Lifting the Veil on Hyper-parameters for Value-based Deep Reinforcement Learning

João G.M. Araújo*
Cohere AI
joaogui1@cohere.ai

Johan S. Obando-Ceron*
MILA, Université de Montréal
johan.ceron@mila.quebec

Pablo Samuel Castro
Google Research, Brain Team
psc@google.com

Abstract

Successful applications of deep reinforcement learning (deep RL) combine algorithmic design and careful hyper-parameter selection. The former often comes from iterative improvements over existing algorithms, while the latter is either inherited from prior methods or tuned for the specific method being introduced. Although critical to a method's performance, the effect of the various hyper-parameter choices are often overlooked in favour of algorithmic advances. In this paper, we perform an initial empirical investigation into a number of often-overlooked hyper-parameters for value-based deep RL agents, demonstrating their varying levels of importance. We conduct this study on a varied set of classic control environments which helps highlight the effect each environment has on an algorithm's hyper-parameter sensitivity.

1 Introduction

Deep reinforcement learning (deep RL) is a burgeoning research area with an astounding number of theoretical and empirical advances. Algorithmic progress is usually measured by comparing to pre-existing baselines on a set of established benchmarks, where the proposed method is typically evaluated using a single (or small set of) hyper-parameter choices. While this approach helps demonstrate the *potential* of new algorithms, they are less effective at demonstrating their *robustness* to varying hyper-parameters, especially those that have been "inherited" from prior methods. We will refer to these types of publications as *academic deep RL*.

There have been a number of recent success stories in applying deep RL to large-scale real-world tasks [Silver et al., 2016, Bellemare et al., 2020, Mirhoseini et al., 2021], but in all these cases very specific design decisions (including hyper-parameter choices) were necessary for them to work effectively; often, differing from the design choices made in the algorithms leveraged by these works. We will refer to these types of publications as *applied deep RL*.

Although tackling the same underlying problem, there exists a gap between academic and applied deep RL. While progress in the applied side leverages advances made in the academic side, it typically requires a group of highly-specialized researchers to successfully adapt the academic insights into a workable agent; this produces a *de-facto* barrier for non-academics to successfully apply deep RL to their problem. On the other hand, many of the components ultimately used for large-scale real-world problems tend to be specific, resulting in less-than-ideal academic uptake.

*Equal contribution.

Deep Reinforcement Learning workshop at NeurIPS 2021.

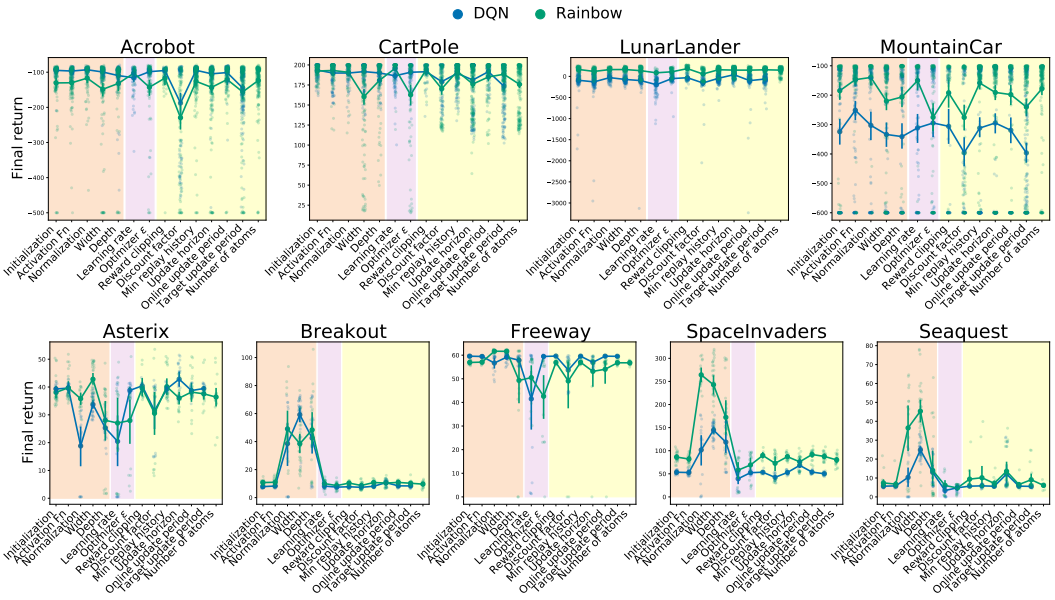

Figure 1: Comparison of hyper-parameter sensitivity for DQN and Rainbow on classic control (top) and MinAtar (bottom) environments. The colors represent groups of hyper-parameters: network components (■, subsection 3.1), optimizer components (■, subsection 3.2), algorithmic components (■, subsection 3.3). Note that the hyper-parameters considered are only those that are within a reasonable range of the default settings.

We argue that a key contributor to the gap between academic and applied deep RL is a poor empirical understanding of the impact of the aforementioned hyper-parameters on the performance of the overall algorithm. In this paper, we begin to "lift the veil" on some commonly overlooked hyper-parameters for value-based deep RL in the hope that they help bridge this gap. We aim to (at least partially) answer the following questions:

1. How sensitive are deep reinforcement learning algorithms to variations in hyper-parameter selection?
2. Does the choice of environment vary the sensitivity to hyper-parameter selection?
3. Are existing methods under-performing due to a poor hyper-parameter choice?

2 Background

Reinforcement learning problems are typically formulated as a Markov decision process (MDP), which consists of a 5-tuple $\langle \mathcal{X}, \mathcal{A}, R, \mathcal{P}, \gamma \rangle$, where $\mathcal{X}$ denotes the (possibly infinite) state space, $\mathcal{A}$ denotes a finite set of actions, $R : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\mathcal{P} : \mathcal{X} \times \mathcal{A} \to Dist(\mathcal{X})$ encodes the transition dynamics (often written as $\mathcal{P}(x'|x, a)$), and $\gamma \in [0, 1)$ is a discount factor. The principal objective of reinforcement learning is to learn a *behaviour policy* $\pi : \mathcal{X} \to Dist(\mathcal{A})$ that maximizes the discounted sum of expected rewards. While there are a number of valid approaches (see, e.g. Sutton and Barto [1998]), in this paper we focus on *value-based* methods. These aim to learn an approximation to the so-called Q^*-values, defined via the Bellman recurrence: $Q^*(x, a) := R(x, a) + \gamma \mathbb{E}_{x' \sim \mathcal{P}(x,a)}[\max_{a' \in \mathcal{A}} Q^*(x', a')]$. The optimal policy π^* can then be obtained from Q^* as $\pi^*(x) := \max_{a \in \mathcal{A}} Q^*(x, a)$.

One of the most common approaches for learning Q^* is via the method of temporal differences: given an estimate Q and a transition tuple (x, a, r, x'), we can obtain a new estimate via: $Q(x, a) \leftarrow Q(x, a) + \alpha \left[r + \gamma \max_{a' \in \mathcal{A}} Q(x', a') - Q(x, a)\right]$, where α is a learning rate. We often refer to the term $[r + \gamma \max_{a' \in \mathcal{A}} Q(x', a')]$ as the *Bellman target*.

2.1 DQN

Mnih et al. [2015] defined DQN by combining temporal-difference learning with deep networks, and demonstrated its capabilities in achieving super-human performance on the Arcade Learning Environment (ALE) [Bellemare et al., 2012]. Specifically, a deep network, parameterized by a vector θ, was trained to approximate Q^*: $Q_\theta \approx Q^*$. Mnih et al. [2015] introduced two key components that helped stabilize the learning process.

The first was the use of a large *replay buffer* to store experienced transitions and, after collecting a sufficient number of transitions (referred to as the *min replay history*), use samples from that buffer to update the network. By having many transitions in the buffer and sampling from it one can reduce the dependency between the elements of a training batch, which helps with neural network training, in addition to leveraging speedups from hardware like GPUs. However, this results in *off-policy* learning, which means that the agent is learning from experience obtained from a different (e.g. older) policy than the policy it is currently using to act. Exploring the difficulties in off-policy learning is an active area of research; nonetheless, the use of replay buffers is ubiquitous in deep RL.

The second was the use of a *target* network (parameterized by $\bar{\theta}$), in addition to the main *online* network, for stabler bootstrapping targets. The Q-update then becomes

$$\Delta Q_\theta = -\alpha \nabla_\theta \left(r_t + \gamma \max_{a'} Q_{\bar{\theta}}(x', a') - Q_\theta(x, a) \right)^2$$

The target network parameters are not updated by gradient descent, but rather they are synchronized with the online parameters at a lower frequency (e.g. $\bar{\theta} \leftarrow \theta$). We refer to the frequency of online network updates (via gradient descent) as the *online update period*, and the frequency of the target network updates as the *target update period*.

2.2 Rainbow

Although DQN benchmarked on the 57 ALE games with the same set of hyper-parameters, Anschel et al. [2017] and Cini et al. [2020] demonstrated that in some environments it can prove to be rather unstable resulting in degraded performance. There were a number of papers that improved on it to improve its stability and performance. Hessel et al. [2018] combined many of these into a single agent they called "Rainbow". Specifically, they combined DQN with double Q-learning [van Hasselt et al., 2016b], prioritized experience replay [Schaul et al., 2016], dueling networks [Wang et al., 2016], multi-step learning [Sutton, 1988], noisy nets [Fortunato et al., 2018], and distributional reinforcement learning [Bellemare et al., 2017].

2.3 Experimental details

As suggested by Obando-Ceron and Castro [2021], we evaluate on four classic control environments (CartPole, Acrobot, LunarLander, and MountainCar)[2], which can be useful for conducting thorough investigations with numerous independent runs. Our implementation is based on the Dopamine framework [Castro et al., 2018], but the default hyper-parameter settings used for the online experiments are those specified by Obando-Ceron and Castro [2021]. All experiments were run on a CPU, and we report the mean and 95% confidence intervals, averaged over 30 independent seeds.

We also evaluated on the MinAtar environment [Young and Tian, 2019], as they can help shed some light into the effect of convolutional layers and more complex environments. For the MinAtar experiments we use the default settings suggested by Obando-Ceron and Castro [2021], except where noted. Given the increased computational complexity of this suite, we ran each setting with 5 independent seeds (each on a separate GPU), and we report the mean 75% confidence intervals.

3 Lifting the veil

We organize our experiments into three groups of hyper-parameters: *network components* (subsection 3.1), *optimization hyper-parameters* (subsection 3.2), and *algorithmic parameters* (subsection 3.3). Figure 1 presents an overall summary of these experiments, and we will present and discuss

[2] Available in the OpenAI Gym library [Brockman et al., 2016]

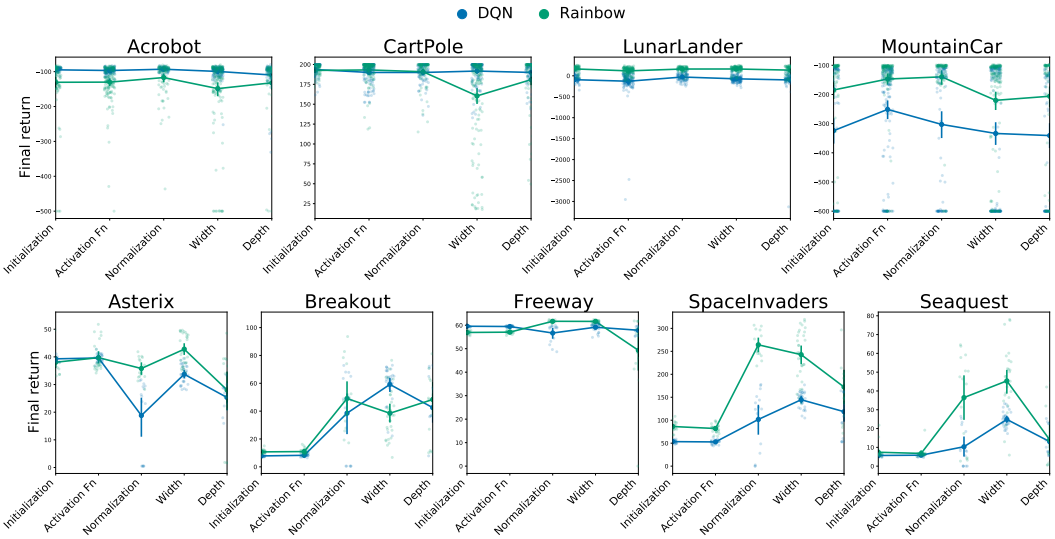

Figure 2: Comparison of hyper-parameter sensitivity for DQN and Rainbow, for the network component hyper-parameters (subsection 3.1).

each individual group below. Due to space constraints we include all the figures in Appendix A and include only aggregate plots in the main paper.

3.1 Network components

The aggregate results for these experiments are summarized in Figure 2.

Initializations Initialization is the term used to define the initial values for the parameters of a neural network. While there exists a broad literature studying initialization strategies for neural networks [Glorot and Bengio, 2010, He et al., 2015, Saxe et al., 2014]. With a few exceptions, these strategies have not been studied in detail in deep reinforcement learning [Andrychowicz et al., 2020, Hussenot et al., 2021, Paine et al., 2020].

In Figure 5 and Figure 6 we can observe that, for the most part, there is little difference when varying the initialization scheme. This is somewhat expected due to RL's "self-correcting nature"; that is, it is not optimizing towards a fixed target, but rather a shifting one (as a consequence of bootstrapping). There are two notable exceptions worth mentioning:

(1) All-zeros and all-ones initialization almost always fails to learn with DQN. This is most likely due to zero gradients occurring as a consequence of the combination of constant predictions with little reward variability. In Rainbow this issue is not present, perhaps due to the presence of noisy networks (which break the constancy of predictions). **(2)** DQN on MountainCar seems to be rather sensitive to the choice of initialization. This is likely due to the properties of this particular environment, which we discuss below.

In Figure 7 we can see that we have the same level of stability in the MinAtar environments.

Activation functions Non-linear activation functions are a fundamental part of Deep Neural Networks, as their removal effectively turns the network into a linear function approximator. Many different activation functions have been proposed for different settings ([Devlin et al., 2019, Elfwing et al., 2018, Dauphin et al., 2017]), yet it's rare to see comparisons between the many possible options [Shamir et al., 2020] and, to the best of our knowledge, there are no previous examples of doing such comparison in the Reinforcement Learning setting.

In Figure 8 (and in Figure 10 for MinAtar environments) we can see that there is little difference across the top activation functions, with DQN on MountainCar once again being a notable exception. However, in Figure 9 we can see that there is a fair bit of variability across the 16 possible activation functions. Of note is how sensitive Rainbow seems to be to these on CartPole, which is arguably the

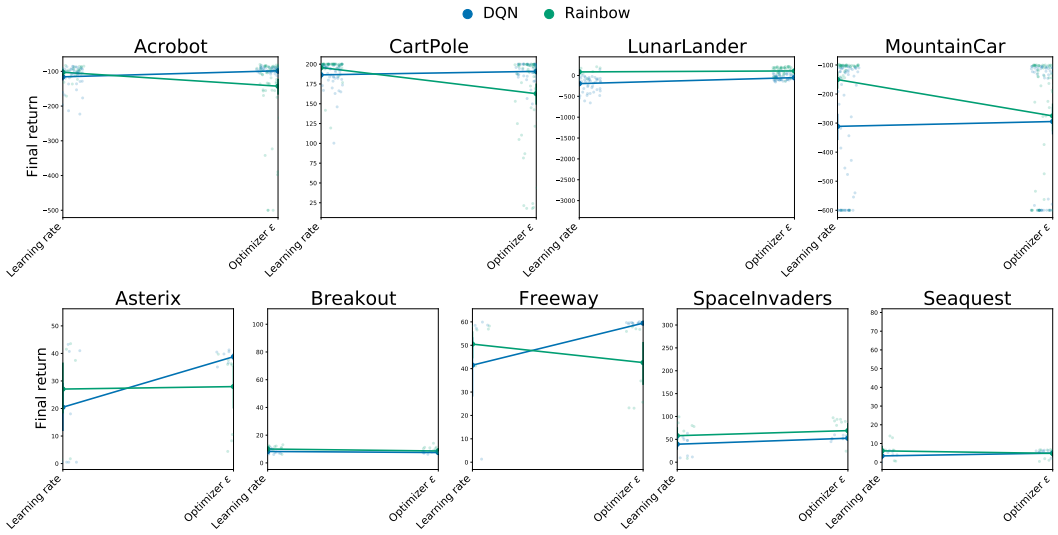

Figure 3: Comparison of hyper-parameter sensitivity for DQN and Rainbow, for the optimizer hyper-parameters (subsection 3.2).

simplest of all the environments. A more detailed investigation of the effect (and interaction with other components) of activation functions is an interesting avenue for future work.

Normalization Normalization plays an important role in various deep learning applications, and it is well established in supervised learning [Tan and Le, 2020, Xie et al., 2017]. Surprisingly, the use of normalization is fairly rare in deep reinforcement learning, with a few exceptions [Bhatt et al., 2019, Arpit et al., 2019, Lillicrap et al., 2019, Silver et al., 2017]. We explore two types of normalization: **batch normalization** [Ioffe and Szegedy, 2015] and **layer normalization** [Ba et al., 2016]. In Figure 11 we can see that although there are some minor differences, there appears to be little gain to using normalization. This may very well be a consequence of the simplicity of the environments (indeed, the biggest differences are observed in MountainCar) and the shallowness of the networks we're using. Although batch normalization does not seem to provide clear benefits, it also does not seem to hurt (with the exception of MountainCar), a finding that is somewhat at odds with that of Salimans and Kingma [2016], who affirm that batch normalization is not well suited for Deep Reinforcement Learning.

We investigated this further on the MinAtar suite in Figure 12. It is worth noting that since the default setting for MinAtar is to use a single convolutional layer with no dense layers, we added one dense layer (based on the depth results from Figure 14) and applied the normalization on this extra dense layer. The results provide even more evidence that there is little gain to using normalization, and this can in fact hurt performance. It is worth investigating whether applying normalization to the convolutional layer can help address this.

Network capacity Improvements in deep learning architectures have played a vital role in moving forward the state of supervised and unsupervised learning in computer vision, but neural network architecture design for deep reinforcement learning is relatively unexplored, with a few exceptions [Sinha et al., 2020, Andrychowicz et al., 2020].

We vary network capacity via the **depth** (e.g. the number of hidden layers) and the **width** (e.g. the number of neurons) of each hidden layer. In Figure 13 we observe an interesting phenomenon where *fewer* layers seems to help DQN (e.g. MoutainCar) whereas in Rainbow performance is correlated with the number of layers. We hypothesize that this form of network capacity has a non-trivial interaction with the type of loss (e.g. expectational versus distributional), and may help further understand the difference between the two forms of TD-learning [Lyle et al., 2019]. Huh et al. [2021] and Kumar et al. [2020] demonstrated that deeper networks tend to be biased towards lower rank (e.g. simpler) solutions. This may help explain why in DQN, shallower networks work best with Mountaincar (the most complex off the four classic control environments).

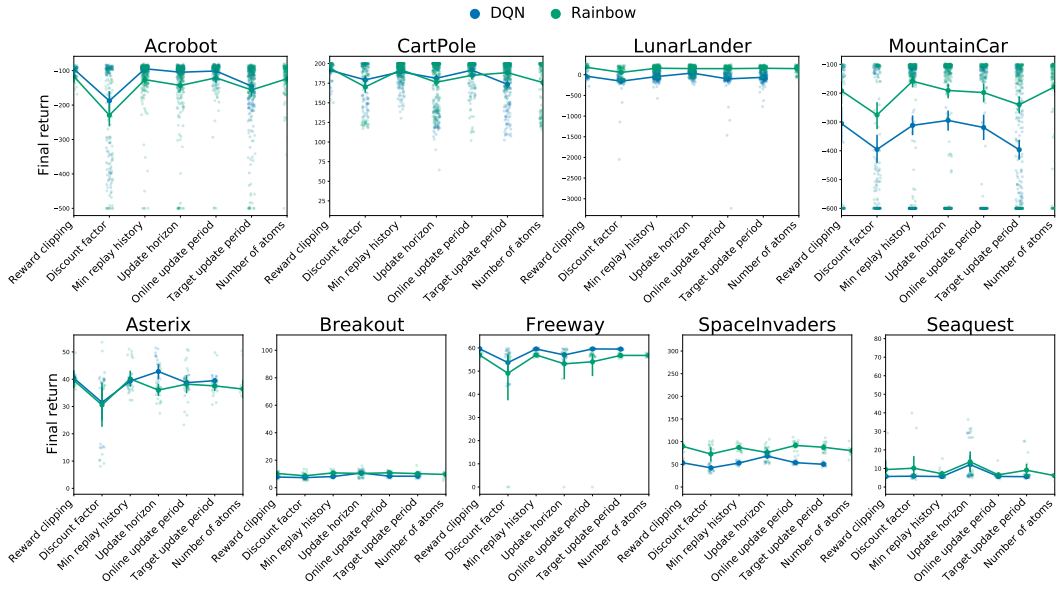

Figure 4: Comparison of hyper-parameter sensitivity for DQN and Rainbow, for the algorithmic hyper-parameters (subsection 3.1).

In Figure 14 we investigate the effect of depth on the MinAtar suite. The default setting for MinAtar is to use a single convolutional layer (depth = 0), so we investigated the effect of adding dense layers after the convolutional layer. In DQN there is a clear benefit to adding more layers, but the best results are obtained when adding only a single layer. It is interesting to note that in Freeway, the simplest of the five games, the best performance is obtained without any extra layers. This strongly suggests that there is a close relationship between network depth and environment complexity, where the best performance is obtained in a "goldilocks region", and this region varies across environments. In Rainbow, we can further observe how much this "goldilocks region" varies from game to game: from a wide region (Asterix) to a very narrow one (Seaquest).

In Figure 15 we can see that, in general, increasing width improves performance. It is somewhat surprising the significant differences observed with Rainbow in CartPole (arguably the easiest of the environments), yet there is no noticeable difference in DQN.

Given the results of the depth experiments, we explored varying width in the MinAtar environments by adding one dense layer after the convolutional layer. In Figure 16 we see surprising variation across environments where performance is sometimes proportional to width (SpaceInvaders), and sometimes inversely proportional (Asterix and Freeway).

3.2 Optimizer hyper-parameters

Although the choice of optimizer is a design choice in itself, Obando-Ceron and Castro [2021] demonstrated the superior performance of the Adam optimizer [Kingma and Ba, 2015] relative to candidates such as RMSProp. We explore two hyper-parameters of this optimizer: **learning rate** and **epsilon** values.The aggregate results for these hyper-parameters are summarized in Figure 3.

In Figure 17 we can see that both algorithms are relatively robust to varying learning rates between 10^{-4} and 0.01; however in Figure 18 we explore a wider range and observe a deterioration in performance in both algorithms. In Figure 19 we can see a fair bit of variation across the MinAtar environments for the learning rates considered, with the ordering based on performance varying between environments (e.g. Asterix versus Breakout on DQN).

We also observe that DQN seems to exhibit greater stability with larger ϵ values than Rainbow (see Figure 20 and Figure 21). To further investigate this observation, we ran the ϵ experiment on the MinAtar suite (Figure 22). We observe that the qualitative differences between DQN and Rainbow

are mostly gone, suggesting that the observations in the classic control environment are perhaps due more to Rainbow's stability on simpler environments.

3.3 Algorithmic parameters

There are a number of algorithmic design choices for RL agents that are often overlooked, but can play a significant role in performance. Hessel et al. [2019] explored this to some extent, focusing on variants of the A2C algorithm [Mnih et al., 2016]. The aggregate results for this group of hyper-parameters is presented in Figure 4.

Reward clipping Reward clipping was an important design decision for the original DQN algorithm, as it enabled normalizing scores across games [Mnih et al., 2015]. Clipping rewards changes the objective, which can result in qualitatively different learned behaviours. This design decision has often been adopted by derivative algorithms for other domains without properly evaluating its efficacy. Indeed, Figure 23 suggests that the agents can actually perform better without reward clipping. This is somewhat at odds with the findings of van Hasselt et al. [2016a], where they found reward-clipping to be useful for learning. In the MinAtar environments we do not see much difference between the two (Figure 24).

Discount factor The importance of γ has been observed in a number of recent works [Amit et al., 2020, Hessel et al., 2019, Gelada and Bellemare, 2019, van Seijen et al., 2019, François-Lavet et al., 2016], and in particular the discrepancy between the γ value used for *training* and the one used for *evaluation*. As Figures 25 and 26 demonstrate, algorithmic performance is rather sensitive to the choice of γ; a result consistent with the findings of Hessel et al. [2019].

In Figure 27 we evaluate three values of γ on the MinAtar environments and see a fair bit of variability across environments (compare DQN on SpaceInvaders versus Seaquest, for instance).

Minimum replay history As mentioned above, the agent stores its experience in a replay memory, from which it then samples mini-batches for learning. It is common practice to only begin sampling from the replay buffer when a minimum number of transitions have been recorded: the minimum replay history. The purpose of this parameter is to avoid overfitting to a sample set of samples at the early stages of training. Perhaps, surprisingly, Figure 28 and Figure 29 suggest that the choice of this parameter has very little effect on the performance of either algorithm.

Update horizon Multi-step learning [Sutton, 1988] computes the temporal difference error using multi-step transition, instead of a single step. DQN uses a single-step update by default, whereas Rainbow chose a 3-step update. In Figure 30 we compare various update horizons. It is interesting to note that DQN is mostly unaffected by the update horizon, whereas Rainbow seems to have degraded performance with a single-step update horizon. It thus seems that one of the Rainbow components benefits from the multi-step update, an interesting question left for future work.

The update horizon has been argued to trade-off between the bias and the variance of the return estimate [Kearns and Singh, 2000]. This effect has been observed in the linear function approximation case, but it has not been very well studied with deep networks. Hernandez-Garcia and Sutton [2019] perform a statistical analysis on the effect the update horizon has on the performance of six reinforcement algorithms in MountainCar and find that the performance of each algorithm at the beginning of training was better with larger update horizons. These findings are confirmed by our results on MountainCar (Figure 30), but they are less evident on the other environments. Indeed, DQN seems to perform *worse* with a larger update horizon on Acrobot. These findings further highlight the sensitivity of the different DRL components relative to the environment on which they are run.

In Figure 31 we see this effect in high relief, where the benefit of the update horizon seems to be closely tied with environmental complexity. Compare, for instance, both algorithms on Freeway and Seaquest: in the former, an update horizon of 1 performs best, while in the latter an update horizon of 10 yields a *dramatic* improvement for both algorithms.

Update periods For synchronous agents (like DQN and Rainbow), the update of the online network parameters is not done after every step taken in the environment; instead, it is performed at a frequency

specified by the *update period*. The default value used is 4, which means that the online network parameters are only updated after every 4 steps taken in the environment.

The effect of varying the update period for the online network can be seen in Figure 32. The results vary from environment to environment (compare the performance of an update period equal to 8 in Acrobot and MountainCar with DQN); this variability is likely due to the complex learning dynamics of RL and merits further study. Interestingly, this parameter does not seem to have much of an effect on the MinAtar environments (Figure 31).

As mentioned previously, the target network parameters are updated less frequently than the online parameters. It has been observed that less frequent updates can result in improvements [Hernandez-Garcia and Sutton, 2019]. However, our findings in Figure 34 suggest that the best performance is obtained between values of 50-200, but any higher or lower results in decreased performance. We also find that DQN seems to be more sensitive to this choice than Rainbow in the classic control environments, but there is little variation for both DQN and Rainbow in the MinAtar environments (Figure 35).

Number of atoms One of the key components of the Rainbow algorithm is distributional reinforcement learning. With this approach, the output layer predicts the distribution of the returns for each action a in a state s, instead of the mean $Q^\pi(s, a)$. Rainbow uses the distribution parameterization originally proposed by Bellemare et al. [2017]; namely, representing the return distribution as a categorical distribution parameterized by N "atoms". Bellemare et al. [2017] found that empirically setting $N = 51$ proved best for the ALE; in Figure 36 we revisit this design choice.

Somewhat surprisingly, we observe little sensitivity to this choice in the four environments considered. Obando-Ceron and Castro [2021] hypothesized that the role of the number of atoms may be larger when the deep networks include convolutional layers. To investigate this, we repeated this experiment on the MinAtar suite (see Figure 37) and observe that, rather than being dependent on the use of convolutional layers, the role of this parameter is affected by the *environment*. In particular, we observe very little difference between the number of atoms in Asterix, but see a clear difference in SpaceInvaders, where more atoms results in better performance.

4 Discussion

There is a growing interest and concern in the effect of hyperparameter choice, and the ensuing reproducibiliity, for deep reinforcement learning. Henderson et al. [2019] and Islam et al. [2017] highlight issues with reproducibility in RL, including performance differences between different code implementations, hyperparameters, and the high level of non-determinism due to random seeds. Fu et al. [2019] experimentally investigate potential issues of deep Q-learning algorithms. Other large-scale studies similar to ours have been carried out by Andrychowicz et al. [2020], Hussenot et al. [2021], and Paine et al. [2020], but not for online value-based methods.

Figure 1 summarizes our experiments for both agents. We hearken back to the questions raised in the introduction and discuss them with respect to our findings.

How sensitive are DQN and Rainbow to variations in hyper-parameter selection? While in aggregate the two agents have comparable sensitivity, there are some notable exceptions. Network capacity seems to be strongly affected by *environment complexity*; indeed, in the simpler classic control environments Rainbow has a higher sensitivity to over-parameterization, while in the more difficult MinAtar suite both algorithms have noticeable sensitivity. Rainbow seems to be more affected by the choice of ϵ for the Adam optimizer. High values of ϵ make Adam behave more like SGD with Momentum then as diagonal natural gradient descent [Choi et al., 2020]; this leads us to wonder about the relationship between the distributional loss and these different forms of optimization. In the classic control environments, Rainbow seems to be unaffected by different choices of the number of atoms (in contrast with the original findings of Bellemare et al. [2017]); this result may very well be a consequence of the relative simplicity of these environments, as further evidenced by the findings in MinAtar (Figure 37).

Does the choice of environment vary the sensitivity to hyper-parameter selection? This is most certainly the case, as we observed stark qualitative differences between MountainCar and the other

classic control environments. Indeed, any differences between hyper-parameter values seem to be brought into high-relief when evaluated in MountainCar. Additionally, a somewhat surprising finding is that CartPole (often considered the simplest of all four) proved quite effective at highlighting important qualitative differences between DQN and Rainbow (e.g. network width/depth, optimizer ϵ). We observed similar qualitative differences between the different MinAtar games (e.g. Figure 37 demonstrated the effect of number of atoms is best observed in SpaceInvaders, relative to the other games).

Some of our experiments are interesting in that their behavior is different from all other combinations of actor-environment:

- While for most environments initializations seemed to have little effect on the performance of our agents, we see a much higher effect when varying initializations on DQN on Mountaincar.
- Although in general changing activation functions did not impact performance significantly, we can see that in both DQN on Mountaincar, and Rainbow on Breakout there are clearly activation functions that are better than the rest, most interestingly is how relu6 shows a clear improvement over all other activations in Rainbow on Breakout.
- The next striking behavior comes from the depth experiments in DQN on Mountaicar, where we see shallower networks being significantly better than deeper networks, a result opposite to what we see in the other combinations of agents and classic control environments.
- The experimental results on MinAtar show that the optimal choice for a number of hyper-parameters are quite sensitive to the environment itself. Specifically, see the results and discussion on varying learning rates, layer width, γ, and the update horizon.

Are existing methods under-performing due to a poor hyper-parameter choice? There are no canonical hyper-parameter values for the environments considered in this paper; the ones we have labeled as "default" are using the values provided by Obando-Ceron and Castro [2021]. However, some interesting findings that merit further investigation are:

- As mentioned previously, there seems to be a “goldilocks region” for network capacity that is conditional on the environment. As we observed in Figure 13, a network with a single hidden layer seems to be sufficient for DQN, and in fact yields improved performance on MountainCar. On the other hand, in Figure 14 we see that deeper networks can help, but *only up to a point*.
- As discussed above, while reward clipping may be useful for the ALE, it is worth revisiting for any new environment as it can sometimes prove detrimental to performance.
- Our results suggest both algorithms are quite sensitive to the choice of γ, and this sensitivity varies across environments. As such, this parameter must be selected with care for new environments.
- While 51 atoms may have proved best for the ALE experiments conducted by Bellemare et al. [2017], our results suggest smaller values can suffice for simpler environments. However, this question can be somewhat sidestepped by using different parameterizations of the return distribution [Dabney et al., 2018a,b].
- Although we provided some insights into some aspects of the replay buffer and network updates, a more thorough investigation into the relationship between the size and frequency of updates, dubbed the *replay ratio* by Fedus et al. [2020] is warranted.

The present work aims at investigating the importance of a broad set of hyper-parameters that need to be chosen when designing and implementing off-policy learning algorithms. Although we found surprising insights on the classic control environments, we would like to run further experiments on the ALE [Bellemare et al., 2012].

References

Ron Amit, Ron Meir, and Kamil Ciosek. Discount factor as a regularizer in reinforcement learning, 2020.

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters in on-policy reinforcement learning? a large-scale empirical study, 2020.

Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning, 2017.

Devansh Arpit, Victor Campos, and Yoshua Bengio. How to initialize your network? robust initialization for weightnorm & resnets, 2019.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, Vol. 47:253–279, 2012. cite arxiv:1207.4708.

Marc G. Bellemare, Will Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *ICML*, 2017.

Marc G. Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C. Machado, Subhodeep Moitra, Sameera S. Ponda, and Ziyu Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, 2020. doi: 10.1038/s41586-020-2939-8. URL `https://doi.org/10.1038/s41586-020-2939-8`.

Aditya Bhatt, Max Argus, Artemij Amiranashvili, and Thomas Brox. Crossnorm: Normalization for off-policy td reinforcement learning, 2019.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G. Bellemare. Dopamine: A research framework for deep reinforcement learning, 2018.

Dami Choi, Christopher J. Shallue, Zachary Nado, Jaehoon Lee, Chris J. Maddison, and George E. Dahl. On empirical comparisons of optimizers for deep learning, 2020.

Andrea Cini, Carlo D'Eramo, Jan Peters, and Cesare Alippi. Deep reinforcement learning with weighted q-learning, 2020.

W. Dabney, M. Rowland, Marc G. Bellemare, and R. Munos. Distributional reinforcement learning with quantile regression. In *AAAI*, 2018a.

Will Dabney, Georg Ostrovski, David Silver, and Remi Munos. Implicit quantile networks for distributional reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1096–1105. PMLR, 2018b.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 933–941. JMLR.org, 2017.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

Stefan Elfwing, E. Uchibe, and K. Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks : the official journal of the International Neural Network Society*, 107:3–11, 2018.

W. Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, H. Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. In *ICML*, 2020.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alexander Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration. 2018.

Vincent François-Lavet, Raphael Fonteneau, and Damien Ernst. How to discount deep reinforcement learning: Towards new dynamic strategies, 2016.

Justin Fu, Aviral Kumar, Matthew Soh, and Sergey Levine. Diagnosing bottlenecks in deep q-learning algorithms, 2019.

Carles Gelada and Marc G. Bellemare. Off-policy deep reinforcement learning by bootstrapping the covariate shift, 2019.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL `https://proceedings.mlr.press/v9/glorot10a.html`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters, 2019.

J. Fernando Hernandez-Garcia and Richard S. Sutton. Understanding multi-step deep reinforcement learning: A systematic study of the dqn target, 2019.

Matteo Hessel, Joseph Modayil, H. V. Hasselt, T. Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, M. G. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 2018.

Matteo Hessel, Hado van Hasselt, Joseph Modayil, and David Silver. On inductive biases in deep reinforcement learning. *CoRR*, abs/1907.02908, 2019. URL `http://arxiv.org/abs/1907.02908`.

Minyoung Huh, Hossein Mobahi, Richard Zhang, Brian Cheung, Pulkit Agrawal, and Phillip Isola. The low-rank simplicity bias in deep networks. *arXiv preprint arXiv:2103.10427*, 2021.

Leonard Hussenot, Marcin Andrychowicz, Damien Vincent, Robert Dadashi, Anton Raichuk, Lukasz Stafiniak, Sertan Girgin, Raphael Marinier, Nikola Momchev, Sabela Ramos, Manu Orsini, Olivier Bachem, Matthieu Geist, and Olivier Pietquin. Hyperparameter selection for imitation learning, 2021.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control, 2017.

Michael J. Kearns and Satinder P. Singh. Bias-variance error bounds for temporal difference updates. In *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, COLT '00, page 142–147, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 155860703X.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning, 2020.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

Clare Lyle, Pablo Samuel Castro, and Marc G. Bellemare. A comparative analysis of expected and distributional reinforcement learning. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI'19)*, 2019.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. A graph placement methodology for fast chip design. *Nature*, 594 (7862):207–212, 2021. doi: 10.1038/s41586-021-03544-w. URL `https://doi.org/10.1038/s41586-021-03544-w`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

Johan S Obando-Ceron and Pablo Samuel Castro. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In *International Conference on Machine Learning (ICML)*, 2021.

Tom Le Paine, Cosmin Paduraru, Andrea Michi, Caglar Gulcehre, Konrad Zolna, Alexander Novikov, Ziyu Wang, and Nando de Freitas. Hyperparameter selection for offline reinforcement learning, 2020.

Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks, 2016.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, 2014.

T. Schaul, John Quan, Ioannis Antonoglou, and D. Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2016.

G. Shamir, Dong Lin, and Lorenzo Coviello. Smooth activations and reproducibility in deep networks. *ArXiv*, abs/2010.09931, 2020.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016. ISSN 0028-0836. doi: 10.1038/nature16961.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 10 2017. doi: 10.1038/nature24270.

Samarth Sinha, Homanga Bharadhwaj, Aravind Srinivas, and Animesh Garg. D2rl: Deep dense architectures in reinforcement learning, 2020.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3 (1):9–44, August 1988.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

Hado van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude, 2016a.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirthieth AAAI Conference On Artificial Intelligence (AAAI), 2016*, 2016b. cite arxiv:1509.06461Comment: AAAI 2016.

Harm van Seijen, Mehdi Fatemi, and Arash Tavakoli. Using a logarithmic mapping to enable lower discount factors in reinforcement learning, 2019.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48, pages 1995–2003, 2016.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks, 2017.

Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments, 2019.

A All figures

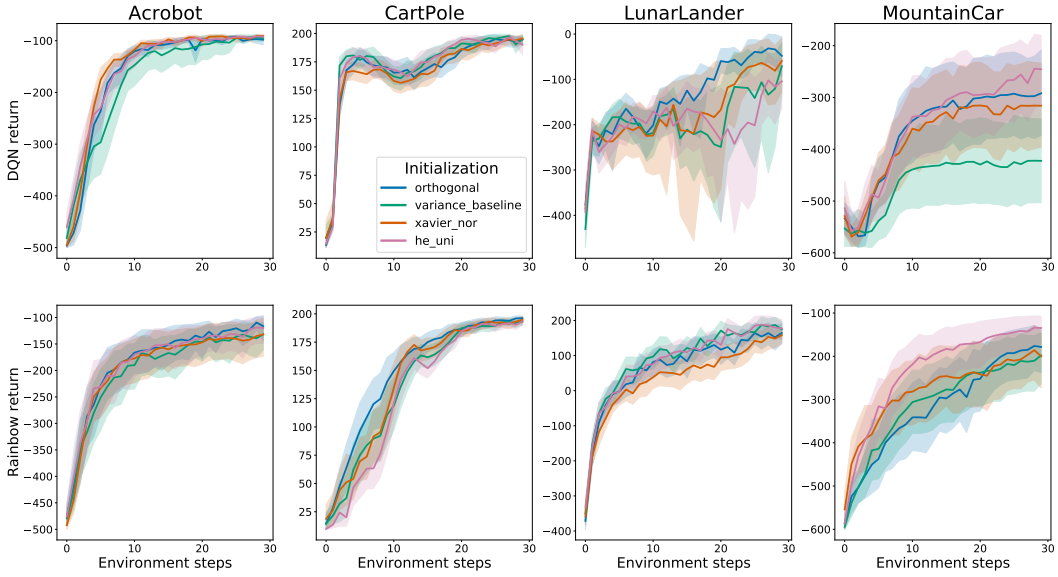

Figure 5: Comparison of initializations on DQN (top) and Rainbow (bottom). The default initialization is Xavier Normal.

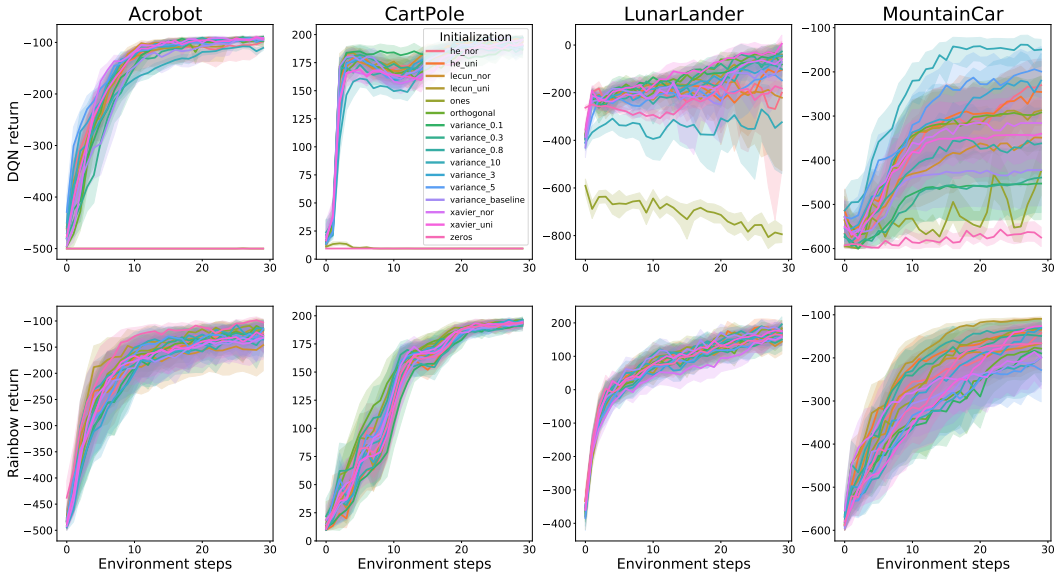

Figure 6: Comparison of all initializations on DQN (top) and Rainbow (bottom). The default initialization is Xavier Normal.

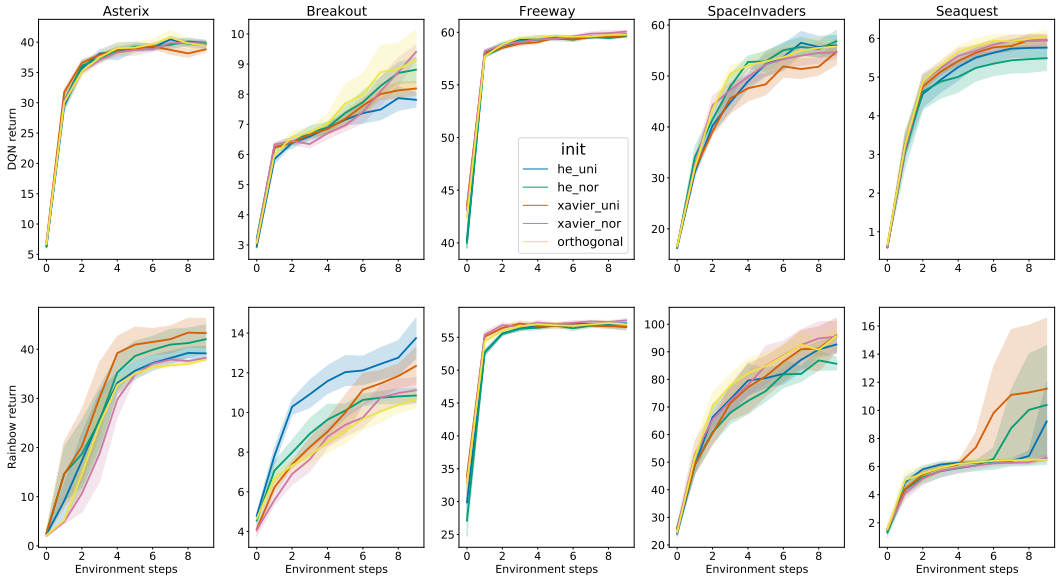

Figure 7: MinAtar comparison of initializations on DQN (top) and Rainbow (bottom). The default initialization is Xavier Normal.

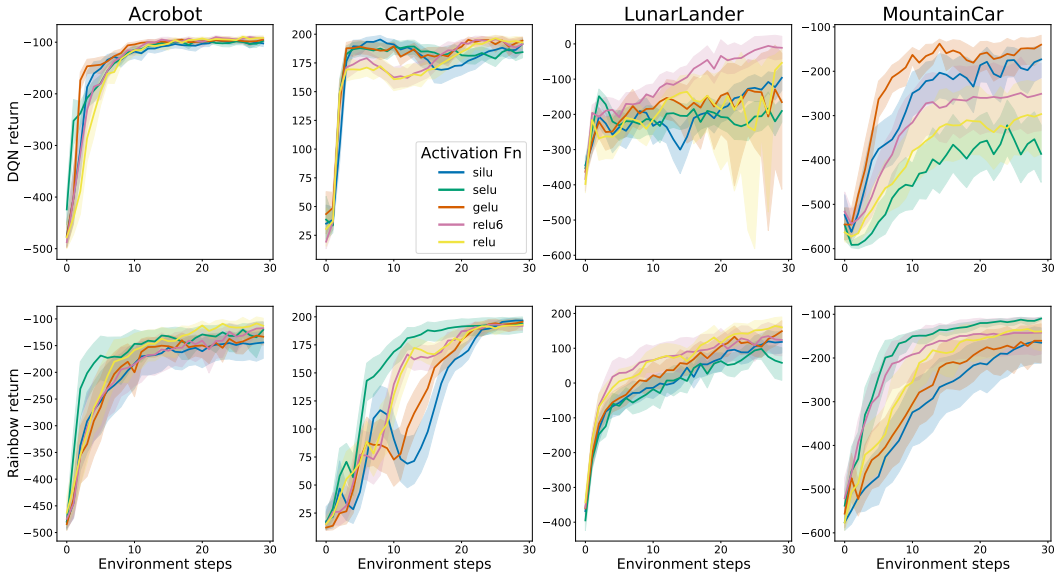

Figure 8: Comparison of activations on DQN (top) and Rainbow (bottom). The Default Activation is ReLU.

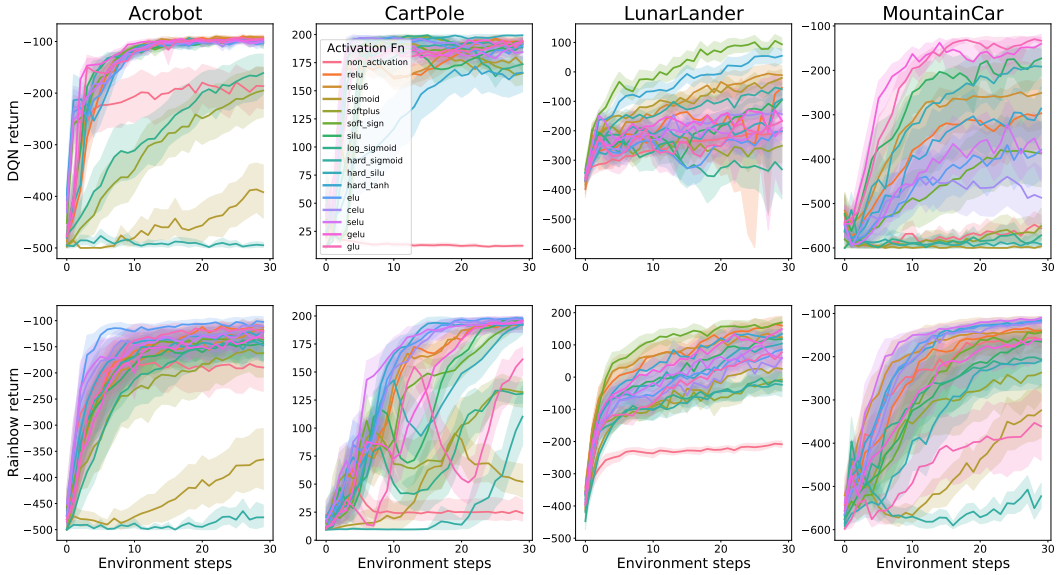

Figure 9: Comparison of all activations on DQN (top) and Rainbow (bottom). The default activation is ReLU.

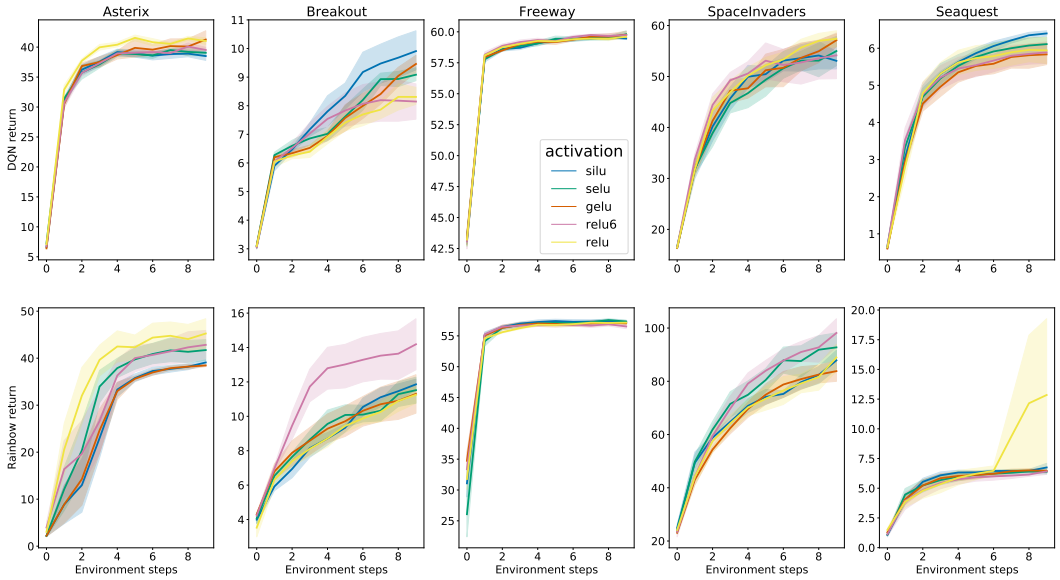

Figure 10: MinAtar comparison of activations on DQN (top) and Rainbow (bottom). The default initialization is Xavier Normal.

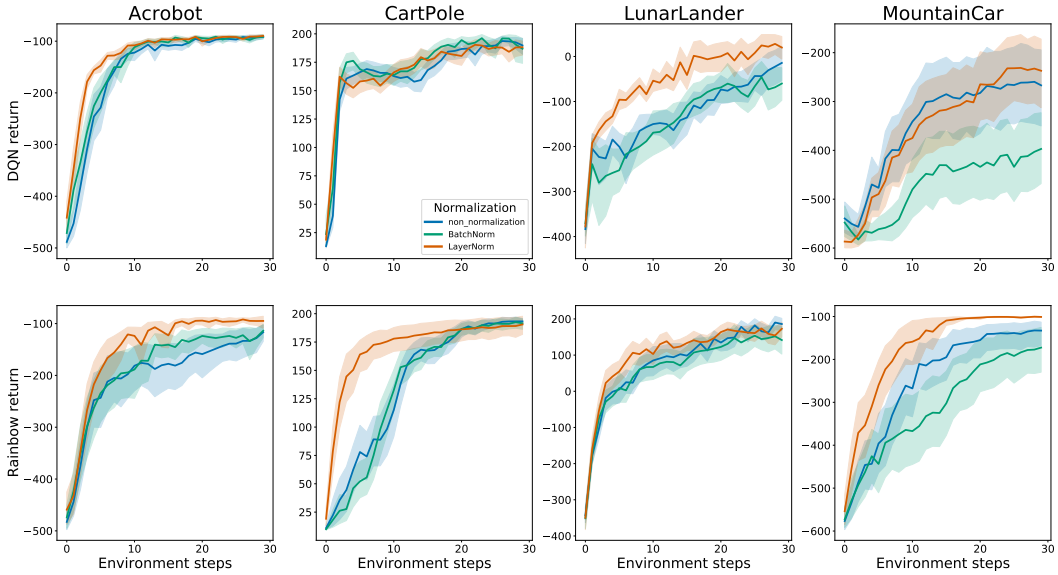

Figure 11: Comparison of normalization on DQN (top) and Rainbow (bottom). The default normalization is None.

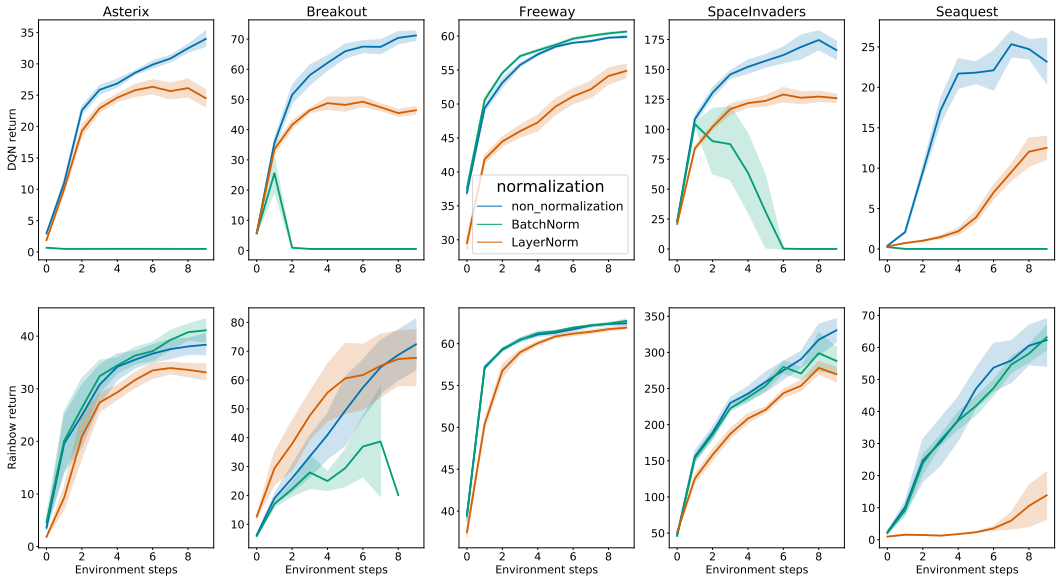

Figure 12: MinAtar comparison of normalization on DQN (top) and Rainbow (bottom). The default normalization is None.

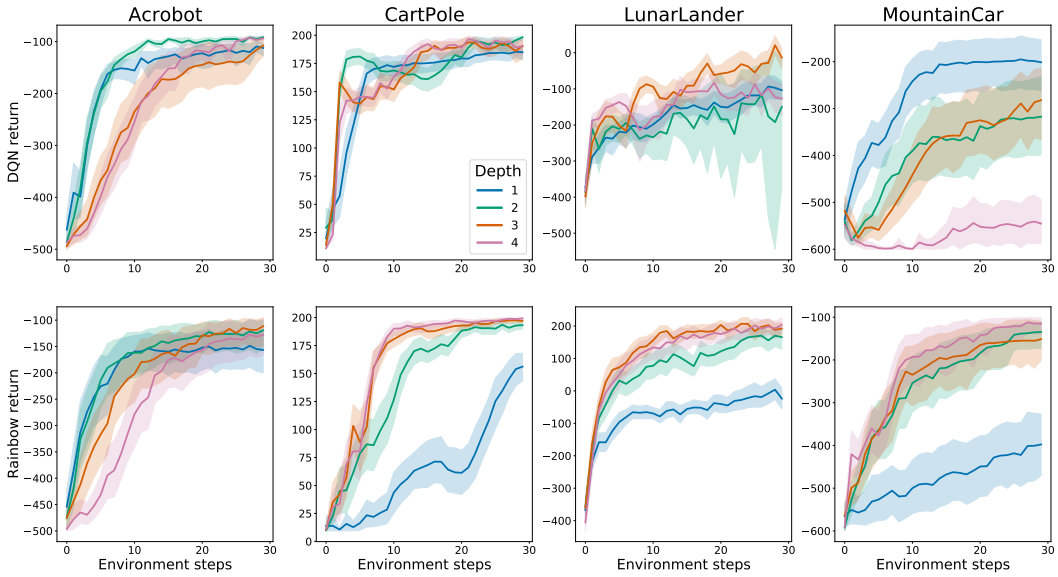

Figure 13: Comparison of depth on DQN (top) and Rainbow (bottom). The default depth is 2.

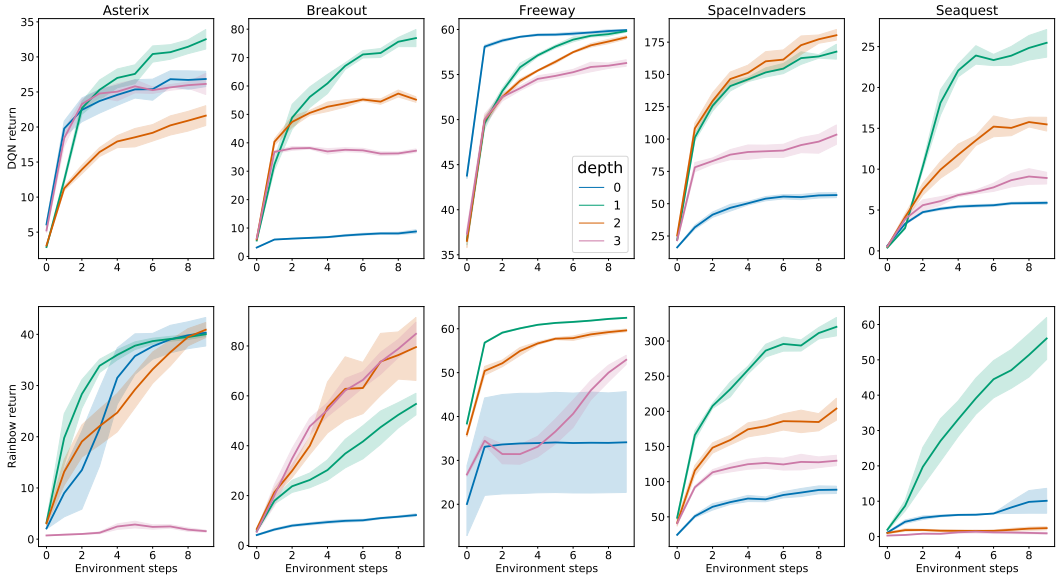

Figure 14: MinAtar comparison of the number of dense layers (on top of the single convolutional layer) on DQN (top) and Rainbow (bottom). The default is a depth of 0.

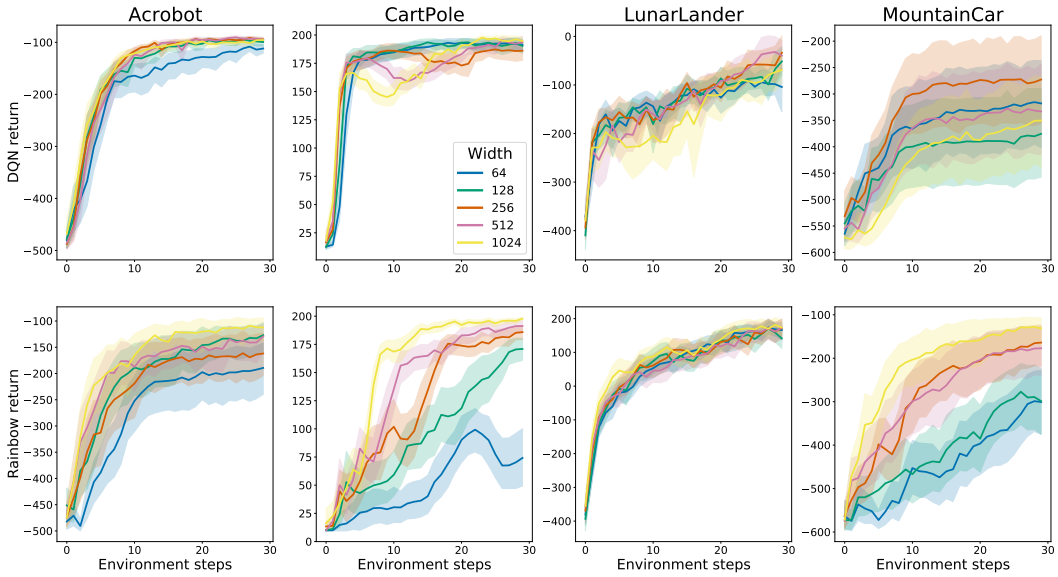

Figure 15: Comparison of width on DQN (top) and Rainbow (bottom). The default width is 512.

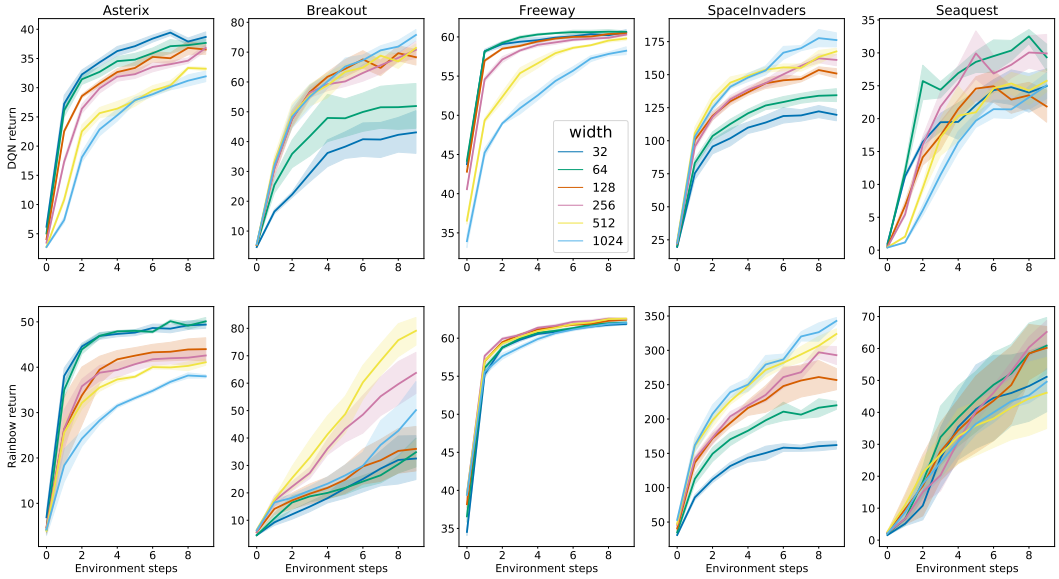

Figure 16: MinAtar comparison of width of one dense layer (on top of the single convolutional layer) on DQN (top) and Rainbow (bottom). The default is a depth of 0.

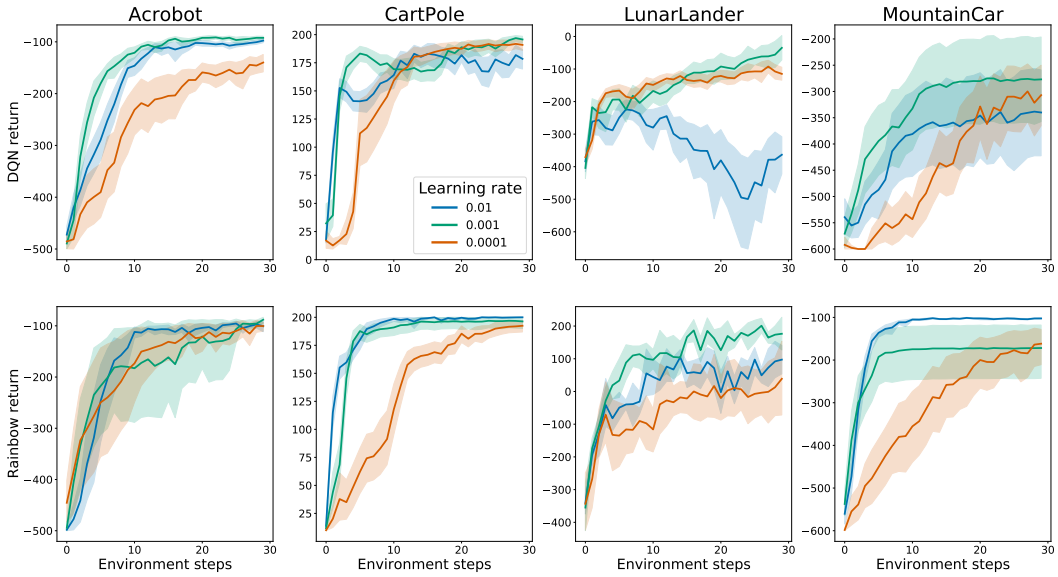

Figure 17: Comparison of learning rates on DQN (top) and Rainbow (bottom). The default learning rate is 0.001, except for MountainCar where it is 0.01.

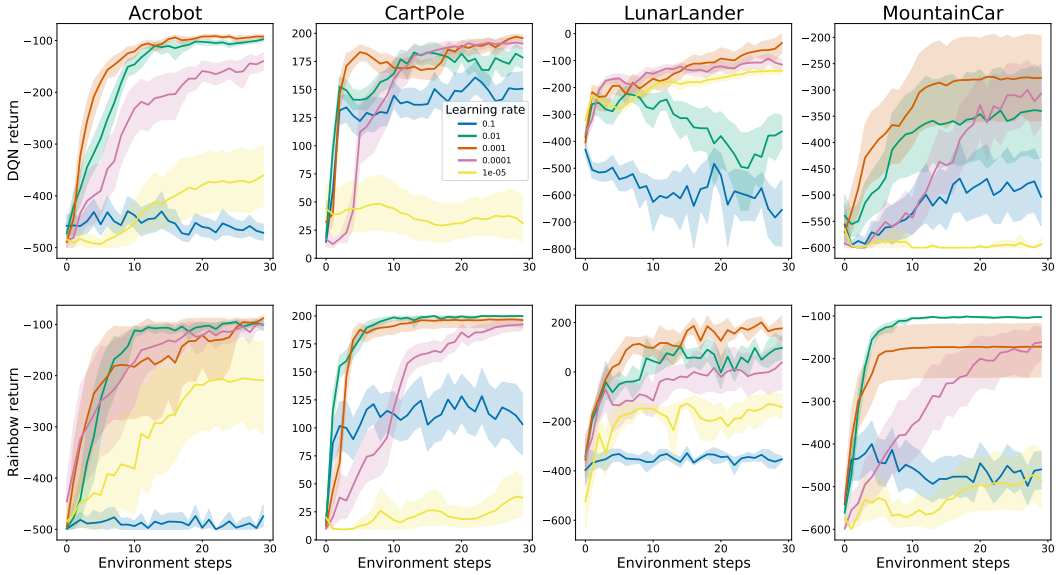

Figure 18: Comparison of all learning rates on DQN (top) and Rainbow (bottom). The default learning rate is 0.001, except for MountainCar where it is 0.01.

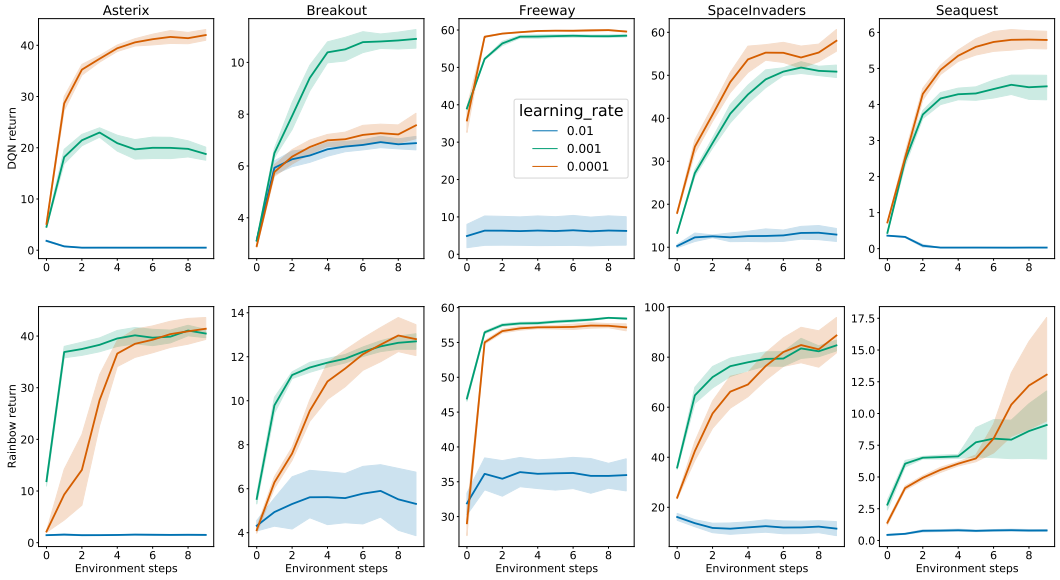

Figure 19: MinAtar comparison of learning rates on DQN (top) and Rainbow (bottom).

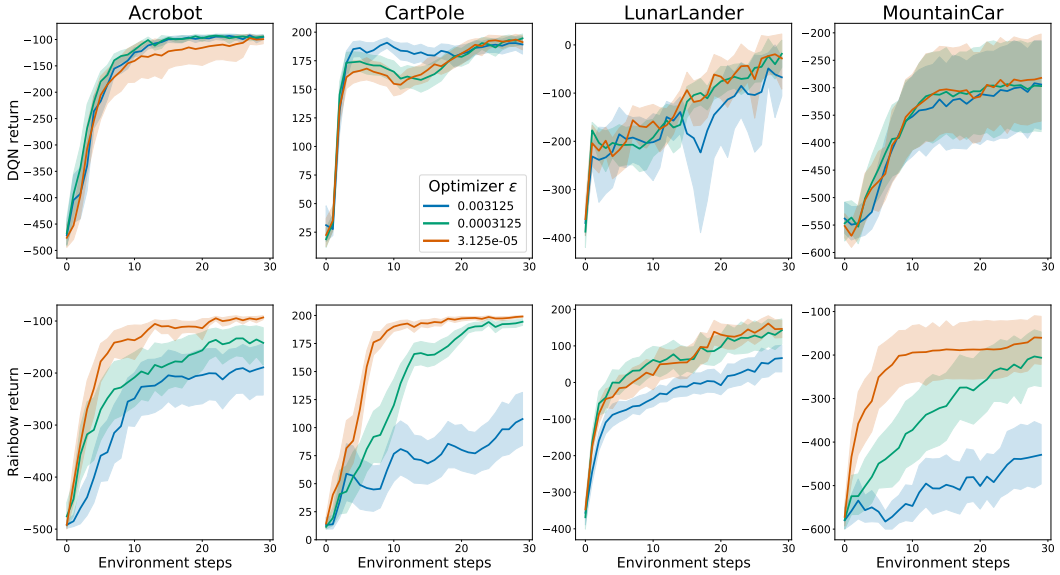

Figure 20: Comparison of optimizer ϵ on DQN (top) and Rainbow (bottom). The default ϵ is 0.0003125.

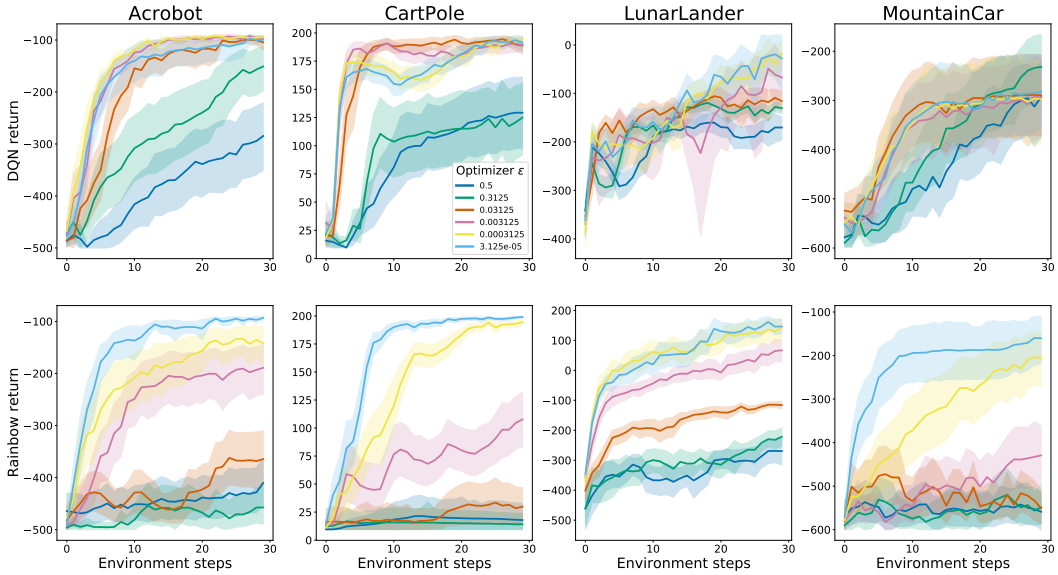

Figure 21: Comparison of all optimizer ϵ values on DQN (top) and Rainbow (bottom). The default ϵ is 0.0003125.

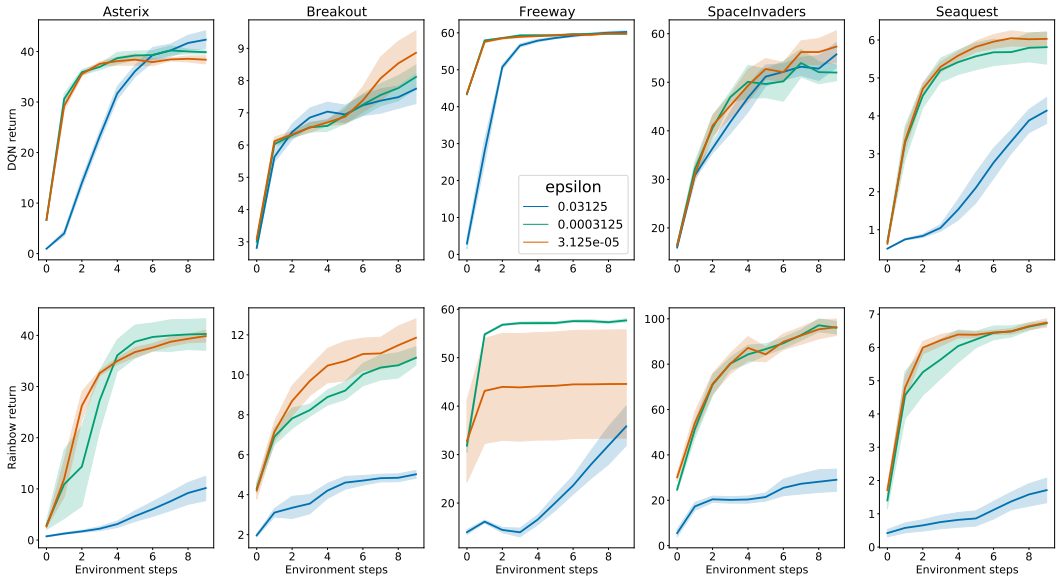

Figure 22: Minatar comparison of optimizer ϵ on DQN (top) and Rainbow (bottom). The default ϵ is 0.0003125.

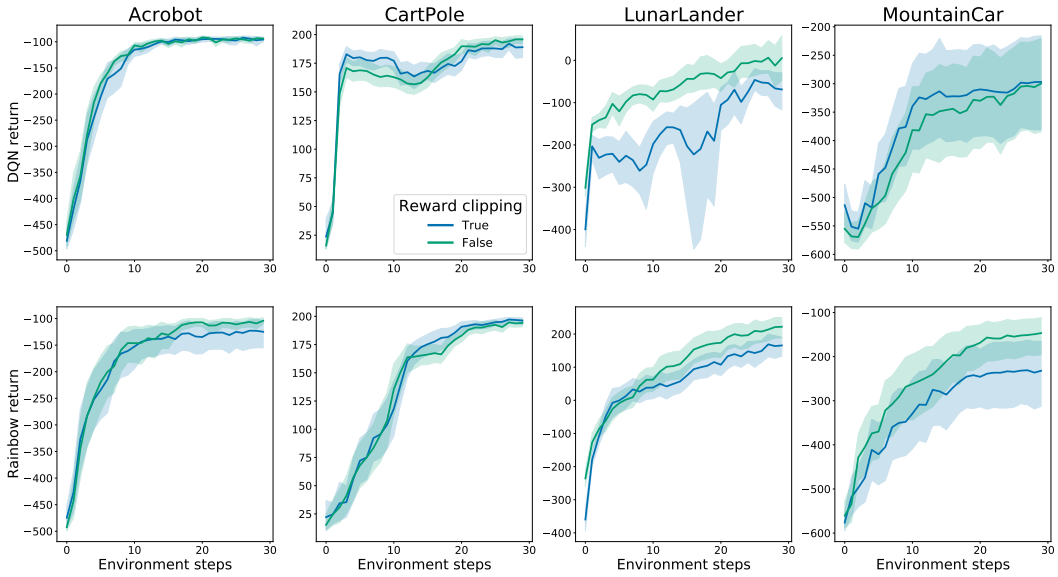

Figure 23: Comparison of reward clipping on DQN (top) and Rainbow (bottom). The default is True.

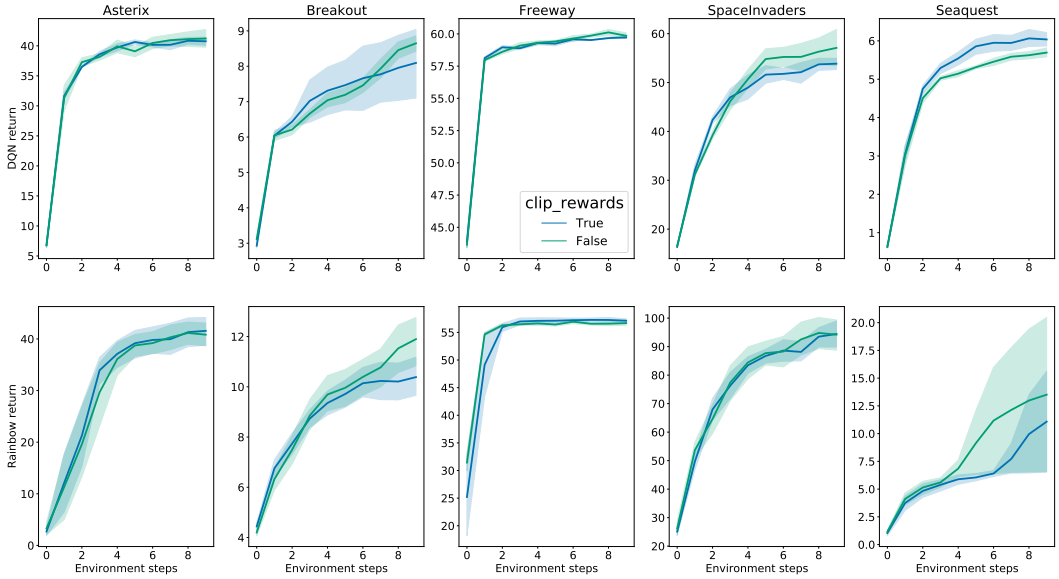

Figure 24: MinAtar comparison of reward clipping on DQN (top) and Rainbow (bottom). The default is True.

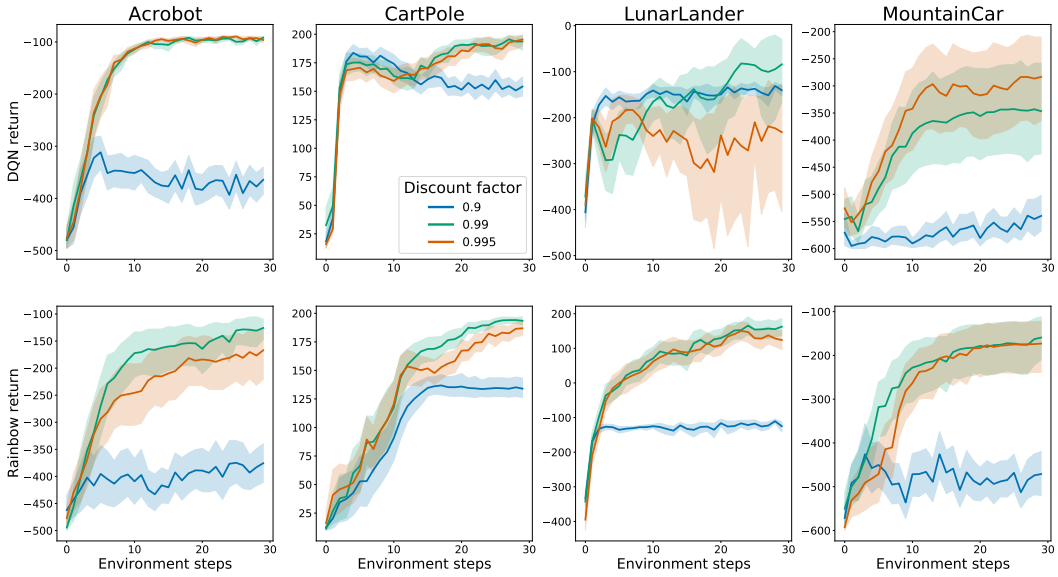

Figure 25: Comparison of γ on DQN (top) and Rainbow (bottom). The default γ is 0.99.

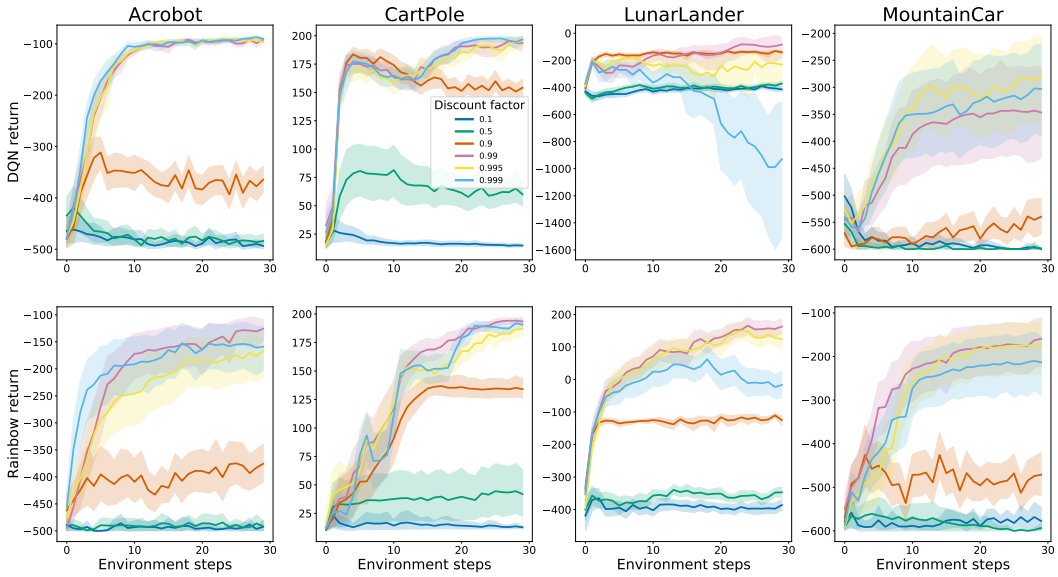

Figure 26: Comparison of all γ values on DQN (top) and Rainbow (bottom). The default γ is 0.99.

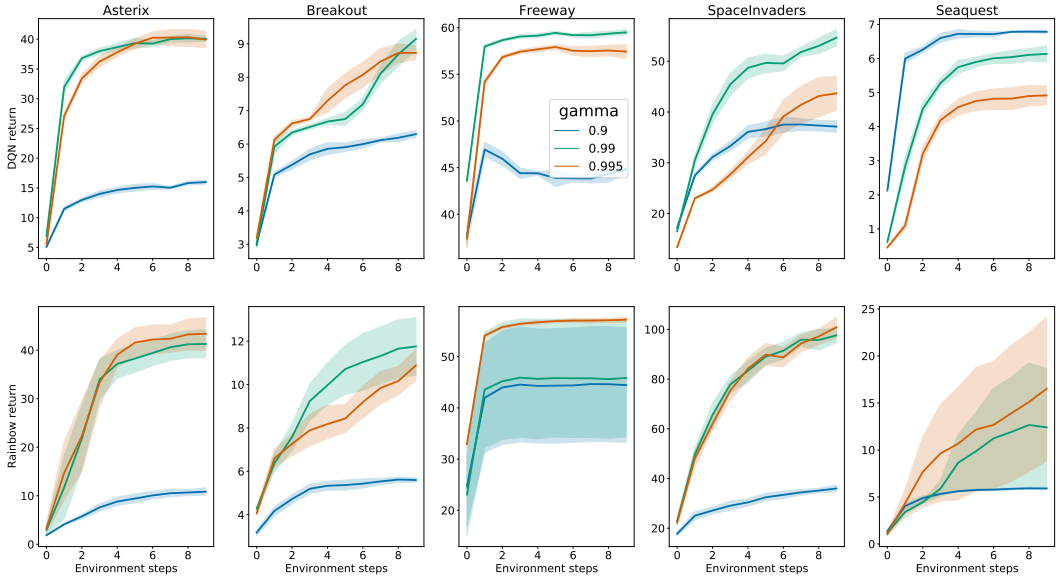

Figure 27: MinAtar comparison of γ on DQN (top) and Rainbow (bottom). The default γ is 0.99.

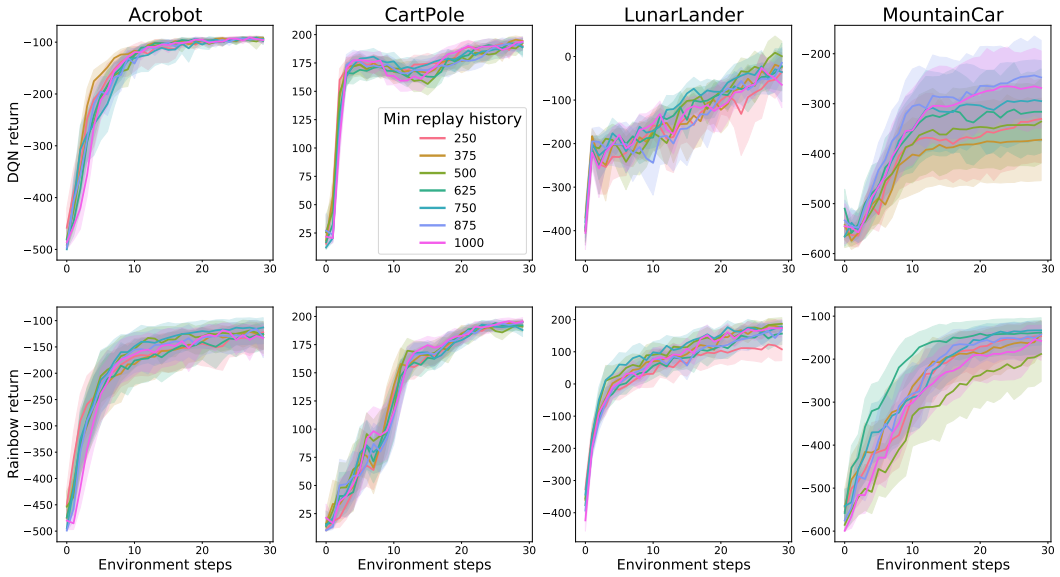

Figure 28: Comparison of minimum replay history on DQN (top) and Rainbow (bottom). The default value is 500.

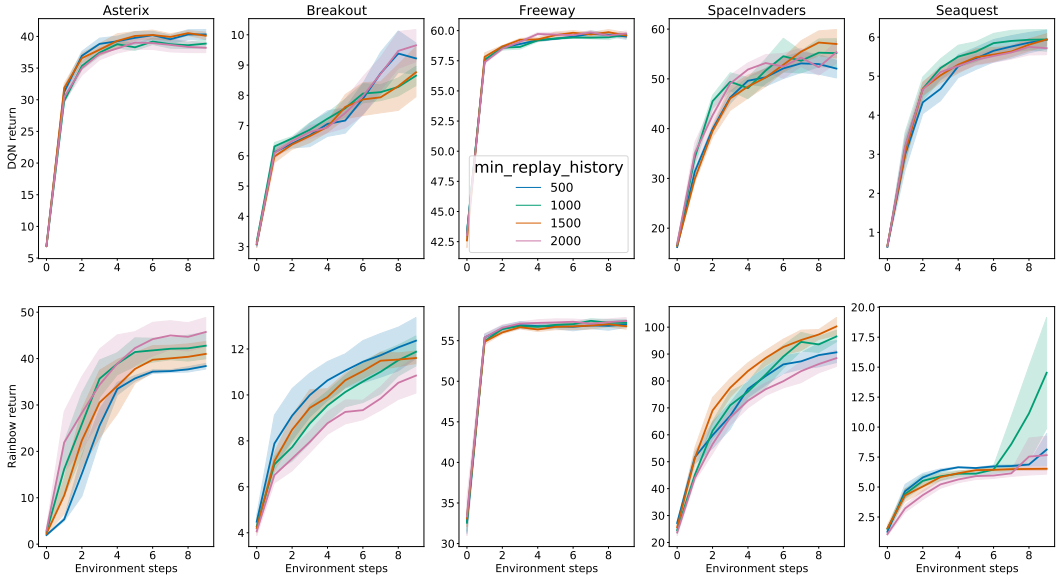

Figure 29: MinAtar comparison of minimum replay history on DQN (top) and Rainbow (bottom).

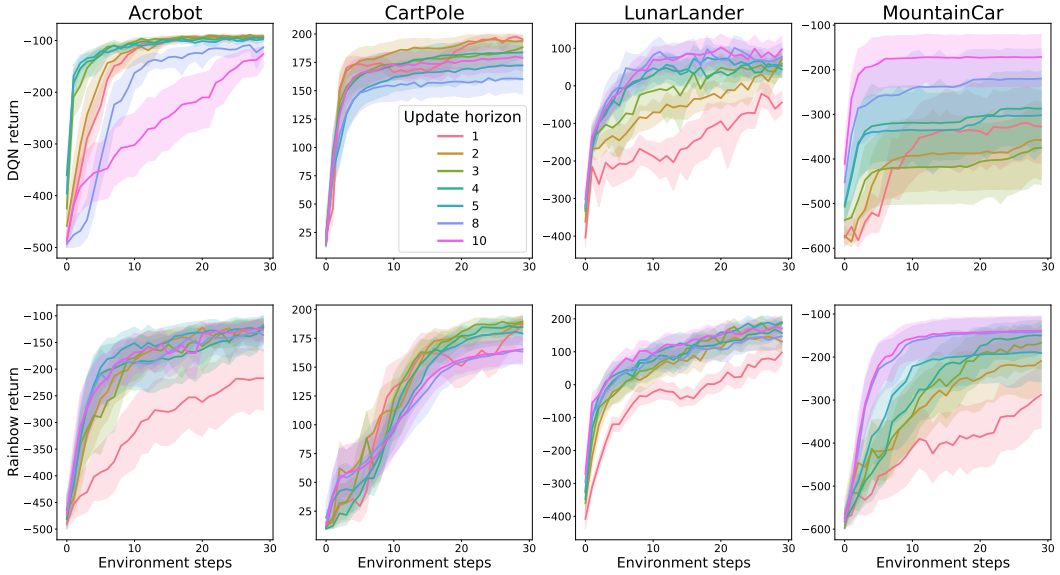

Figure 30: Comparison of update horizon on DQN (top) and Rainbow (bottom). The default value for DQN is 1, while for Rainbow it is 3.

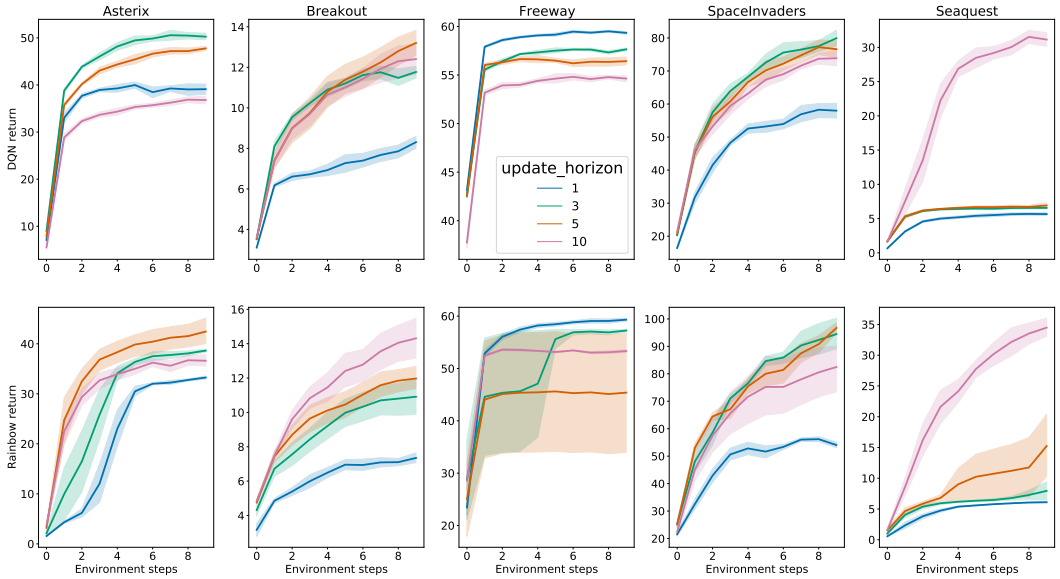

Figure 31: MinAtar comparison of update horizon on DQN (top) and Rainbow (bottom). The default value for DQN is 1, while for Rainbow it is 3.

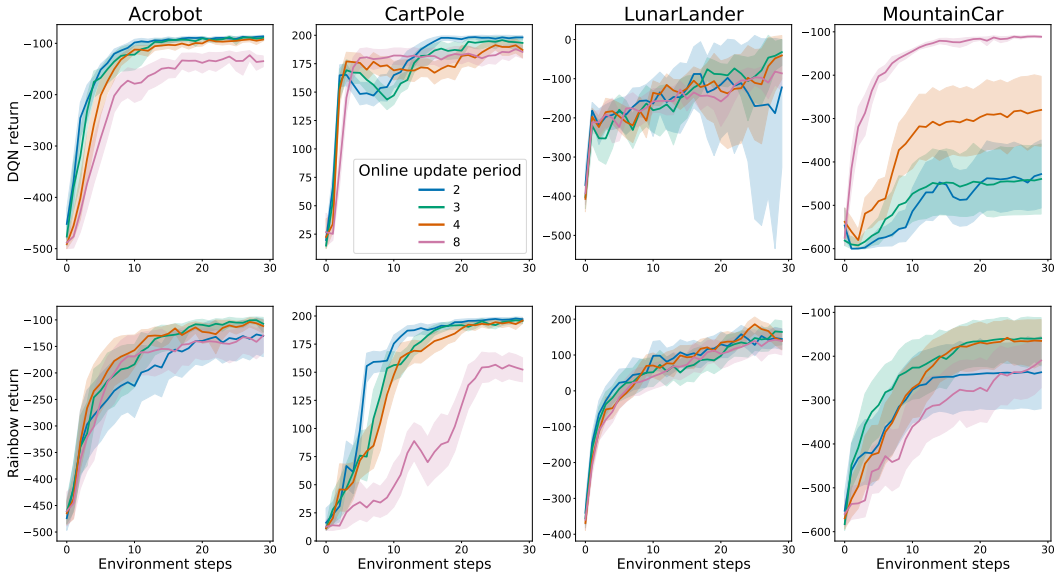

Figure 32: Comparison of online network update period on DQN (top) and Rainbow (bottom). The default value is 4.

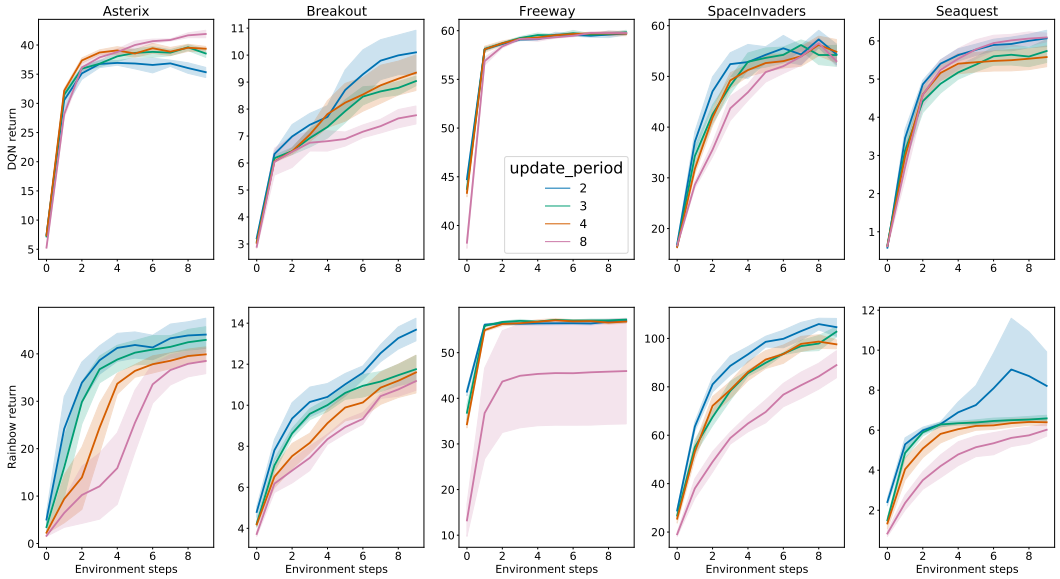

Figure 33: MinAtar comparison of online network update period on DQN (top) and Rainbow (bottom). The default value is 4.

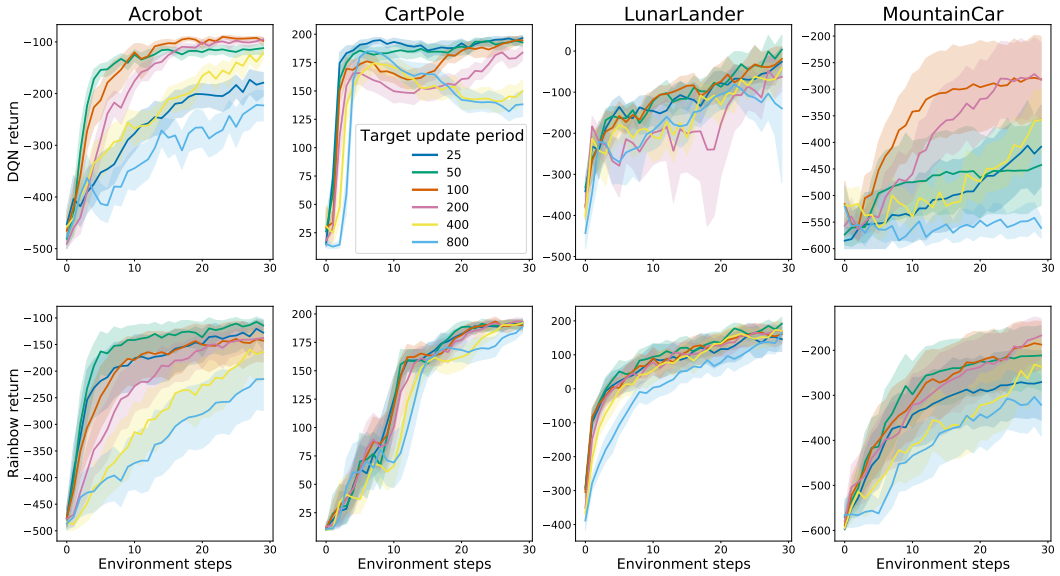

Figure 34: Comparison of target network update period on DQN (top) and Rainbow (bottom). The default value is 100.

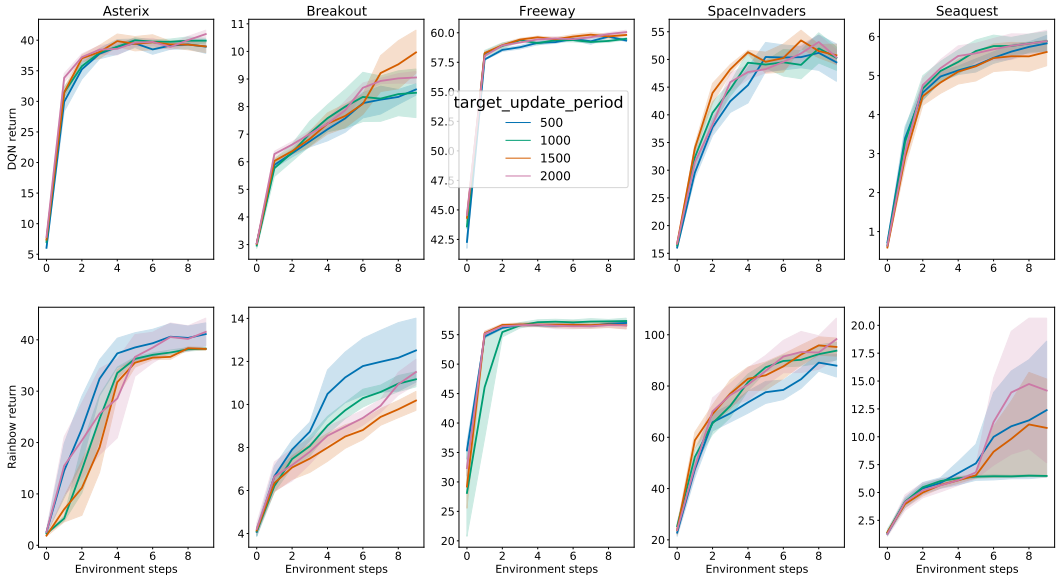

Figure 35: MinAtar comparison of target network update period on DQN (top) and Rainbow (bottom).

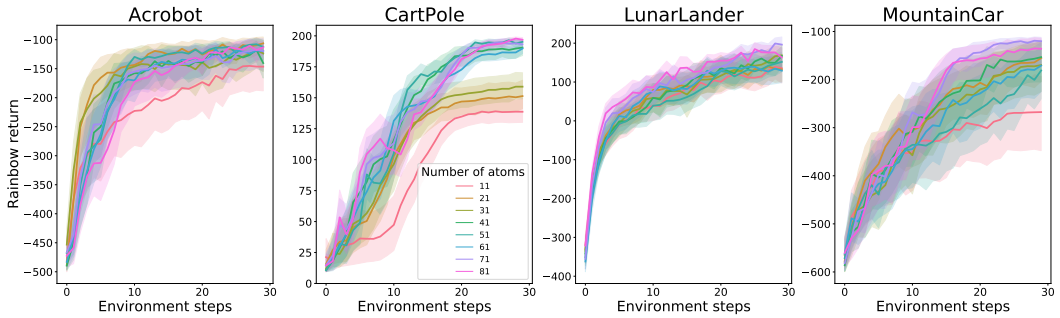

Figure 36: Comparison of number of atoms on Rainbow. The default value is 51.

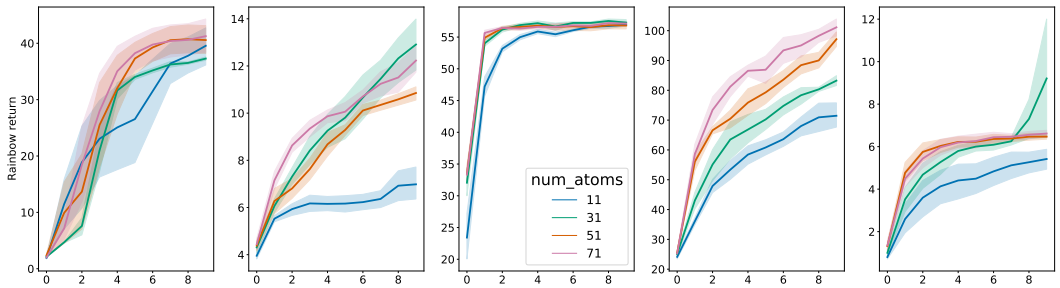

Figure 37: MinAtar comparison of number of atoms on Rainbow. The default value is 51.

B Best Hyperparameters settings for DQN and Rainbow accros the environments

Table 1 shows the best values for each hyperparameter across all the classic control environments and MinAtar games.

Table 1: Best Hyperparameters settings for DQN and Rainbow accros the environments

	Classic control envs		MinAtar	
Hyperparameter	DQN	Rainbow	DQN	Rainbow
gamma	0.99	0.99		
update horizon	4	10		
min replay history	1000	625		
update period	4	2		
target update period	100	50		
hidden layer	1	3	1	1
neurons	256	1024		
initialization	Orthogonal	He uniform	Orthogonal	Xavier uniform
activation function	Gelu	Selu		
normalization	LayerNorm	LayerNorm	Non-normalization	Batch-normalization
num atoms		71		71
clip	False	False		
learning rate	0.001	0.01		
eps	0.003125	3.125e-5	3.125e-5	0.0003125

C Hyperparameters settings for Classic control environments and MinAtar games

Table 2, 3, 4 list the choice configurations we used to run several experiments on classic control environments and MinAtar games.

Table 2: Hyperparameters settings for Classic control environments and MinAtar games

activation	init	learning_rate	epsilon	normalization
None activation	orthogonal	10	1	None
relu	zeros	5	0.5	BatchNorm
relu6	ones	2	0.3125	LayerNorm
sigmoid	xavier_uni	1	0.03125	
softplus	xavier_nor	0.1	0.003125	
soft_sign	lecun_uni	0.01	0.0003125	
silu	lecun_nor	0.001	3.125e-05	
swish	he_uni	0.0001	3.125e-06	
log_sigmoid	he_nor	1e-05		
hard_sigmoid	variance_baseline			
hard_silu	variance_0.1			
hard_swish	variance_0.3			
hard_tanh	variance_0.8			
elu	variance_3			
celu	variance_5			
selu	variance_10			
gelu				
glu				

Table 3: Hyperparameters settings for Classic control environments and MinAtar games

target_update_period	update_period	width	depth
10	1	32	1
25	2	64	2
50	3	128	3
100	4	256	4
200	8	512	
400	10	1024	
800	12		
1600			

Table 4: Hyperparameters settings for Classic control environments and MinAtar games

min_replay_history	update_horizon	gamma	num_atoms	clip_rewards
125	1	0.1	11	True
250	2	0.5	21	False
375	3	0.9	31	
500	4	0.99	41	
625	5	0.995	51	
750	8	0.999	61	
875	10			
1000				