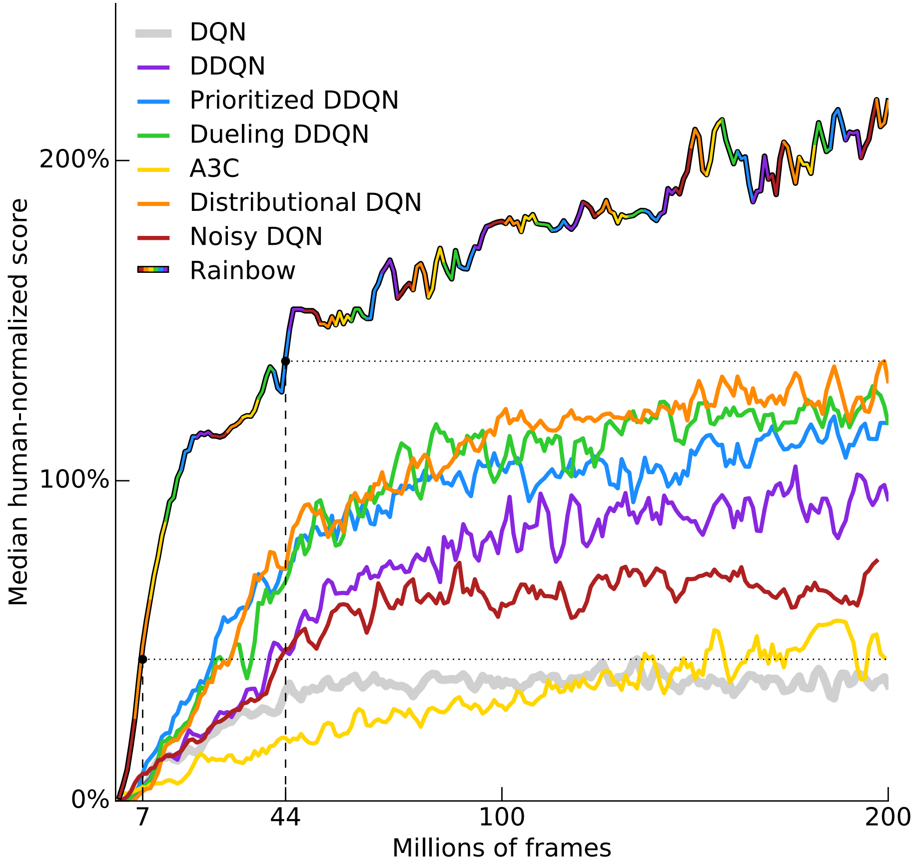

Figure 1 of "Rainbow: Combining Improvements in Deep Reinforcement Learning" by Matteo Hessel et al.

Agent	no-ops	human starts
DQN	79%	68%
DDQN (*)	117%	110%
Prioritized DDQN (*)	140%	128%
Dueling DDQN (*)	151%	117%
A3C (*)	-	116%
Noisy DQN	118%	102%
Distributional DQN	164%	125%
Rainbow	223%	153%

Table 2 of "Rainbow: Combining Improvements in Deep Reinforcement Learning" by Matteo Hessel et al.

Rainbow Results

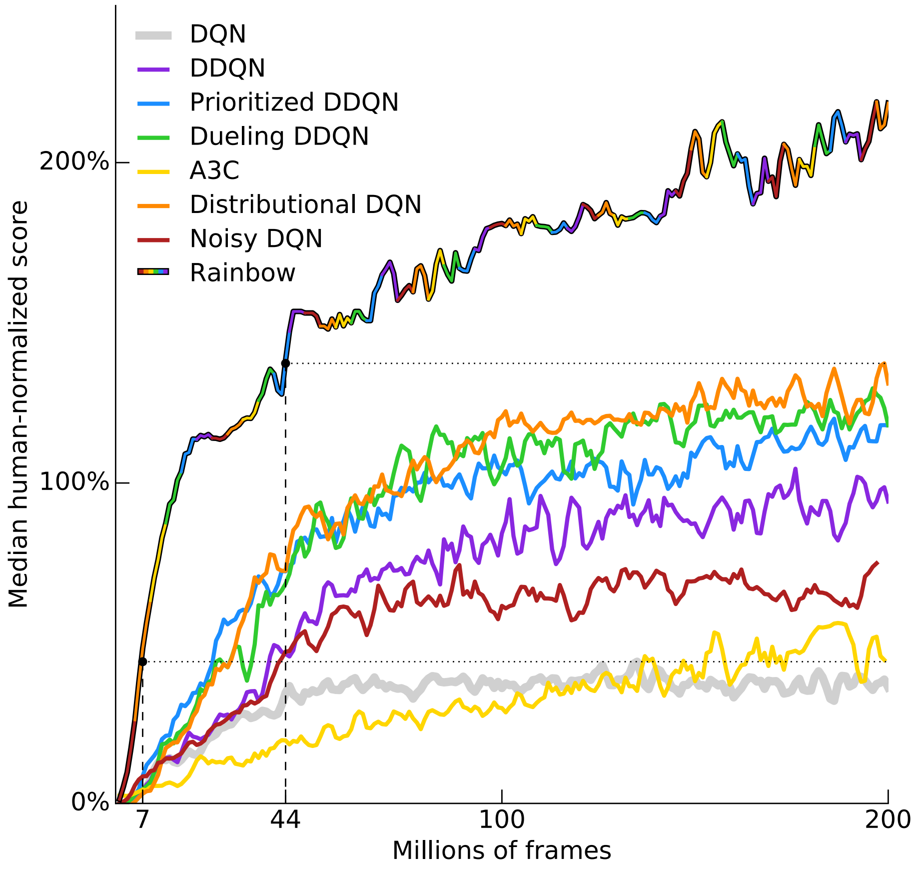

Figure 1 of "Rainbow: Combining Improvements in Deep Reinforcement Learning" by Matteo Hessel et al.

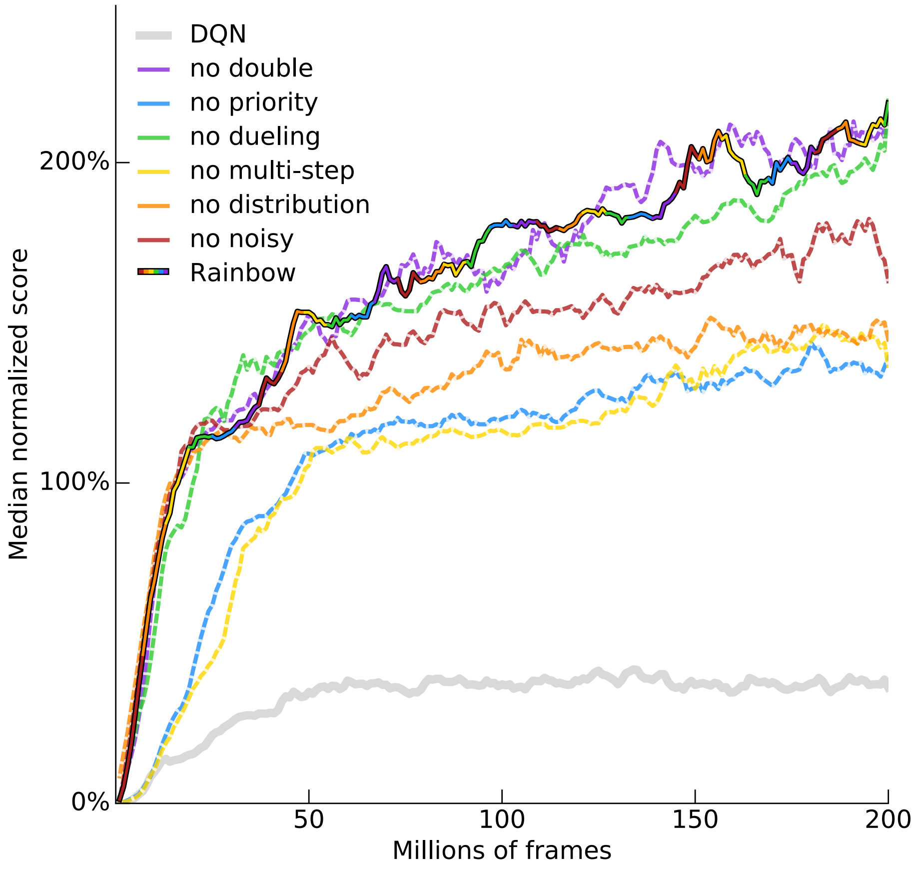

Figure 3 of "Rainbow: Combining Improvements in Deep Reinforcement Learning" by Matteo Hessel et al.

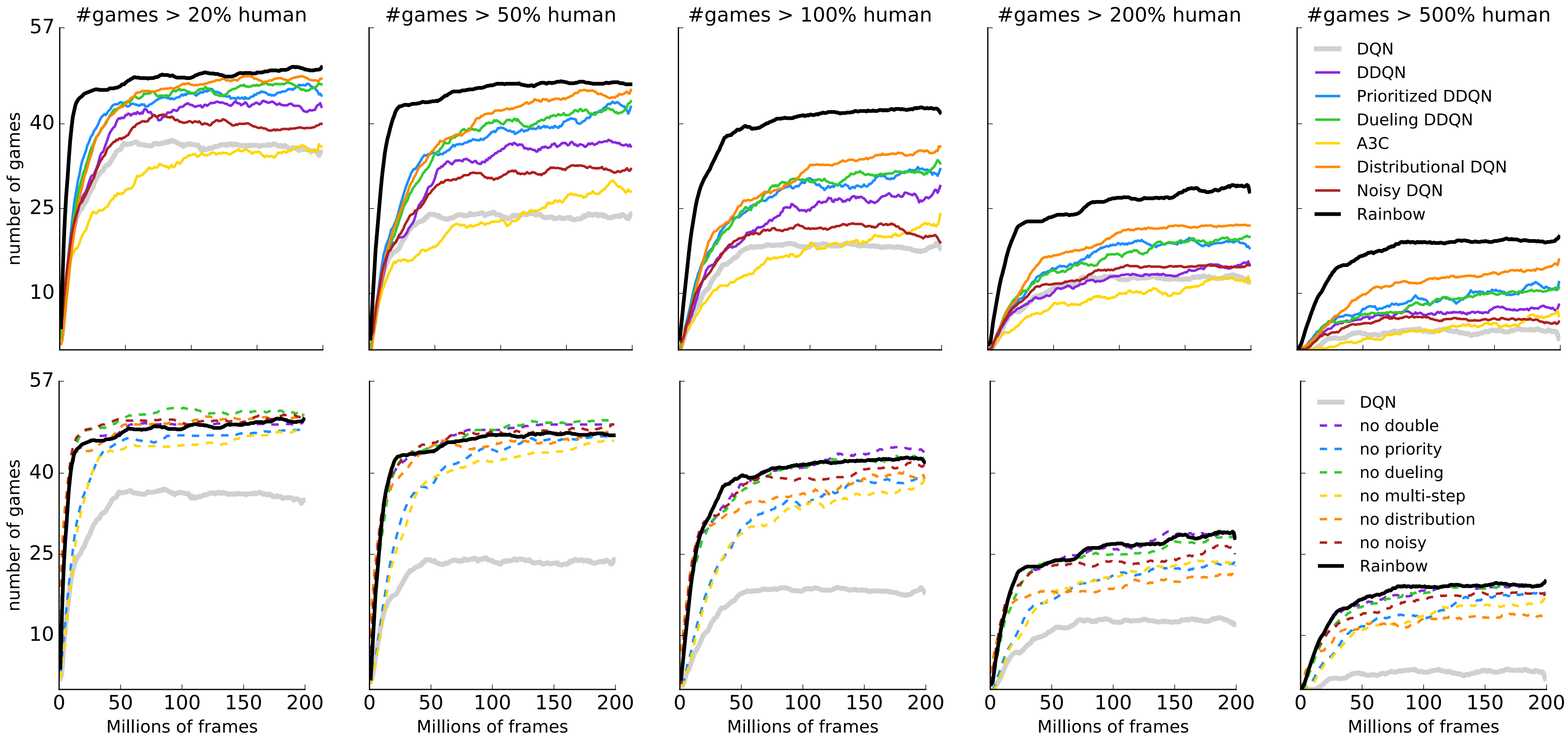

Figure 2: Each plot shows, for several agents, the number of games where they have achieved at least a given fraction of human performance, as a function of time. From left to right we consider the 20%, 50%, 100%, 200% and 500% thresholds. On the first row we compare Rainbow to the baselines. On the second row we compare Rainbow to its ablations.

Figure 2 of "Rainbow: Combining Improvements in Deep Reinforcement Learning" by Matteo Hessel et al.

Rainbow Ablations

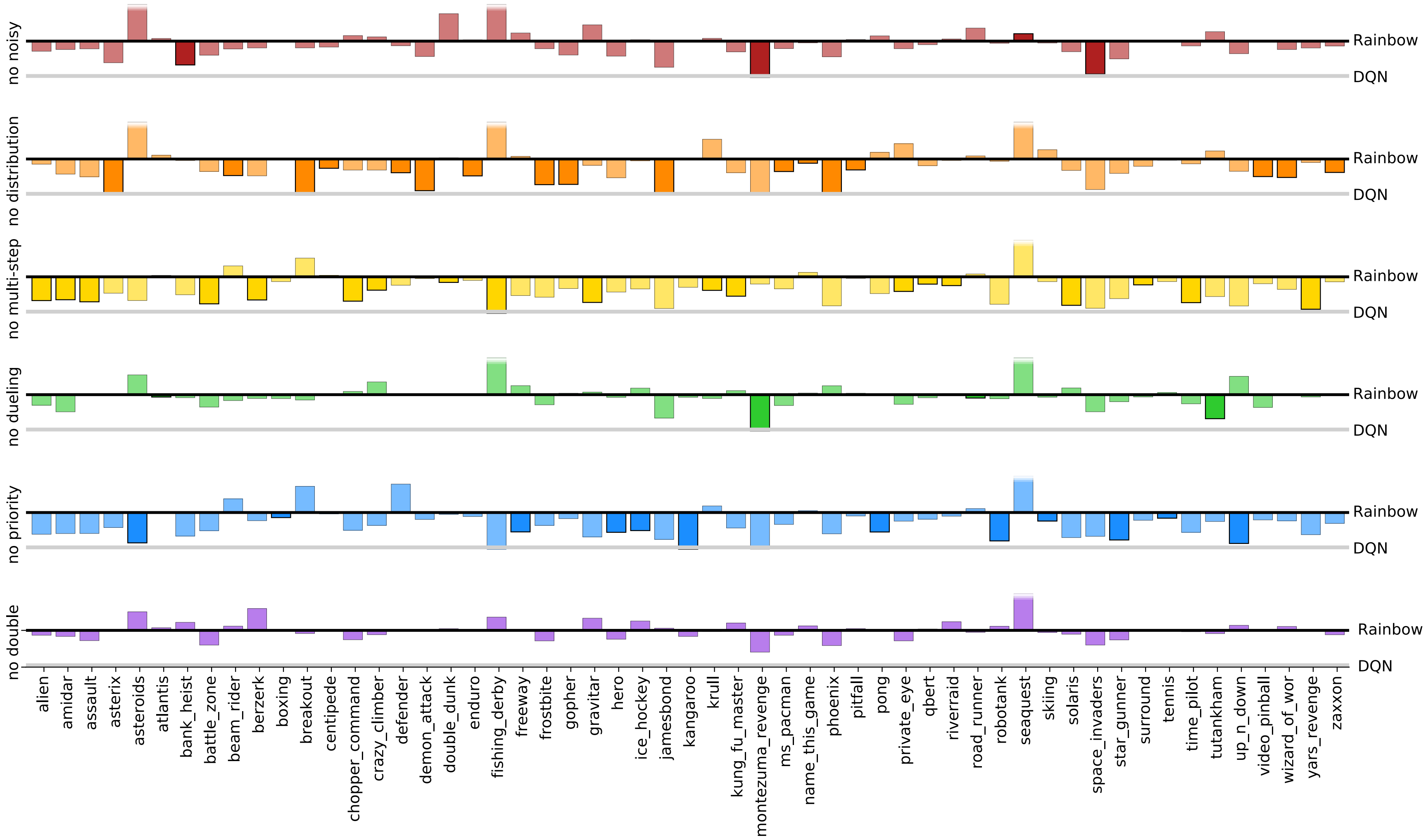

Figure 4 of "Rainbow: Combining Improvements in Deep Reinforcement Learning" by Matteo Hessel et al.