A Study of Off-Policy Learning in Environments with Procedural Content Generation

Andy Ehrenberg, Robert Kirk
University College London

Minqi Jiang, Edward Grefenstette, Tim Rocktäschel
Meta AI, University College London

Abstract

Environments with procedural content generation (PCG environments) are useful for assessing the generalization capacity of Reinforcement Learning (RL) agents. A growing body of work focuses on generalization in RL in PCG environments, with many methods being built on top of *on-policy* algorithms. On the other hand, *off-policy* methods have received less attention. Motivated by this discrepancy, we examine how Deep Q Networks (Mnih et al., 2013) perform on the Procgen benchmark (Cobbe et al., 2020), and look at the impact of various additions to DQN on performance. We find that some popular techniques that have improved DQN on benchmarks like the Arcade Learning Environment (Bellemare et al., 2015, ALE) do not carry over to Procgen, implying that some research has overfit to tasks that lack diversity, and fails to consider the importance of generalization.

1 Introduction

Training general agents has become a central topic of RL research (Kirk et al., 2021). For RL algorithms to be applicable to real world problems, they must produce agents with the ability to handle diverse and changing conditions. Because of the variety in non-static environments, it is practically impossible to show all possible instances of such environments to agents during learning. Thus, we tackle the problem of *Generalization in Reinforcement Learning*, where algorithms must be developed that allow agents to succeed in situations not seen during training. Many environments built to evaluate the generalization capacity of algorithms use procedural content generation (Risi & Togelius, 2019) - which involves the algorithmic creation of random content such as game layouts, background colors, dynamics, and entities - to generate the set of environment instances.

Most of the algorithms in recent work that emphasizes performance on PCG environments are built on top of on-policy RL algorithms (Jiang et al., 2021; Raileanu et al., 2020; Raileanu & Fergus, 2021), while off-policy algorithms have received less attention. Off-policy algorithms possess the advantage of being able to update an agent's policy using older experiences in addition to recent ones, which often means that they can be more sample efficient than on-policy algorithms. Approaches that can leverage experience replay hold great promise for real-world applications, especially in situations where it may be costly or dangerous to collect many new trajectories. Understanding how off-policy algorithms perform in PCG environments can illuminate whether recent advances in off-policy algorithms have overfit to non-PCG environments, and what sorts of adaptations can be made to train agents with greater generalization capacity.

We focus our study on DQN (Mnih et al., 2013) due to its popularity, its ability to work well on environments with the pixel observations and discrete action spaces encountered in the Procgen benchmark.

Our main result is that some popular extensions to DQN like Prioritized Experience Replay (Schaul et al., 2016) and Dueling Networks (Wang et al., 2015) fail to improve performance on the Procgen benchmark, suggesting that research on off-policy methods has overfit to singleton environments (Whiteson et al., 2011). We also find that DQN and PPO experience similar patterns in generalization gaps, indicating that the two algorithms may struggle to generalize in similar ways. Finally, we find that combining data augmentation with distributional RL consistently improves DQN performance on the Procgen benchmark.

2 Motivation and Related Work

A large body of work exists on the problem of overfitting in deep RL, emphasizing the need for better generalization. Many benchmarks are insufficient for answering key questions about deep RL, including how to train more general agents. In response to this issue, several environments with procedural content generation have been proposed, including the Procgen benchmark (Cobbe et al., 2020), the NetHack Learning Environment (Kuttler et al., 2020), and MiniGrid (Chevalier-Boisvert et al., 2018). These environments provide a key step towards studying the important problem of generalization in RL.

Several studies on generalization in RL propose carry over methods typically used to prevent overfitting in deep supervised learning, such as dropout (Srivastava et al., 2014), ℓ_2 regularization (Nigam et al., 1999; Farebrother et al., 2018), and batch normalization (Ioffe & Szegedy, 2015). Other works study the impact of data augmentation on deep RL performance (Laskin et al., 2020; Kostrikov et al., 2021; Raileanu et al., 2020), finding that it can improve sample efficiency and enable robustness to environment changes. Representation learning techniques like learning to associate observations that are certain numbers of timesteps away (Stooke et al., 2021), and learning representations that assign similarity to states with close subsequent trajectories (Agarwal et al., 2021; Zhang et al., 2021; Mazoure et al., 2021) have also been shown to improve generalization. Many recent works on generalization in RL use the Procgen benchmark for evaluation (Jiang et al., 2021; Raileanu et al., 2020; Raileanu & Fergus, 2021; Mazoure et al., 2021), though they mostly build upon on-policy algorithms like PPO. For an in depth survey on generalization in RL, refer to Kirk et al. (2021).

Much less is known about how off-policy algorithms perform on environments where generalization is emphasized. Anand et al. (2021) analyzes MuZero's (Schrittwieser et al., 2020) performance on Procgen, and while MuZero does learn from off-policy experiences, the study focuses more on how planning and self-supervision influence generalization. Cobbe et al. (2020) includes an experiment comparing the performance of Rainbow and PPO on the Procgen benchmark when trained on the full level distribution, where Rainbow outperforms PPO on 6 out of the 16 games. We look at how additions to a baseline DQN implementation influence performance on the Procgen benchmark.

Popular extensions to DQN include Prioritized Experience Replay (Schaul et al., 2016), double Q learning (Hasselt et al., 2016), multi-step learning (Sutton & Barto, 2005), dueling networks (Wang et al., 2015), noisy nets (Fortunato et al., 2018) and distributional RL (Bellemare et al., 2017). These extensions are combined in Hessel et al. (2018) to form an integrated agent called Rainbow that achieved state-of-the-art performance on ALE, suggesting that these extensions can complement each other. An ablation study shows that PER, multi-step learning and distributional RL are more crucial than the other components for Atari performance. We ask if this is also true for Procgen.

3 Experiments

Many studies of RL in PCG environments examine the generalization capacity of agents. While progress has been made, few works include experiments with off-policy algorithms such as DQN. We fill some of this gap in the literature by studying the advantages and disadvantages of using DQN in PCG environments, and how various additions to DQN impact performance. We are guided by the following questions:

1. How do baseline versions of DQN and PPO compare on environments with procedural content generation in terms of test performance and generalization gaps?
2. How do various additions to DQN influence performance on procedurally generated environments?
3. What further changes can be made to DQN to improve performance on PCG environments?

We first compare baseline implementations of DQN and PPO (refer to Appendix A for implementation details)[1] to see if they have similar patterns of generalization gaps. Figure 1 shows that PPO generally outperforms a baseline DQN, with PPO getting better mean test scores on nine games, DQN doing better on four, and there being similar scores on three. In terms of average performance across all games, PPO gets 0.3 min-max normalized returns on test levels, while DQN gets

[1] Code available here

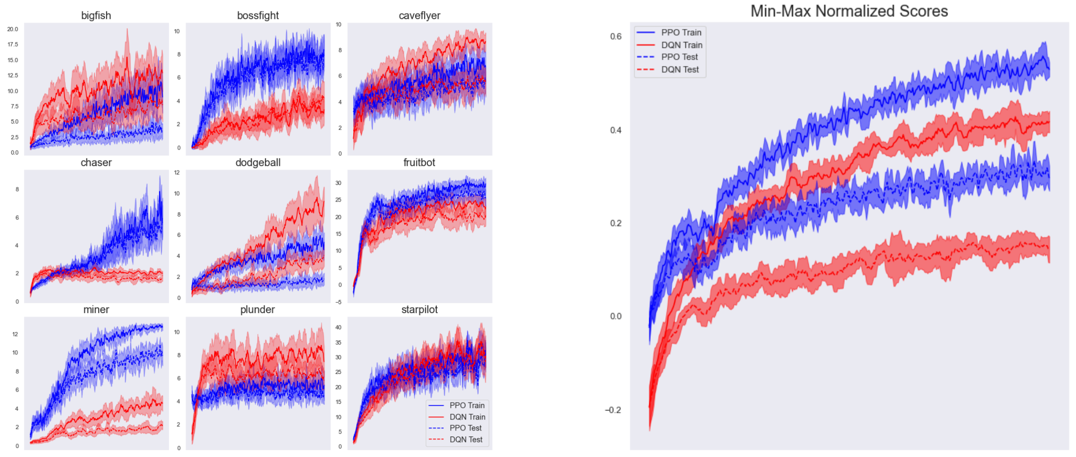

Figure 1: Left: Train and Test returns for PPO and DQN with 200 training levels and 25 million environment steps. Right: Min-max normalized scores across all games for PPO and DQN

0.15 after 25 million environment steps on 200 training levels on the easy mode of Procgen. The two algorithms tend to drop in performance by similar percentages from train to test levels, though DQN experiences a larger percent drop on 12 out of 16 games (Figure 5). The correlation between DQN's and PPO's percent drops is .8, indicating that the two algorithms may struggle to generalize in similar ways.

Our experiments indicate that DQN is more likely to outperform or score on par with PPO on games with fewer factors of variation that mainly control the spawn times and locations of other entities. These include games like Starpilot, Bigfish, Leaper and Plunder. DQN seems to generally receive test scores that are significantly worse than PPO on environments with procedural generation for mazes, level layouts, and goal locations. Games with these attributes include Coinrun, Maze, Miner and Chaser. Of course, this does not necessarily mean that DQN systematically handles the factors of variation for games like Starpilot better than those for games like Chaser, and further work should study this. Moreover, it should examine why different algorithms cope well with different forms of variation, perhaps through the development of environments where these can easily be turned on and off.

We observe that PER only has higher test performance than uniform sampling on one level, and sometimes degrades test scores (Figure 2). We validate this finding by confirming that our implementation gives the expected improvement on Atari environments and by experimenting with several hyperparameter settings. This corroborates (Raileanu & Fergus, 2021)'s idea that lower value errors can be symptomatic of aliasing and overfitting due to value functions being prone to learning spurious correlations in order to accurately estimate values. Value functions could use features that are irrelevant to the optimal policy like background colors to learn different values for states that are semantically identical, and given that DQN's policy directly comes from its value estimates, these spurious features for estimating values would in turn be spurious features for the agent's policy. Our findings are in stark contrast with the result from Hessel et al. (2018) that shows PER is one of the most helpful factors for the performance of Rainbow on Atari environments. We implement an off-policy variant of Prioritized Level Replay (Jiang et al., 2021) to see if this form of sampling is favorable above PER and uniform level, and get equivocal results. Further investigation should be done to improve upon our adaptation of PLR.

Of the extensions to DQN that we experiment with, only data augmentations and distributional RL (using quantile regression (Dabney et al., 2018) reliably improve performance, and combining them makes DQN much more competitive with our implementation of PPO, with roughly similar min-max normalized performance across games (Figure 3, Figure 9). Notably, in an ablation study, we

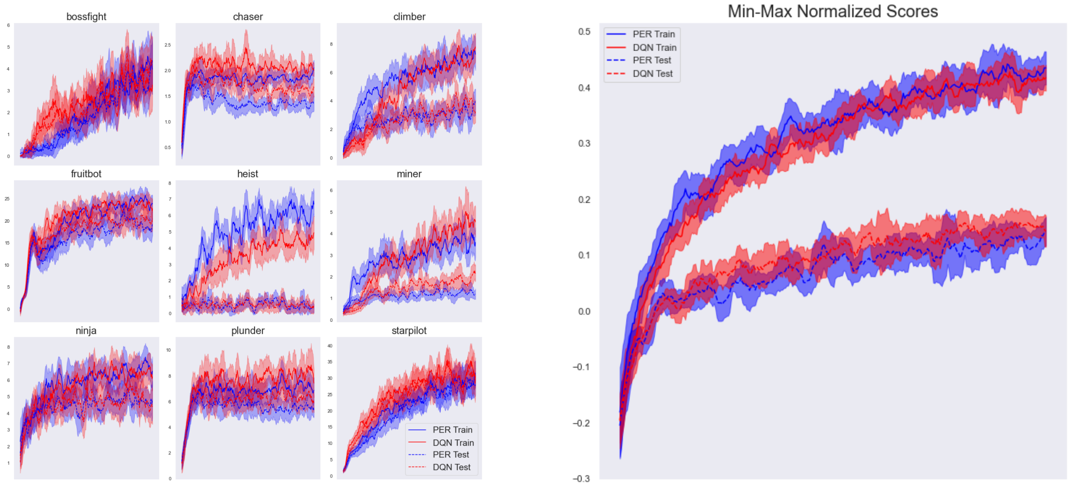

Figure 2: Left: Train and Test returns for PER and uniform sampling with 200 training levels and 25 million environment steps. Right: Min-max normalized scores across all games for PER and uniform sampling

observe that they generally perform on par with each other when not combined, but that large increases in performance occurs when they are put together (Figure 11). Further study on whether data augmentations enable learned value distributions to ignore spurious features could be informative. Refer to Appendix C for further experimental results including more games and other additions to DQN.

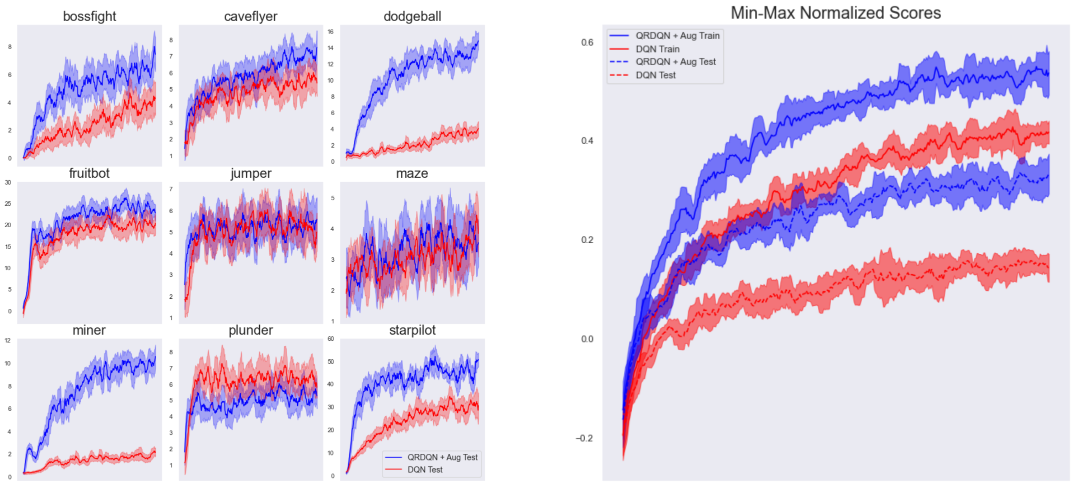

Figure 3: Left: Train and Test returns for QR-DQN trained using data augmentations and baseline DQN with 200 training levels and 25 million environment steps. Right: Min-max normalized scores across all games for QR-DQN with augmentations and baseline DQN

4 Conclusion

Here we have given evidence that commonly used additions to DQN may not aid test performance on environments with procedural content generation. This emphasizes the problem of evaluating agents on singleton environments - chasing the state of the art on benchmarks like ALE has resulted in algorithms that do not sufficiently consider generalization capacity. To make progress towards viability for real world applications of RL, diverse environments where generalization capacity is required to obtain high returns should be used. The comparisons with PPO illuminate either a deficiency in the potential of off-policy algorithms, or a failure to find the potential of off-policy algorithms that are more capable of zero-shot generalization. While some components do not aid DQN performance on Procgen, we motivate further investigation of off-policy RL algorithms in procedurally generated environments by showing that certain components can improve the test scores of DQN Procgen environments. Namely, we demonstrate that a simple combination of data augmentation and quantile regression for learning the value distribution gives promising results.

Our work naturally leads to several fascinating research directions that will bring agents closer to real world applications by understanding how agents can generalize - we discuss this in Appendix B. To give one example, evaluating agents on environments like MiniHack (Samvelyan et al., 2021), where different factors of variation can be set on and off, can help to isolate which sources of variability cause different agents to struggle the most. An improved grasp of where and why current approaches fail will serve as a great stepping stone for the next generation of RL algorithms.

References

Rishabh Agarwal, Marlos C. Machado, Pablo Samuel Castro, and Marc G. Bellemare. Contrastive behavioral similarity embeddings for generalization in reinforcement learning. *ArXiv*, abs/2101.05265, 2021.

Ankesh Anand, Jacob Walker, Yazhe Li, Eszter V'ertes, Julian Schrittwieser, Sherjil Ozair, Théophane Weber, and Jessica B. Hamrick. Procedural generalization by planning with self-supervised world models. *ArXiv*, abs/2111.01587, 2021.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents (extended abstract). In *IJCAI*, 2015.

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

Emmanuel Bengio, Joelle Pineau, and Doina Precup. Interference and generalization in temporal difference learning. In *ICML*, 2020.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning. *ArXiv*, abs/1912.01588, 2020.

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

Jesse Farebrother, Marlos C. Machado, and Michael H. Bowling. Generalization and regularization in dqn. *ArXiv*, abs/1810.00123, 2018.

Meire Fortunato, M. G. Azar, Bilal Piot, Jacob Menick, Ian Osband, A. Graves, Vlad Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg. Noisy networks for exploration. *ArXiv*, abs/1706.10295, 2018.

H. V. Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *ArXiv*, abs/1509.06461, 2016.

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.

Jacob Hilton, Nick Cammarata, Shan Carter, Gabriel Goh, and Chris Olah. Understanding rl vision. *Distill*, 2020. doi: 10.23915/distill.00029. https://distill.pub/2020/understanding-rl-vision.

S. Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. *ArXiv*, abs/2010.03934, 2021.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktaschel. A survey of generalisation in deep reinforcement learning. *ArXiv*, abs/2111.09794, 2021.

Ilya Kostrikov, Denis Yarats, and R. Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *ArXiv*, abs/2004.13649, 2021.

Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. *ArXiv*, abs/2010.14498, 2021.

Heinrich Kuttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. *ArXiv*, abs/2006.13760, 2020.

M. Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, P. Abbeel, and A. Srinivas. Reinforcement learning with augmented data. *ArXiv*, abs/2004.14990, 2020.

Bogdan Mazoure, Ahmed M. Ahmed, Patrick MacAlpine, R. Devon Hjelm, and Andrey Kolobov. Cross-trajectory representation learning for zero-shot generalization in rl. *ArXiv*, abs/2106.02193, 2021.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013.

K. Nigam, J. Lafferty, and A. McCallum. Using maximum entropy for text classification. 1999.

Roberta Raileanu and R. Fergus. Decoupling value and policy for generalization in reinforcement learning. *ArXiv*, abs/2102.10330, 2021.

Roberta Raileanu, Maxwell Goldstein, Denis Yarats, Ilya Kostrikov, and R. Fergus. Automatic data augmentation for generalization in deep reinforcement learning. *ArXiv*, abs/2006.12862, 2020.

Sebastian Risi and Julian Togelius. Increasing generality in machine learning through procedural content generation. *Nature Machine Intelligence*, pp. 1–9, 2019.

Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Kuttler, Edward Grefenstette, and Tim Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. URL `https://openreview.net/forum?id=skFwlyefkWJ`.

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

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, L. Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588 7839:604–609, 2020.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2014.

Nitish Srivastava, Geoffrey E. Hinton, A. Krizhevsky, Ilya Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15: 1929–1958, 2014.

Adam Stooke, Kimin Lee, P. Abbeel, and M. Laskin. Decoupling representation learning from reinforcement learning. *ArXiv*, abs/2009.08319, 2021.

R. Sutton and A. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 16:285–286, 2005.

Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015. URL `http://arxiv.org/abs/1511.06581`.

Shimon Whiteson, Brian Tanner, Matthew E. Taylor, and Peter Stone. Protecting against evaluation overfitting in empirical reinforcement learning. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, April 2011. URL `http://www.cs.utexas.edu/users/ai-lab?ADPRL11-shimon`.

A. Zhang, Rowan McAllister, R. Calandra, Y. Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. *ArXiv*, abs/2006.10742, 2021.

A IMPLEMENTATION DETAILS

We adapt DQN to work with the Procgen benchmark. To align with recent work on Procgen, we must allow DQN to work with parallel actor processes, each collecting experience on one of the training levels. We can specify which levels are available for training, and then either use random sampling or some other level replay strategy, such as sequential sampling or PLR (Jiang et al., 2021), to select levels for each actor. Many existing works on DQN use a single actor, so a few details need to change with regards to adding experience to a replay buffer. When using multi-step returns, a list of queues is used to track which trajectories contain enough transitions to add a multi-step transition to the replay buffer, and if multiple transitions are available to be added to the replay buffer, the buffer supports adding multiple transitions at once to improve speed.

With 64 processes, 64 transitions occur at each step, meaning that the replay buffer fills up more quickly, and that transitions exist in the buffer for less time. Existing DQN implementations commonly do a gradient update using experience replay at small intervals of steps in the environment (e.g. 4). In contrast, if experience replay is done after every iteration when there are 64 processes, this is equivalent to doing experience replay after 64 environment steps. We find that on Procgen, this doesn't create problems, as performance does not improve when experience replay is done more than 1 time for every 64 environment steps. Furthermore, in our experiments, using less processes never improves performance, and can sometimes hurt performance.

For the first 2000 steps (128000 total environment steps), actions are selected at random to warm up the replay buffer. For the first 8000 steps (512000 environment steps), the value of ϵ for ϵ-greedy exploration decreases from 1 to 0.1. We also track the effective rank of the first linear layer of the neural network to see if DQN experiences similar rank collapse on Procgen compared to the environments used for experiments in Kumar et al. (2021) - we observe some reduction in the effective rank, but do not see drops as severe as the original experiments on Atari and Gym environments.

Prioritized Experience Replay is implemented using proportional prioritization ($p_i = \delta_i + \epsilon$), with a sum tree data structure being used to increase speed - experiments using rank based prioritization yielded worse results. We also implement replay buffers that allow for data augmentation, and other non-uniform sampling strategies. For data augmentations, we follow the methods of Data Regularized Q (DrQ) (Kostrikov et al., 2021). Namely, when a transition is sampled for experience replay, the state, next state, and copies of each of them are augmented using a random crop (after being padded to result in the same dimension). Bootstrap target Q values are constructed for each augmented version of the next state, and their values are averaged. Q value estimates are then calculated for each augmented state, and the loss is calculated for each estimate.

B FUTURE WORK

In light of our findings, we hope that future works on off-policy reinforcement learning emphasize performance on environments that necessitate good generalization ability. This work also motivates future study focusing on improving our understanding of how differences in learning dynamics between on-policy and off-policy algorithms influence differences in generalization capacity. Recent studies on the relationship between TD-learning, bootstrapping, and generalization Bengio et al. (2020) serve as a good example for this.

Table 1: DQN Hyperparameters

Hyperparameter	Value
State shape	$(64, 64, 3)$
Max env steps	25×10^6
Replay buffer size	1×10^6
Num processes	64
Warm up steps	128000
Batch size	512
Optimizer	Adam
Learning rate	2.5×10^{-4}
Env steps per update	128
Discount γ	0.99
Target update freq	32000
Hidden units for MLP	512
Initial epsilon	1
End epsilon	0.1
Epsilon decay period	512000

Table 2: PPO Hyperparameters

Hyperparameter	Value
State shape	$(64, 64, 3)$
Max env steps	25×10^6
Num processes	64
Optimizer	RMSProp
Learning rate	5×10^{-4}
Env steps per update	128
Discount γ	0.999
GAE λ	0.95
Entropy coef	0.01
Value loss coef	0.5
Return normalization	True
A2C Forward Steps	256
Num PPO epochs	3
Num PPO minibatches	8
Gradient clip	0.2

An exciting direction for research would be to isolate which kinds of factors of variation cause on-policy and/or off-policy algorithms to struggle the most. Designing environments where the user can turn on and off factors of variation would make this sort of study viable. Further work in this area could help to either corroborate or contradict our observations regarding the kinds of procedural generation that favor DQN or PPO. Also, applying interpretability techniques like attribution (Simonyan et al., 2014) may illuminate differences in the features learned by these algorithms - reproducing the results of Hilton et al. (2020) and transferring its methods to DQN could be a good starting place.

Further research can more closely examine extensions to DQN. For example, with QR-DQN, it may be interesting to look further into the learned distributions for different states across different environments. We observe that in environments where QR-DQN significantly improves performance, the predicted value distribution usually resembles a Gaussian distribution. Perhaps learned distributions could look more smooth or more multi-modal according to different levels of diversity encountered during training. Other work can examine why PER may not be helpful on PCG environments. Our initial experiments looking at correlations between a level's sampling rate and the agent's reward on the level were not highly informative. Focusing too much on a subset of levels could hurt generalization, although work like Prioritized Level Replay (PLR) (Jiang et al., 2021) shows that updating more on different levels at different stages of training can help generalization. We adapt PLR for DQN and find that in some cases it can help test returns on some Procgen games,

Table 3: QR-DQN Hyperparameters

Hyperparameter	Value
State shape	$(64, 64, 3)$
Max env steps	25×10^6
Replay buffer size	1×10^6
Num processes	64
Warm up steps	128000
Batch size	512
Optimizer	Adam
Learning rate	2.5×10^{-4}
Env steps per update	128
Discount γ	0.99
Target update freq	32000
Hidden units for MLP	512
Initial epsilon	1
End epsilon	0.1
Epsilon decay period	512000
Num quantiles	200

Table 4: PER Hyperparameters

Hyperparameter	Value
Alpha α	0.5
Beta β	0.4
Min priority	0.01
Type	Proportional

though can be detrimental on others. Further work on adapting techniques originally built on top of on-policy algorithms to off-policy algorithms could illustrate the differences between these two methods.

C Other Figures

Game	Our Winner	Winner for Cobbe et al.	Match?
Bigfish	DQN	Rainbow	Yes
Bossfight	PPO	Close (near tie)	Yes
Caveflyer	Close (PPO)	DQN	No
Chaser	PPO	PPO	Yes
Climber	PPO	PPO	Yes
Coinrun	PPO	PPO	Yes
Dodgeball	DQN	Rainbow	Yes
Fruitbot	PPO	PPO	Yes
Heist	PPO	PPO	Yes
Jumper	Close (PPO)	Rainbow	No
Leaper	Close (DQN)	PPO	No
Maze	PPO	PPO	Yes
Miner	PPO	PPO	Yes
Ninja	PPO	PPO	Yes
Plunder	DQN	PPO	No
Starpilot	Close (DQN)	Rainbow	Yes

Table 5: Comparison between our PPO vs DQN experiment and the PPO vs Rainbow experiment from Cobbe et al. (2020)

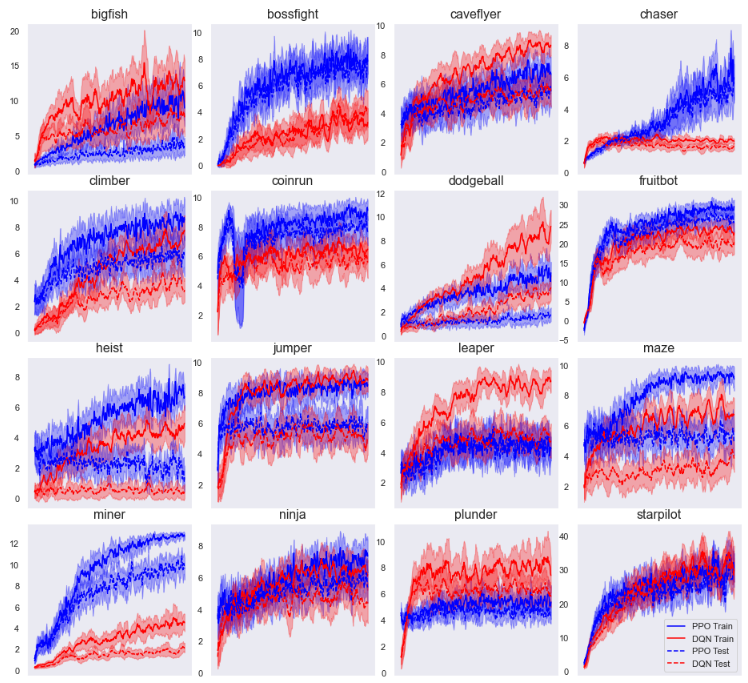

Figure 4: Train and test level returns for PPO vs DQN (200 training levels, 25 million environment steps)

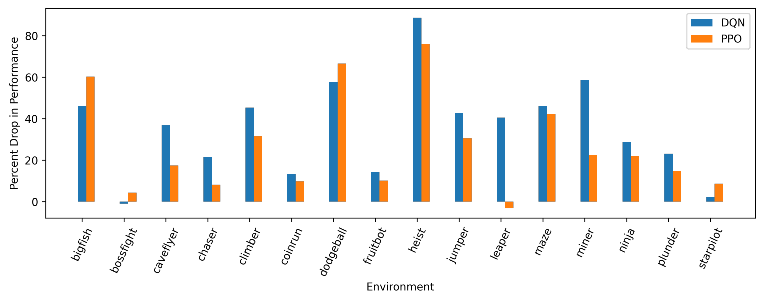

Figure 5: Percent drops from train to test level scores for PPO and DQN

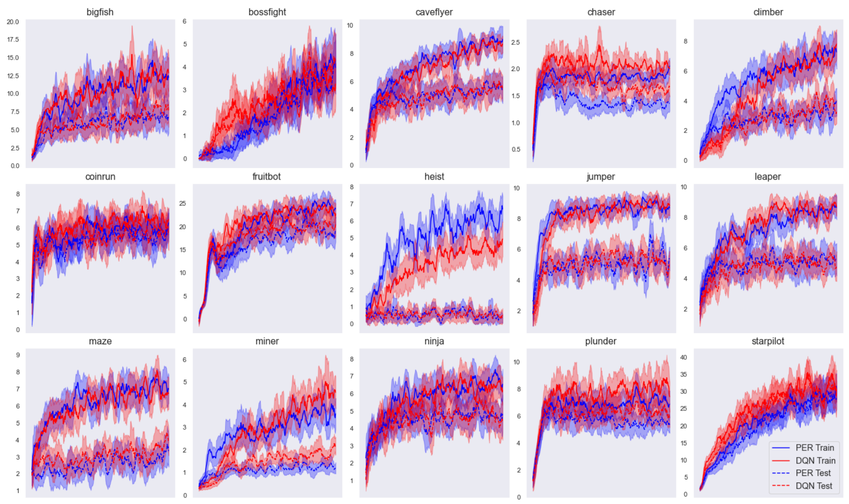

Figure 6: PER vs Uniform Sampling (200 training levels, 25 million environment steps)

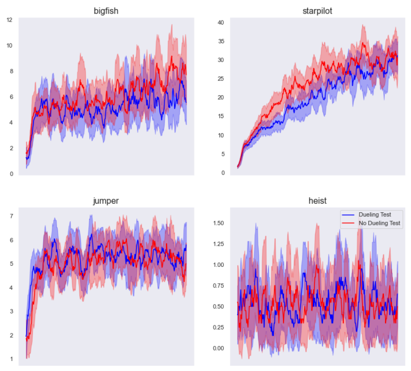

Figure 7: Test returns for dueling architecture vs baseline DQN (200 training levels, 25 million environment steps)

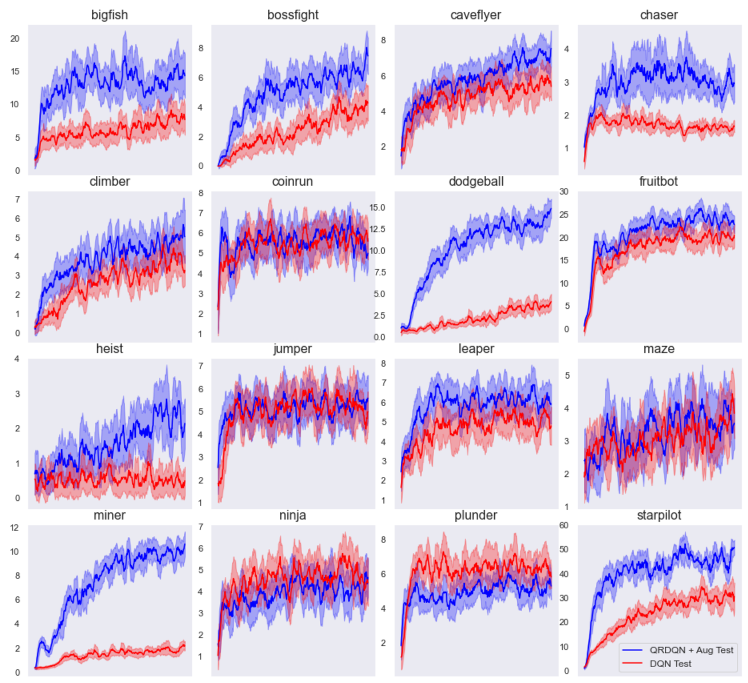

Figure 8: QR-DQN with Data Augmentations vs Baseline DQN (200 training levels, 25 million environment steps)

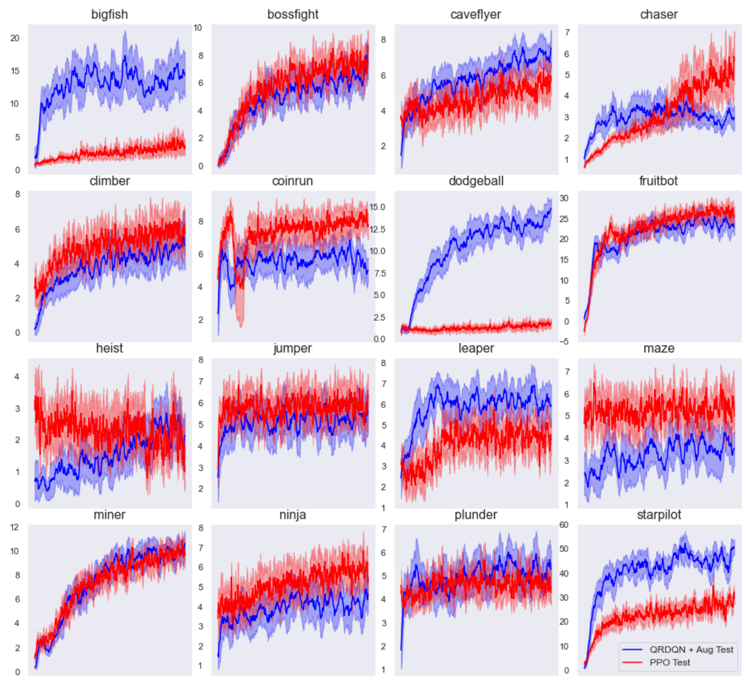

Figure 9: QR-DQN with Data Augmentations vs PPO (200 training levels, 25 million environment steps)

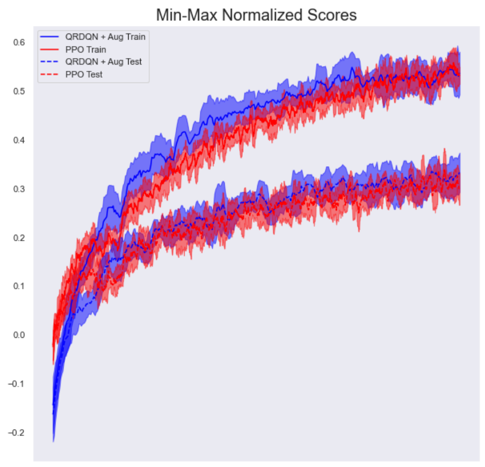

Figure 10: Min-max normalized scores across all environments for QR-DQN with Data Augmentations vs PPO (200 training levels, 25 million environment steps)

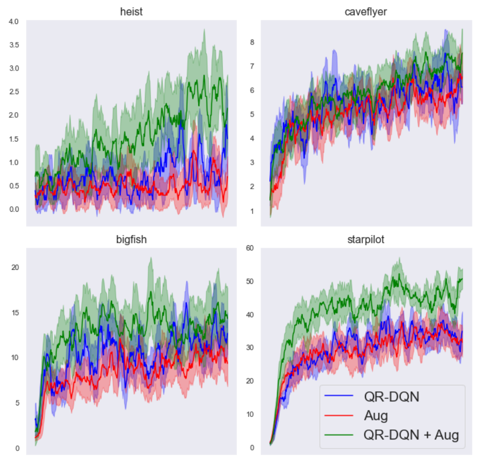

Figure 11: Ablations on distributional RL and data augmentations