Function Approximation

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

We denote the approximations as

\[
\hat{v}(s, \mathbf{w}), \\
\hat{q}(s, a, \mathbf{w}).
\]

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

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

where the state distribution \( \mu(s) \) is usually on-policy distribution.
The functional approximation (i.e., the weight vector $w$) is usually optimized using gradient methods, for example as

$$w_{t+1} \leftarrow w_t - \frac{1}{2} \alpha \nabla \left[ v_\pi(S_t) - \hat{v}(S_t, w_t) \right]^2$$

$$\leftarrow w_t + \alpha [v_\pi(S_t) - \hat{v}(S_t, w_t)] \nabla \hat{v}(S_t, w_t).$$

As usual, the $v_\pi(S_t)$ is estimated by a suitable sample. For example in Monte Carlo methods, we use episodic return $G_t$, and in temporal difference methods, we employ bootstrapping and use $R_{t+1} + \gamma \hat{v}(S_{t+1}, w)$. 
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 as shown earlier).

Figure 1 of the paper “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 4th 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)
Deep Q Networks

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

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

- An \( \varepsilon \)-greedy behavior policy is utilized.

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;
- separate target network \( \bar{\theta} \): to prevent instabilities, a separate target network is used to estimate state-value function. The weights are not trained, but copied from the trained network once in a while;
- reward clipping of \((r + \gamma \max_{a'} Q(s', a'; \bar{\theta}) - Q(s, a; \theta))\) to \([-1, 1]\).
**Deep Q Networks Hyperparameters**

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>minibatch size</td>
<td>32</td>
</tr>
<tr>
<td>replay buffer size</td>
<td>1M</td>
</tr>
<tr>
<td>target network update frequency</td>
<td>10k</td>
</tr>
<tr>
<td>discount factor</td>
<td>0.99</td>
</tr>
<tr>
<td>training frames</td>
<td>50M</td>
</tr>
<tr>
<td>RMSProp learning rate and momentum</td>
<td>0.00025, 0.95</td>
</tr>
<tr>
<td>initial $\varepsilon$, final $\varepsilon$ and frame of final $\varepsilon$</td>
<td>1.0, 0.1, 1M</td>
</tr>
<tr>
<td>replay start size</td>
<td>50k</td>
</tr>
<tr>
<td>no-op max</td>
<td>30</td>
</tr>
</tbody>
</table>
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 7 of them into a single architecture they call *Rainbow*.

---

*Figure 1 of the paper "Rainbow: Combining Improvements in Deep Reinforcement Learning" by Matteo Hessel et al.*
Rainbow DQN Extensions

Double Q-learning

Similarly to double Q-learning, instead of

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

we minimize

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

Figure 1: The orange bars show the bias in a single Q-learning update when the action values are \( Q(s, a) = V_*^\epsilon (s) + \epsilon_a \) 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 the paper “Deep Reinforcement Learning with Double Q-learning” by Hado van Hasselt et al.
Rainbow DQN Extensions

**Double Q-learning**

- **True value and an estimate**
  - $Q^*(s, a)$
  - $Q_t(s, a)$

- **All estimates and max**
  - $\max_a Q_t(s, a)$

- **Bias as function of state**
  - $\max_a Q_t(s, a) - \max_a Q^*(s, a)$
  - $Q_t(s, a)$

**Figure 2** of the paper “Deep Reinforcement Learning with Double Q-learning” by Hado van Hasselt et al.
Rainbow DQN Extensions

Double Q-learning

Figure 3 of the paper “Deep Reinforcement Learning with Double Q-learning” by Hado van Hasselt et al.
# Rainbow DQN Extensions

## Double Q-learning

<table>
<thead>
<tr>
<th></th>
<th>DQN</th>
<th>Double DQN</th>
<th>Double DQN (tuned)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td>93.5%</td>
<td>114.7%</td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>241.1%</td>
<td>330.3%</td>
<td></td>
</tr>
</tbody>
</table>

*Table 1 of the paper “Deep Reinforcement Learning with Double Q-learning” by Hado van Hasselt et al.*

<table>
<thead>
<tr>
<th></th>
<th>DQN</th>
<th>Double DQN</th>
<th>Double DQN (tuned)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td>47.5%</td>
<td>88.4%</td>
<td>116.7%</td>
</tr>
<tr>
<td>Mean</td>
<td>122.0%</td>
<td>273.1%</td>
<td>475.2%</td>
</tr>
</tbody>
</table>

*Table 2 of the paper “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

\[ p_t \propto \left| r + \gamma \max_{a'} Q(s', a'; \bar{\theta}) - Q(s, a; \theta) \right|^{\omega}, \]

where \( \omega \) 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.
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

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

Algorithm 1 of the paper “Prioritized Experience Replay” by Tom Schaul et al.
Duelling Networks

Instead of computing directly $Q(s, a; \theta)$, 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 $s$.

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

*Figure 1 of the paper “Dueling Network Architectures for Deep Reinforcement Learning” by Ziyu Wang et al.*
Duelling 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 the paper “Dueling Network Architectures for Deep Reinforcement Learning” by Ziyu Wang et al.
Rainbow DQN Extensions

Duelling Networks

Figure 2 of the paper “Dueling Network Architectures for Deep Reinforcement Learning” by Ziyu Wang et al.
### Rainbow DQN Extensions

## Duelling Networks

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

*Table 1 of the paper “Dueling Network Architectures for Deep Reinforcement Learning” by Ziyu Wang et al.*
Multi-step Learning

Instead of Q-learning, we use $n$-step variant Q-learning (to be exact, we use $n$-step Expected Sarsa) to maximize

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

This changes the off-policy algorithm to on-policy, but 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 $\theta$ are represented as

$$
\theta \overset{\text{def}}{=} \mu + \sigma \odot \varepsilon,
$$

where $\varepsilon$ is zero-mean noise with fixed statistics. We therefore learn the parameters $\zeta \overset{\text{def}}{=} (\mu, \sigma)$.

Therefore, a fully connected layer

$$
y = wx + b
$$

is represented in the following way in Noisy Nets:

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

The noise $\varepsilon$ 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 $\varepsilon_i$ for input neurons, independent noise $\varepsilon_j$ for output neurons, and set

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

for $f(x) = \text{sign}(x) \sqrt{|x|}$. The authors generate noise samples for every batch, sharing the noise for all batch instances.

**Deep Q Networks**

When training a DQN, $\varepsilon$-greedy is no longer used and all policies are greedy, and all fully connected layers are parametrized as noisy nets.
Rainbow DQN Extensions

Noisy Nets

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th></th>
<th>NoisyNet</th>
<th></th>
<th>Improvement (On median)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Median</td>
<td>Mean</td>
<td>Median</td>
<td></td>
</tr>
<tr>
<td>DQN</td>
<td>319</td>
<td>83</td>
<td>379</td>
<td>123</td>
<td>48%</td>
</tr>
<tr>
<td>Dueling</td>
<td>524</td>
<td>132</td>
<td>633</td>
<td>172</td>
<td>30%</td>
</tr>
<tr>
<td>A3C</td>
<td>293</td>
<td>80</td>
<td>347</td>
<td>94</td>
<td>18%</td>
</tr>
</tbody>
</table>

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

Figure 2 of the paper "Noisy Networks for Exploration" by Meire Fortunato et al.
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.
Distributional RL

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

These distributions satisfy a distributional Bellman equation:

$$Z(s, a) = R(s, a) + \gamma Z(s', a').$$

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 $V_{\text{MIN}}, V_{\text{MAX}} \in \mathbb{R}$. Support of the distribution are atoms

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

The atom probabilities are predicted using a softmax distribution as

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

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

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

\[
D_{KL}(\Phi(R + \max_a Z(s', a'))||Z(s, a)).
\]
Distributional RL

Algorithm 1 Categorical Algorithm

\begin{algorithm}
\textbf{input} A transition $x_t, a_t, r_t, x_{t+1}, \gamma_t \in [0, 1]$
\[ Q(x_{t+1}, a) := \sum_i z_i p_i(x_{t+1}, a) \]
\[ a^* \leftarrow \text{arg max}_a Q(x_{t+1}, a) \]
\[ m_i = 0, \quad i \in 0, \ldots, N - 1 \]
\textbf{for} $j \in 0, \ldots, N - 1$ \textbf{do}
\hspace{1em} # Compute the projection of $\hat{T}_z$ onto the support \{z_i\}
\hspace{1em} \hat{T}_z_j \leftarrow [r_t + \gamma_t z_j]_{V_{\text{MIN}}}^{V_{\text{MAX}}}
\hspace{1em} b_j \leftarrow (\hat{T}_z_j - V_{\text{MIN}})/\Delta z \quad # b_j \in [0, N - 1]
\hspace{1em} l \leftarrow [b_j], u \leftarrow [b_j]
\hspace{1em} # Distribute probability of $\hat{T}_z$
\hspace{1em} \[ m_l \leftarrow m_l + p_j(x_{t+1}, a^*)(u - b_j) \]
\hspace{1em} \[ m_u \leftarrow m_u + p_j(x_{t+1}, a^*)(b_j - l) \]
\textbf{end for}
\textbf{output} $- \sum_i m_i \log p_i(x_t, a_t)$ \quad # Cross-entropy loss
\end{algorithm}

*Algorithm 1 of the paper "A Distributional Perspective on Reinforcement Learning" by Marc G. Bellemare et al.*
Rainbow DQN Extensions

Distributional RL

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>Median</th>
<th>&gt; H.B.</th>
<th>&gt; DQN</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQN</td>
<td>228%</td>
<td>79%</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>DDQN</td>
<td>307%</td>
<td>118%</td>
<td>33</td>
<td>43</td>
</tr>
<tr>
<td>DUEL.</td>
<td>373%</td>
<td>151%</td>
<td>37</td>
<td>50</td>
</tr>
<tr>
<td>PRIOR.</td>
<td>434%</td>
<td>124%</td>
<td>39</td>
<td>48</td>
</tr>
<tr>
<td>Pr. DUEL.</td>
<td>592%</td>
<td>172%</td>
<td>39</td>
<td>44</td>
</tr>
<tr>
<td>C51</td>
<td>701%</td>
<td>178%</td>
<td>40</td>
<td>50</td>
</tr>
</tbody>
</table>

Figure 6 of the paper “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 the paper “A Distributional Perspective on Reinforcement Learning” by Marc G. Bellemare et al.
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 the paper "A Distributional Perspective on Reinforcement Learning" by Marc G. Bellemare et al.
Rainbow DQN Extensions

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 the paper "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:

\[
D_{KL}(\Phi(G_{t:t+n} + \gamma^n \max_{a'} Z(s', a')) || Z(s, a)).
\]

The prioritized replay chooses transitions according to the probability

\[
p_t \propto \left( D_{KL}(\Phi(G_{t:t+n} + \gamma^n \max_{a'} Z(s', a')) || Z(s, a)) \right)\omega.
\]

Network utilizes duelling 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

\[
p_i(s, a) \overset{\text{def}}{=} \frac{e^{V(f(s; \zeta); \eta) + A_i(f(s; \zeta), a; \psi) - \sum_{a' \in A} A_i(f(s; \zeta), a'; \psi) / |A|}}{\sum_j e^{V(f(s; \zeta); \eta) + A_j(f(s; \zeta), a; \psi) - \sum_{a' \in A} A_j(f(s; \zeta), a'; \psi) / |A|}}.
\]
### Rainbow Hyperparameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min history to start learning</td>
<td>80K frames</td>
</tr>
<tr>
<td>Adam learning rate</td>
<td>0.0000625</td>
</tr>
<tr>
<td>Exploration $\epsilon$</td>
<td>0.0</td>
</tr>
<tr>
<td>Noisy Nets $\sigma_0$</td>
<td>0.5</td>
</tr>
<tr>
<td>Target Network Period</td>
<td>32K frames</td>
</tr>
<tr>
<td>Adam $\epsilon$</td>
<td>$1.5 \times 10^{-4}$</td>
</tr>
<tr>
<td>Prioritization type</td>
<td>proportional</td>
</tr>
<tr>
<td>Prioritization exponent $\omega$</td>
<td>0.5</td>
</tr>
<tr>
<td>Prioritization importance sampling $\beta$</td>
<td>$0.4 \rightarrow 1.0$</td>
</tr>
<tr>
<td>Multi-step returns $n$</td>
<td>3</td>
</tr>
<tr>
<td>Distributional atoms</td>
<td>51</td>
</tr>
<tr>
<td>Distributional min/max values</td>
<td>$[-10, 10]$</td>
</tr>
</tbody>
</table>

*Table 1 of the paper “Rainbow: Combining Improvements in Deep Reinforcement Learning” by Matteo Hessel et al.*
Rainbow Results

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

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

Figure 1 of the paper “Rainbow: Combining Improvements in Deep Reinforcement Learning” by Matteo Hessel et al.

Figure 3 of the paper “Rainbow: Combining Improvements in Deep Reinforcement Learning” by Matteo Hessel et al.
Rainbow Ablations

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 the paper “Rainbow: Combining Improvements in Deep Reinforcement Learning” by Matteo Hessel et al.
Rainbow Ablations

Figure 4 of the paper “Rainbow: Combining Improvements in Deep Reinforcement Learning” by Matteo Hessel et al.