# APPENDICES: Revisiting Rainbow

## A. Environments

For OpenAI environments, data is summarized from https://github.com/openai/gym and information provided on the wiki https://github.com/openai/gym/wiki.

*Figure 8.* The classic control environments. From left to right: CartPole, Acrobot, LunarLander, and MountainCar.

### A.1. CartPole-v0

CartPole is a task of balancing a pole on top of the cart. The cart has access to its position and velocity as state, and can only go left or right for each action. The task is over when the pole falls over (less than $\pm 12$ deg), the cart goes out of the boundaries ($\pm 2.4$ units off the center), or 200 time steps are reached, with each step returning 1 reward. The agent is given a continuous 4-dimensional space describing the environment, and can respond by returning one of two values, pushing the cart either right or left.

### A.2. Acrobot-v1

In the Acrobot environment, the agent is given rewards for swinging a double-jointed pendulum up from a stationary position. The agent can actuate the second joint by returning one of three actions, corresponding to left, right, or no torque. The agent is given a six dimensional vector describing the environment's angles and velocities. The episode ends when the end of the second pole is more than the length of a pole above the base. For each timestep that the agent does not reach this state, it is given a -1 reward. The episode length is 500 timesteps.

### A.3. LunarLander-v2

In the LunarLander environment, the agent attempts to land a lander on a particular location on a simulated 2D world. If the lander hits the ground going too fast, the lander will explode, or if the lander runs out of fuel, the lander will plummet toward the surface. The agent is given a continuous vector describing the state, and can turn its engine on or off. The landing pad is placed in the center of the screen, and if the lander lands on the pad, it is given a reward (100-140 points). The agent also receives a variable amount of reward when coming to rest, or contacting the ground with a leg (10 points). The agent loses a small amount of reward by firing the engine (-0.3 points), and loses a large amount of reward if it crashes (-100 points). The observation consists of the x and y coordinates, the x and y velocities, angle, angular velocity, and ground contact information of the lander (left and right leg) and the action consists of do nothing, fire left orientation engine, fire down engine and fire right orientation engine.

### A.4. MountainCar-v0

MountainCar is a one dimensional track between two mountains. The goal is to drive up the mountain to the right. The agent receives a -1 reward for every time step it does not reach the top. The episode terminates when it reaches 0.5 position, or if 200 iterations are reached. The objective is for the agent to learn to drive back and forth to build momentum that will be enough to push the car up the hill. The observation consists of the car's position and velocity and the action consists of pushing left, pushing right and no push.

### A.5. MinAtar Environments

Some of the best reinforcement learning algorithms require tens or hundreds of millions of timesteps to learn to play Atari games, the equivalent of several weeks of training in real time. MinAtar reduces the complexity of the representation learning problem for 5 Atari games, while maintaining the mechanics of the original games as much as possible. We use these MinAtar games to measure the impact of using some extensions to the DQN algorithm. MinAtar provides analogues to five Atari games which play out on a 10x10 grid. The environments provide a 10x10xn state representation, where each of the n channels correspond to a game-specific object, such as ball, paddle and brick in the game Breakout. Detailed descriptions of each game are available in https://github.com/kenjyoung/MinAtar.

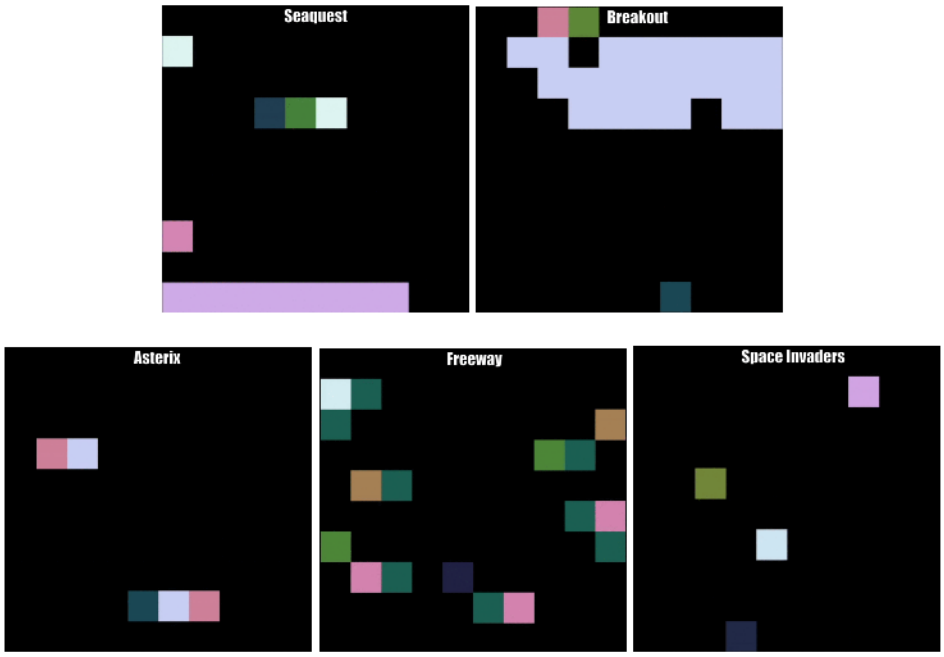


*Figure 9.* Visualization of each MinAtar environment.

## B. Detailed description of the revisting rainbow components

In this section we present the enhancements to DQN that were combined for the revisting rainbow agent.

### B.1. Double Q-learning

van Hasselt et al. (2016) added double Q-learning (Hasselt, 2010) to mitigate overestimation bias in the $Q$-estimates by decoupling the maximization of the action from its selection in the target bootstrap. The loss from Equation 2 is replaced with

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma Q_{\bar{\theta}}(s', \arg\max_{a'\in\mathcal{A}} Q_\theta(s', a')) - Q_\theta(s,a)\right)^2\right]$$

### B.2. Prioritized experience replay

Instead of sampling uniformly from the replay buffer ($U(D)$), prioritized experience replay (Schaul et al., 2016) proposed to sample a trajectory $t = (s, a, r, s')$ with probability $p_t$ proportional to the temporal difference error:

$$p_t \propto \left|r + \gamma \max_{a'\in\mathcal{A}} Q_{\bar{\theta}}(s', a') - Q_\theta(s,a)\right|^\omega$$

where $\omega$ is a hyper-parameter for the sampling distribution.

### B.3. Dueling networks

Wang et al. (2016) introduced dueling networks by modifying the DQN network architecture. Specifically, two streams share the initial convolutional layers, separately estimating $V^*(s)$, and the advantages for each action: $A(s,a) := Q^*(s,a) - V^*(s)$. The output of the full network is defined by:

$$Q_\theta(s,a) = V_\eta(f_\eta(s)) + A_\psi(f_\eta(s), a) - \frac{\sum_{a'\in\mathcal{A}} A_\psi(f_\xi(s), a')}{|\mathcal{A}|}$$

Where $\xi$ denotes the parameters for the shared convolutional layers, $\eta$ denotes the parameters for the value estimator stream, $\psi$ denots the parameters for the advantage estimator stream, and $\theta := \xi \cup \eta \cup \psi$.

### B.4. Multi-step learning

Instead of computing the temporal difference error using a single-step transition, one can use multi-step targets instead (Sutton, 1988), where for a trajectory $(s_0, a_0, r_0, s_1, a_1, \cdots)$ and update horizon $n$:

$$R_t^{(n)} := \sum_{k=0}^{n-1} \gamma^k r_{t+k+1}$$

yielding the multi-step temporal difference:

$$R_t^{(n)} + \gamma^n \max_{a'\in\mathcal{A}} Q_{\bar{\theta}}(s_{t+n}, a') - Q_\theta(s_t, a_t)$$

### B.5. Distributional RL

Bellemare et al. (2017) demonstrated that the Bellman recurrence also holds for *value distributions*:

$$Z(x,a) \overset{D}{=} R(s,a) + \gamma Z(X', A')$$

where $Z$, $R$, and $(X', A')$ are random variables representing the return, immediate reward, and next state-action, respectively. The authors present an algorithm (C51) to maintain an estimate $Z_\theta$ of the return distribution $Z$ by use of a parameterized categorical distribution with 51 atoms.

### B.6. Quantile Regression for Distributional RL

Dabney et al. (2017) computes the return quantile values for $N$ fixed, uniform probabilities. The distribution of the random return is approximated by a uniform mixture of $N$ Diracs with each $\theta_i$ assigned a quantile value trained with quantile regression. Formally, a quantile distribution $Z_\theta \in \mathcal{Z}_Q$ maps each state-action pair $(x, a)$ to a uniform probability distribution is,

$$Z_\theta(x, a) := \frac{1}{N} \sum_{i=1}^{N} \delta_{\theta_i(x,a)}$$

These quantile estimates are trained using the Huber (Huber, 1964) quantile regression loss.

### B.7. Implicit quantile networks

Dabney et al. (2018b) extend the approach of Dabney et al.(2018), from learning a discrete set of quantiles to learningthe full quantile function, a continuous map from probabilities to returns. Moreover, extended the fixed quantile fractions to uniform samples. The approximation of the quantile function $Z$ is,

$$Z_\tau(x, a) \approx f(\psi(x) \odot \phi(\tau))_a$$

where $\odot$ denotes the element-wise (Hadamard) product.

### B.8. Noisy nets

Fortunato et al. (2018) propose replacing the simple $\epsilon$-greedy exploration strategy used by DQN with noisy linear layers that include a noisy stream. Specifically, the standard linear layers defined by $\mathbf{y} = \mathbf{b} + \mathbf{W}\mathbf{x}$ are replaced by:

$$\mathbf{y} = (\mathbf{b} + \mathbf{W}\mathbf{x}) + (\mathbf{b}_{\mathbf{noisy}} \odot \epsilon^b + (\mathbf{W}_{\mathbf{noisy}} \odot \epsilon^w)\mathbf{x})$$

where $\epsilon^b$ and $\epsilon^w$ are noise variables (Hessel et al. (2018) use factorised Gaussian noise) and $\odot$ is the Hadamard product.

### B.9. Munchausen Reinforcement Learning

Vieillard et al. (2020) modify the regression target adding the scaled log-policy to the immediate reward:

$$Y^{M-DQN} = r_t + \alpha\tau \ln \pi_{\bar{\theta}}(a_t \mid s_t) + \gamma \sum_{a' \in \mathcal{A}} \pi_{\bar{\theta}}(a' \mid s_{t+1}) \left(Q_{\bar{\theta}}(s_{t+1}, a') - \tau \ln \pi_{\bar{\theta}}(a' \mid s_{t+1})\right)$$

with $\pi_\theta = \mathrm{sm}\left(\frac{Q_{\bar{\theta}}}{\tau}\right)$ and a scaling factor $\alpha \in [0, 1]$.

### B.10. Loss functions

We ran experiments using Mean Square Error and Huber loss (Huber, 1964), therefore we consider it important to introduce these concepts in this paper. MSE is the sum of squared distances between the target variable and predicted values. MSE gives relatively higher weight (penalty) to large errors/outliers, while smoothening the gradient for smaller errors. The MSE is formally defined by the following equation,

$$\mathrm{MSE}(y_i, \hat{y}_i) = \frac{1}{2}(y_i - \hat{y}_i)^2$$

On the other hand, Huber loss is less sensitive to outliers in data than the squared error loss and it is defined by the following equation,

$$\mathrm{Huber}(y_i, \hat{y}_i) = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{for } |y_i - \hat{y}_i| \leq \delta, \\ \delta |y_i - \hat{y}_i| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases}$$

## C. Hyperparameters settings for classic control environments

*Table 1.* Hyperparameters settings for classic control environments

| Hyperparameter | DQN | Rainbow | QR-DQN | IQN | M-DQN | M-IQN |
|---|---|---|---|---|---|---|
| gamma | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 |
| update horizon | 1 | 3 | 1 | 1 | 1 | 1 |
| min replay history | 500 | 500 | 500 | 500 | 500 | 500 |
| update period | 4 | 2 | 2 | 2 | 4 | 2 |
| target update period | 100 | 100 | 100 | 100 | 100 | 100 |
| normalize obs | True | True | True | True | True | True |
| hidden layer | 2 | 2 | 2 | 2 | 2 | 2 |
| neurons | 512 | 512 | 512 | 512 | 512 | 512 |
| num atoms | - | 51 | 51 | - | - | - |
| vmax | - | 200 | - | - | - | - |
| kappa | - | - | 1 | 1 | - | 1 |
| tau | - | - | - | - | 100 | 0.03 |
| alpha | - | - | - | - | 1 | 1 |
| clip value min | - | - | - | - | -1e3 | -1 |
| num tau samples | - | - | - | 32 | - | 32 |
| num tau prime samples | - | - | - | 32 | - | 32 |
| num quantile samples | - | - | - | 32 | - | 32 |
| quantile embedding dim | - | - | - | 64 | - | 64 |
| learning rate | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| eps | 3.125e-4 | 3.125e-4 | 3.125e-4 | 3.125e-4 | 3.125e-4 | 3.125e-4 |
| num iterations | 30 | 30 | 30 | 30 | 30 | 30 |
| replay capacity | 50000 | 50000 | 50000 | 50000 | 50000 | 50000 |
| batch size | 128 | 128 | 128 | 128 | 128 | 128 |

## D. Hyperparameters settings for MinAtar environments

*Table 2.* Hyperparameters settings for MinAtar environments

| Hyperparameter | DQN | Rainbow | QR-DQN | IQN | M-DQN | M-IQN |
|---|---|---|---|---|---|---|
| gamma | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 |
| update horizon | 1 | 3 | 1 | 1 | 1 | 1 |
| min replay history | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 |
| update period | 4 | 4 | 4 | 4 | 4 | 4 |
| target update period | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 |
| normalize obs | True | True | True | True | True | True |
| num atoms | - | 51 | 51 | - | - | - |
| vmax | - | 100 | - | - | - | - |
| kappa | - | - | 1 | 1 | - | 1 |
| tau | - | - | - | - | 0.03 | 0.03 |
| alpha | - | - | - | - | 0.9 | 0.9 |
| clip value min | - | - | - | - | - 1 | -1 |
| num tau samples | - | - | - | 32 | - | 32 |
| num tau prime samples | - | - | - | 32 | - | 32 |
| num quantile samples | - | - | - | 32 | - | 32 |
| quantile embedding dim | - | - | - | 64 | - | 64 |
| learning rate | 0.00025 | 0.00025 | 0.00025 | 0.00025 | 0.00025 | 0.00025 |
| eps | 3.125e-4 | 3.125e-4 | 3.125e-4 | 3.125e-4 | 3.125e-4 | 3.125e-4 |
| num iterations | 10 | 10 | 10 | 10 | 10 | 10 |
| replay capacity | 100000 | 100000 | 100000 | 100000 | 100000 | 100000 |
| batch size | 32 | 32 | 32 | 32 | 32 | 32 |

## E. Average runtime for environments used in the experiments

*Table 3.* Average runtime.

| Figures | Average time (minutes) |
|---|---|
| Cartpole | 10 |
| Acrobot | 10 |
| LunarLander | 30 |
| MountainCar | 10 |
| MinAtar games | 720 |
| Atari games | 7200 |

## F. Impact of algorithmic components

In the following tables we list the algorithmic components that had the most positive effect on the various agents and environments. The following abbreviations are used in this section: Dist: Distributional, Noi: Noisy, Prio: Prioritized, Mult: Multi-step, Dou: Double , Due: Dueling.

*Table 4.* Impact of algorithmic components for each agent on classic control environments.

| Agent | Operation | Cartpole | Acrobot | LunarLander | MountainCar |
|---|---|---|---|---|---|
| DQN | + | Dist/Noi | Prio/ Mult | Dist/ Mult | Mult/Dou |
| Rainbow | - | Mult/Noi | Mult/Noi | Dist/Mult | Mult/Noi |
| QR-DQN | + | Noi/Prio | Mult | Mult | Noi/Mult |
| IQN | + | Prio /Due | Mult/Noi | Mult/Dou | Noi |
| M-DQN | + | Mult | Mult /Prio | Due/Noi | Mult |
| M-IQN | + | Prio | Mult | Mult /Due | - |

*Table 5.* Impact of algorithmic components for each agent on MinAtar games.

| Agent | Operation | Asterix | Breakout | Freeway | Seaquest | SpaceInvaders |
|---|---|---|---|---|---|---|
| DQN | + | Dist/Due | Mult/Dou | Due/Noi | Mult/Noi | Dist/ Mult |
| Rainbow | - | Dist/Mult | Mult/ Dist | Dist/Due | Mult/Due | Dist/Mult |
| QR-DQN | + | Prio/ Multi-step | Mult/ Prio | Noi | Mult/Noi | Prio/Due |
| IQN | + | Prio/Noi | Mult/Noi | Due | Mult/Due | Prio/Mult |
| M-DQN | + | Prio/Mult | Prio/Mult | - | Mult | Prio/Due |
| M-IQN | + | Mult/Prio | Prio/ Mult | - | Prio/Noi | Prio/ Mult |

*Table 6.* The most important component for classic control environments and MinAtar games.

| Agent | Operation | Classic control | MinAtar |
|---|---|---|---|
| DQN | + | Mult | Mult |
| Rainbow | - | Mult | Dist |
| QR-DQN | + | Mult | Prio |
| IQN | + | Mult | Mult |
| M-DQN | + | Mult | Prio |
| M-IQN | + | Mult | Prio |

*Table 7.* The most important component for each environment.

| Environment | Component |
|---|---|
| Cartpole | Prio |
| Acrobot | Mult |
| LunarLander | Mult |
| MountainCar | Noi |
| Asterix | Prio |
| Breakout | Mult |
| Freeway | Due |
| Seaquest | Mult |
| SpaceInvaders | Mult |

## G. Examining network architectures and batch sizes

In this section we evaluate the effect of varying the number of layers and number of hidden units with DQN (Figure 10) and Rainbow (Figure 11), as well as the effect of batch sizes on both (Figure 12). For all the results in this section we ran 10 independent runs for each, and the shaded areas represent 95% confidence intervals.

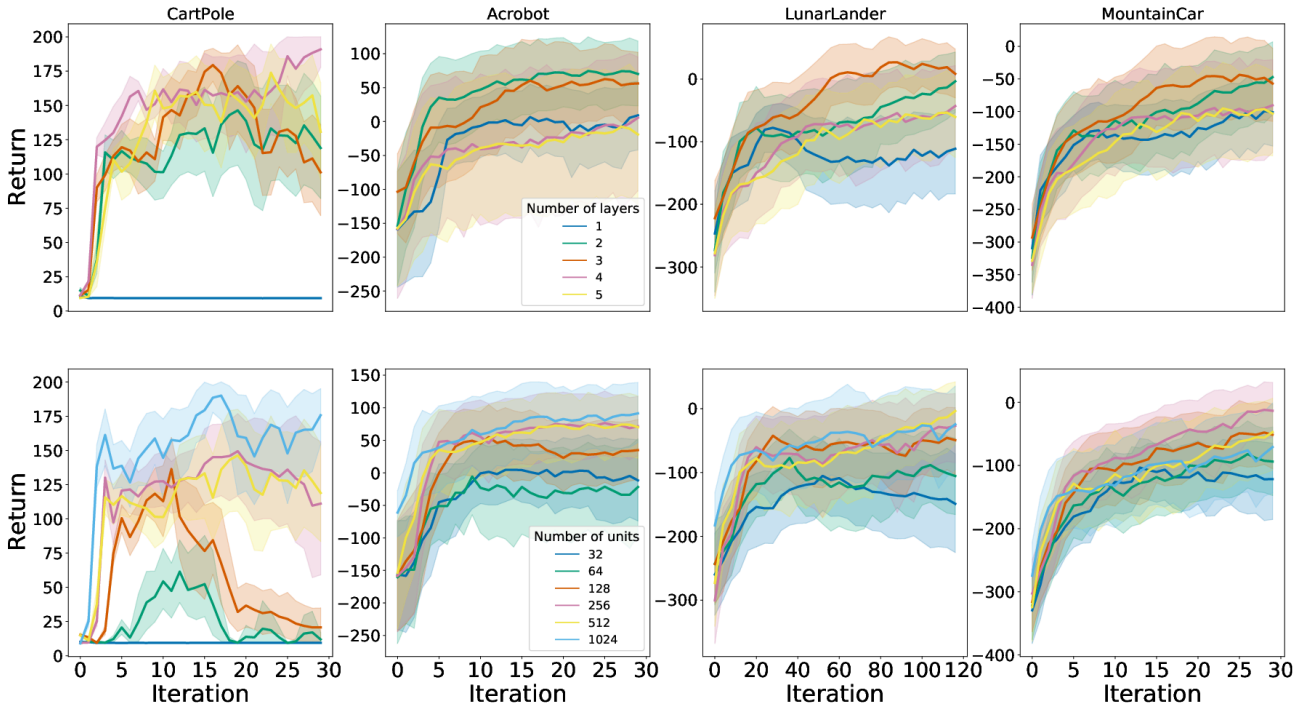


*Figure 10.* Evaluating DQN sensitivity to varying number of layers (top) and units per layer (bottom).

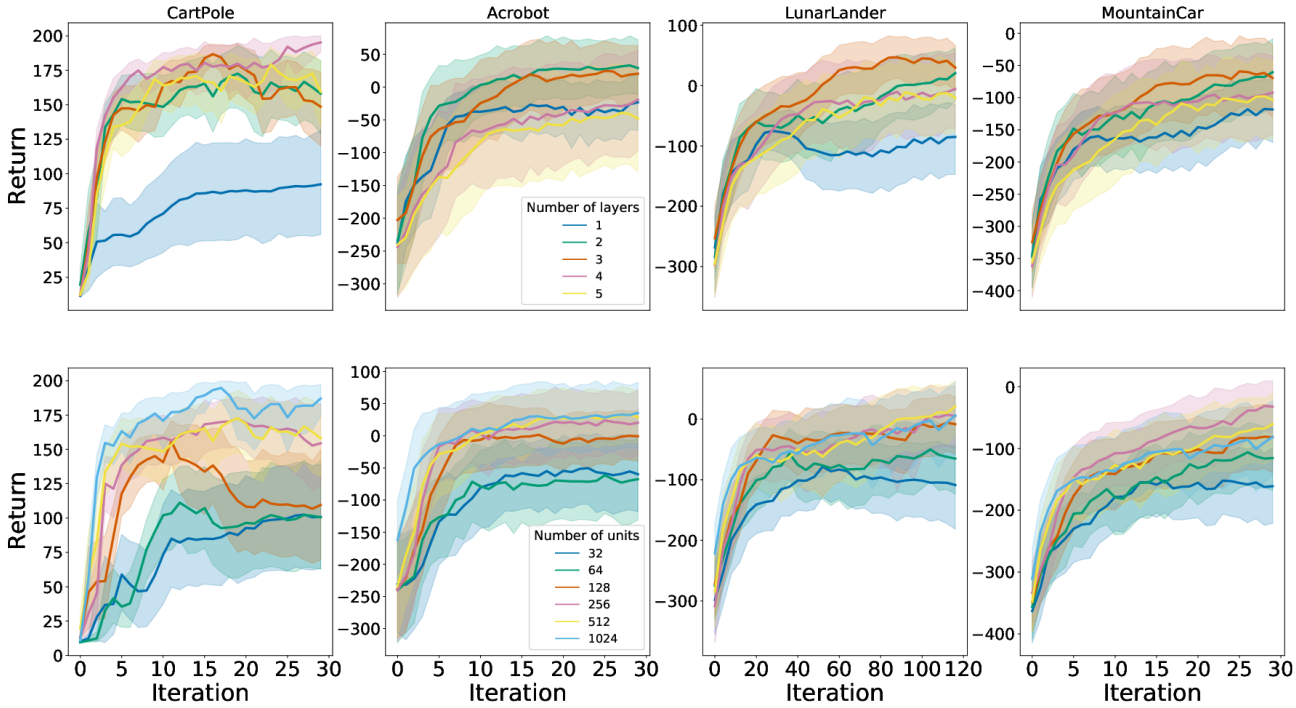


*Figure 11.* Evaluating Rainbow sensitivity to varying number of layers (top) and units per layer (bottom).

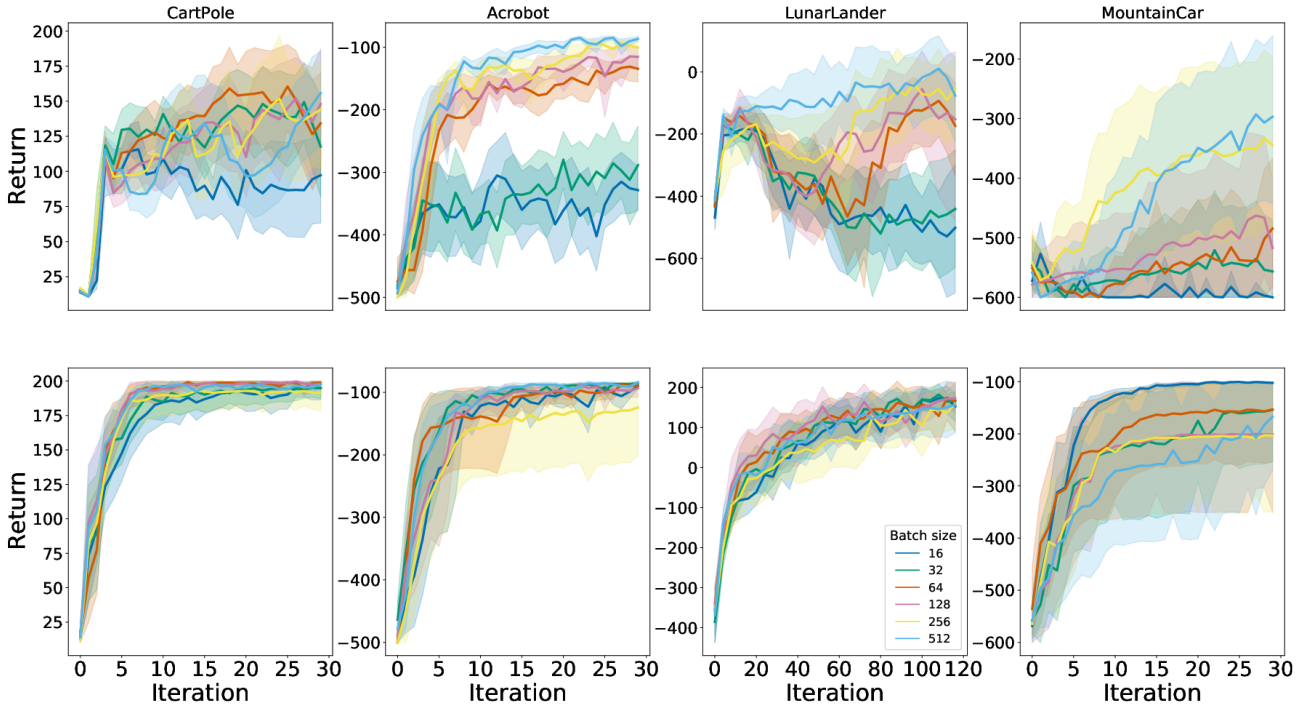


*Figure 12.* Evaluating DQN (top) and Rainbow (bottom) sensitivity to varying batch sizes. The default value used in the rest of the experiments is 128.

## H. Reevaluating the Huber loss, complete results

In this section we present complete results comparing the various combinations possible when using either Adam or RMSProp optimizers, and Huber or MSE losses.

### H.1. Classic control and MinAtar results

We first evaluated the various combinations on all the classic control and MinAtar environments and found that Adam+MSE worked best. Full results displayed in Figure 13.

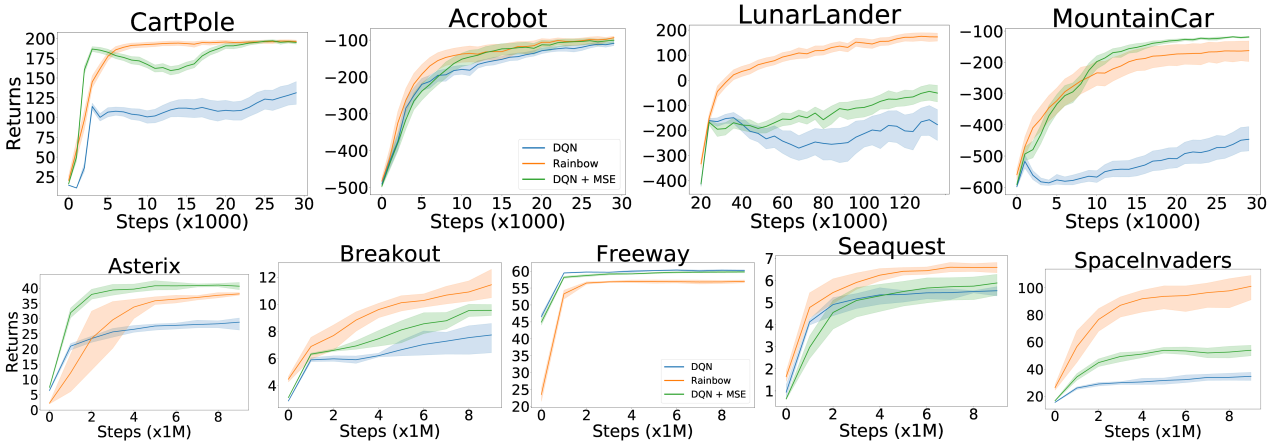


*Figure 13.* Evaluation of the use of the mean-squared error loss, instead of the Huber loss, in DQN.

### H.2. Atari results

Given that the standard practice when training DQN is to use RMSPRop with the Huber loss, yet our results on the classic control and MinAtar environments suggest that MSE would work better, we performed a thorough comparison of the various combinations, which we present in this section.

We compare Adam vs RMSProp when both use the Huber loss (Figure 14) and MSE loss (Figure 15); we also compare the MSE vs Huber loss when using RMSProp (Figure 16) and using Adam (Figure 17). In Figure 18 we compare the various combinations using human normalized scores across all games, and we provide complete training curves for all games in Figure 19. As can be seen, the best results are obtained when using Adam and the MSE loss, in contrast to the standard practice of using RMSProp and the Huber loss.

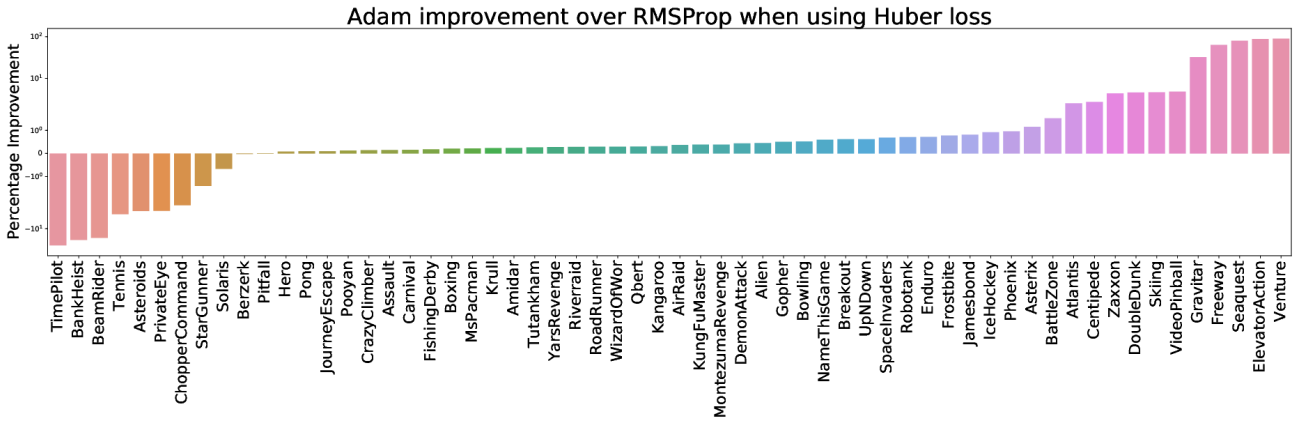


*Figure 14.* Comparing Adam vs RMSProp when both optimizers use the Huber loss.

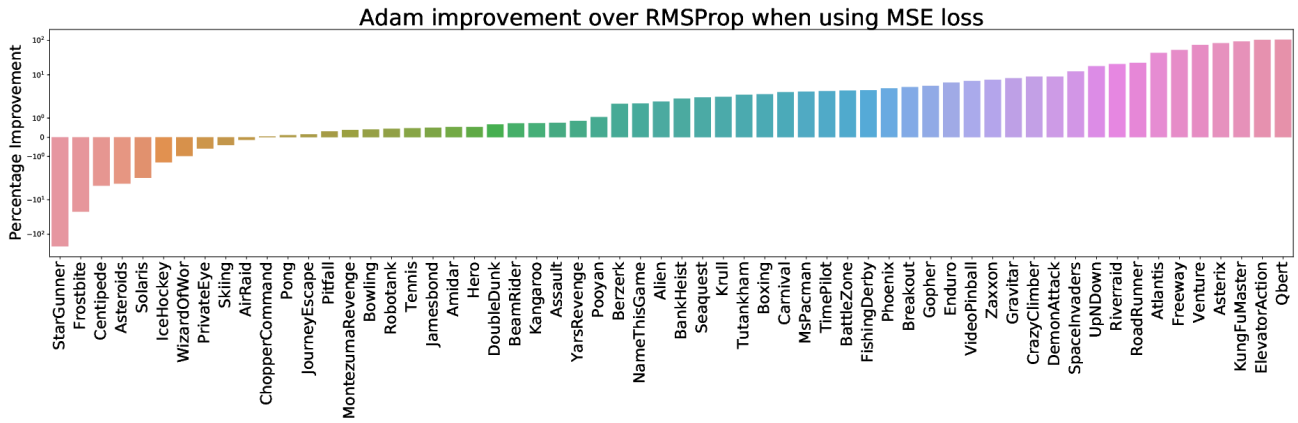


*Figure 15.* Comparing Adam vs RMSProp when both optimizers use the MSE loss.

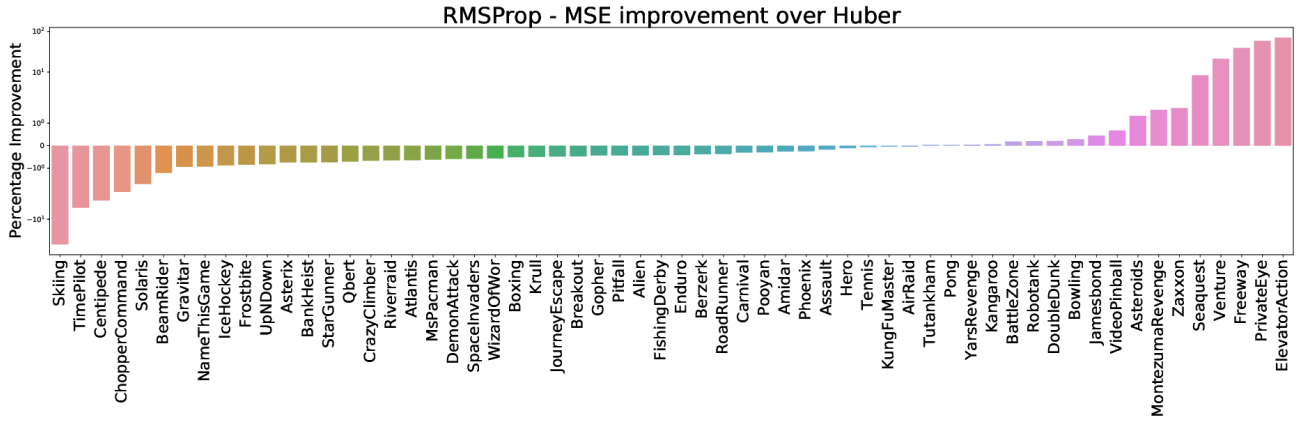


*Figure 16.* Comparing the MSE vs Huber loss when using the RMSProp optimizer.

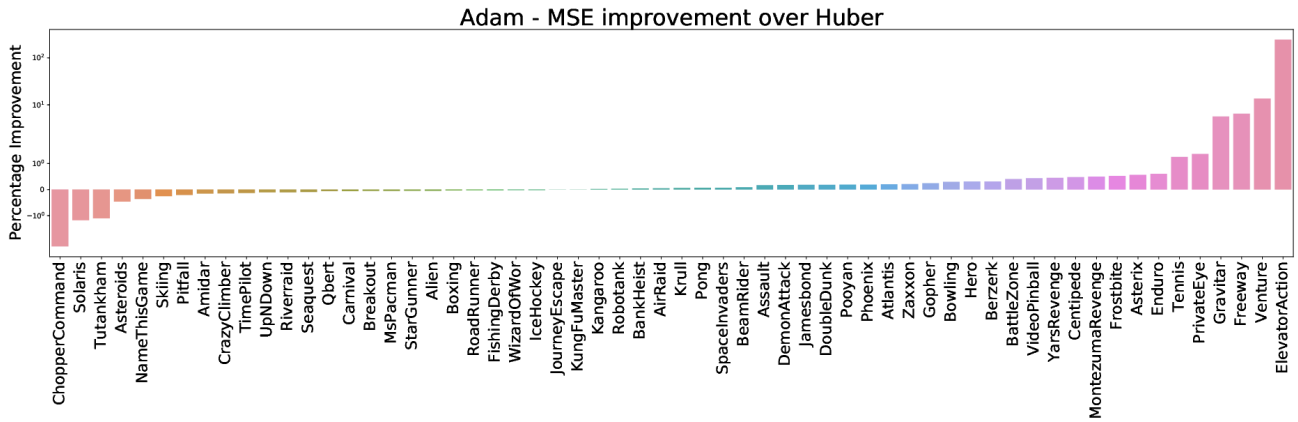


*Figure 17.* Comparing the MSE vs Huber loss when using the Adam optimizer.

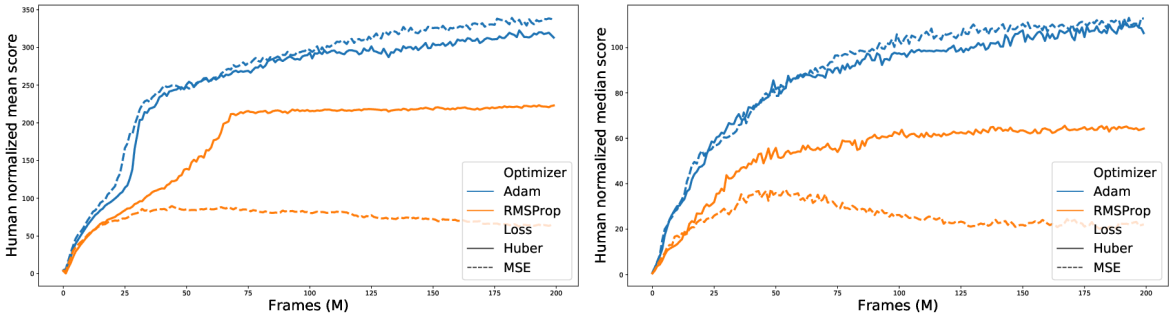


*Figure 18.* Comparison of the various optimizer-loss combinations with human normalized scores (mean left, median right). All results report the average of 5 independent runs.

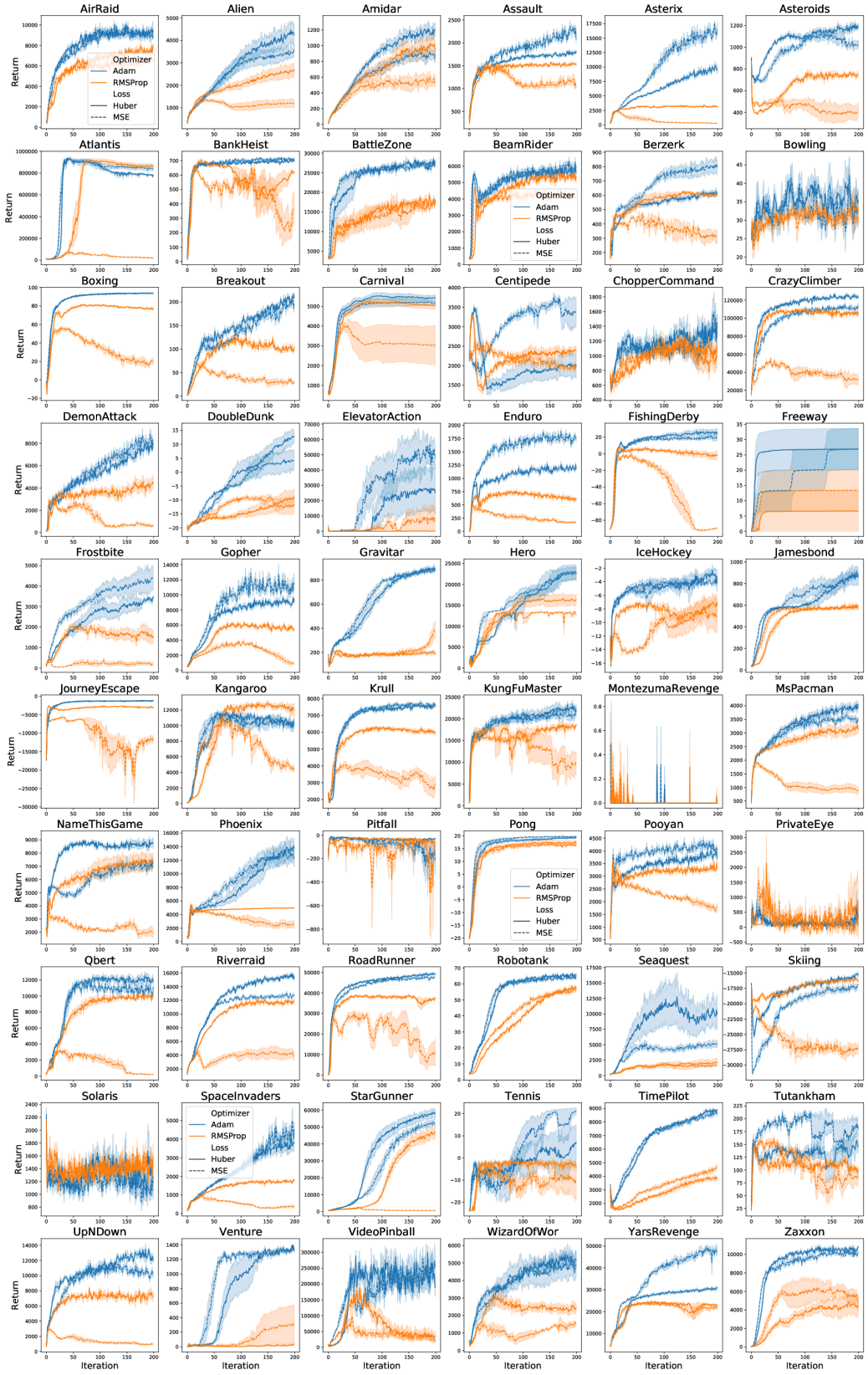


*Figure 19.* Comparison of using Adam (blue) versus RMSProp (orange) with Huber loss (solid) versus MSE loss (dashed) on all 60 Atari games. Each combination was run over 5 independent runs and the shaded areas represent 75% confidence intervals.

# I. Rainbow flavours, full results

In this section we present the complete results for all the environments when comparing DQN against the various Rainbow flavours (Figure 20).

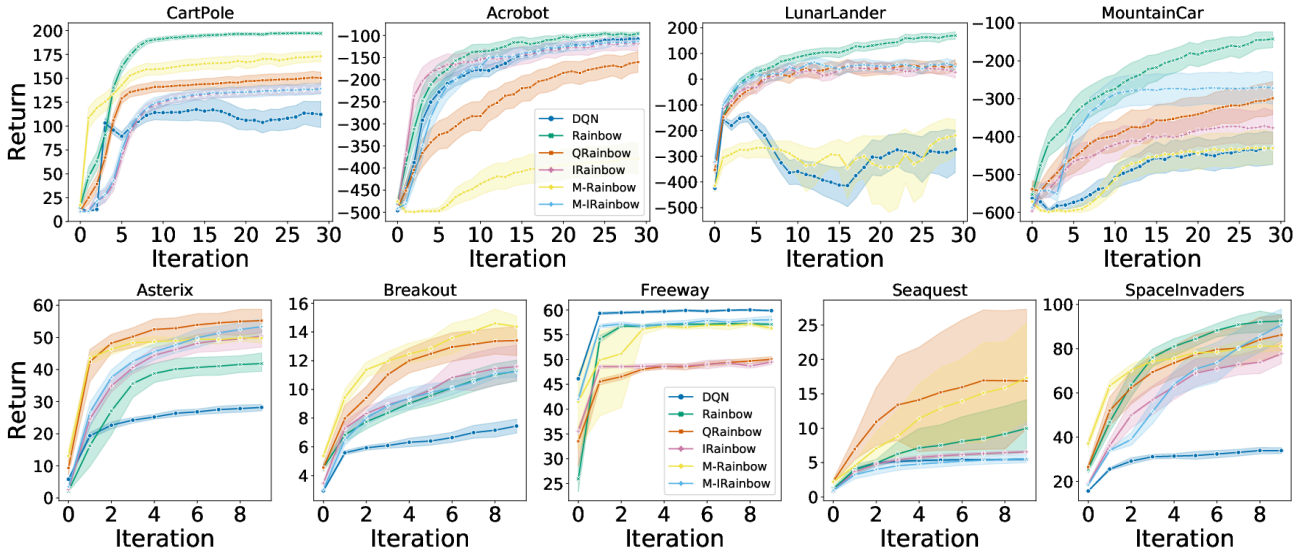


*Figure 20.* Comparing DQN against the various Rainbow flavours.