Quantifying Generalization in Reinforcement Learning

Karl Cobbe [1] **Oleg Klimov** [1] **Chris Hesse** [1] **Taehoon Kim** [1] **John Schulman** [1]

Abstract

In this paper, we investigate the problem of overfitting in deep reinforcement learning. Among the most common benchmarks in RL, it is customary to use the same environments for both training and testing. This practice offers relatively little insight into an agent's ability to generalize. We address this issue by using procedurally generated environments to construct distinct training and test sets. Most notably, we introduce a new environment called CoinRun, designed as a benchmark for generalization in RL. Using CoinRun, we find that agents overfit to surprisingly large training sets. We then show that deeper convolutional architectures improve generalization, as do methods traditionally found in supervised learning, including L2 regularization, dropout, data augmentation and batch normalization.

1. Introduction

Generalizing between tasks remains difficult for state of the art deep reinforcement learning (RL) algorithms. Although trained agents can solve complex tasks, they struggle to transfer their experience to new environments. Agents that have mastered ten levels in a video game often fail catastrophically when first encountering the eleventh. Humans can seamlessly generalize across such similar tasks, but this ability is largely absent in RL agents. In short, agents become overly specialized to the environments encountered during training.

That RL agents are prone to overfitting is widely appreciated, yet the most common RL benchmarks still encourage training and evaluating on the same set of environments. We believe there is a need for more metrics that evaluate generalization by explicitly separating training and test environments. In the same spirit as the Sonic Benchmark (Nichol et al., 2018), we seek to better quantify an agent's ability to generalize.

To begin, we train agents on CoinRun, a procedurally generated environment of our own design, and we report the surprising extent to which overfitting occurs. Using this environment, we investigate how several key algorithmic and architectural decisions impact the generalization performance of trained agents.

The main contributions of this work are as follows:

1. We show that the number of training environments required for good generalization is much larger than the number used by prior work on transfer in RL.
2. We propose a generalization metric using the CoinRun environment, and we show how this metric provides a useful signal upon which to iterate.
3. We evaluate the impact of different convolutional architectures and forms of regularization, finding that these choices can significantly improve generalization performance.

2. Related Work

Our work is most directly inspired by the Sonic Benchmark (Nichol et al., 2018), which proposes to measure generalization performance by training and testing RL agents on distinct sets of levels in the *Sonic the Hedgehog*™ video game franchise. Agents may train arbitrarily long on the training set, but are permitted only 1 million timesteps at test time to perform fine-tuning. This benchmark was designed to address the problems inherent to "training on the test set."

(Farebrother et al., 2018) also address this problem, accurately recognizing that conflating train and test environments has contributed to the lack of regularization in deep RL. They propose using different game modes of Atari 2600 games to measure generalization. They turn to supervised learning for inspiration, finding that both L2 regularization and dropout can help agents learn more generalizable features.

(Packer et al., 2018) propose a different benchmark to measure generalization using six classic environments, each of

[1]OpenAI, San Francisco, CA, USA. Correspondence to: Karl Cobbe <karl@openai.com>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

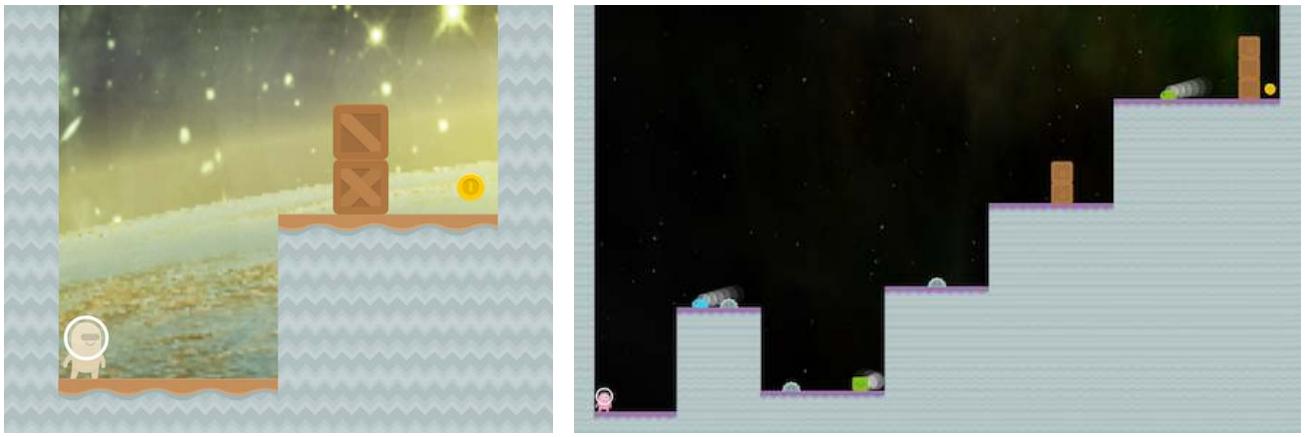


Figure 1. Two levels in CoinRun. The level on the left is much easier than the level on the right.

which has been modified to expose several internal parameters. By training and testing on environments with different parameter ranges, their benchmark quantifies agents' ability to interpolate and extrapolate. (Zhang et al., 2018a) measure overfitting in continuous domains, finding that generalization improves as the number of training seeds increases. They also use randomized rewards to determine the extent of undesirable memorization.

Other works create distinct train and test environments using procedural generation. (Justesen et al., 2018) use the General Video Game AI (GVG-AI) framework to generate levels from several unique games. By varying difficulty settings between train and test levels, they find that RL agents regularly overfit to a particular training distribution. They further show that the ability to generalize to human-designed levels strongly depends on the level generators used during training.

(Zhang et al., 2018b) conduct experiments on procedurally generated gridworld mazes, reporting many insightful conclusions on the nature of overfitting in RL agents. They find that agents have a high capacity to memorize specific levels in a given training set, and that techniques intended to mitigate overfitting in RL, including sticky actions (Machado et al., 2018) and random starts (Hausknecht and Stone, 2015), often fail to do so.

In Section 5.4, we similarly investigate how injecting stochasticity impacts generalization. Our work mirrors (Zhang et al., 2018b) in quantifying the relationship between overfitting and the number of training environments, though we additionally show how several methods, including some more prevalent in supervised learning, can reduce overfitting in our benchmark.

These works, as well as our own, highlight the growing need for experimental protocols that directly address generalization in RL.

3. Quantifying Generalization

3.1. The CoinRun Environment

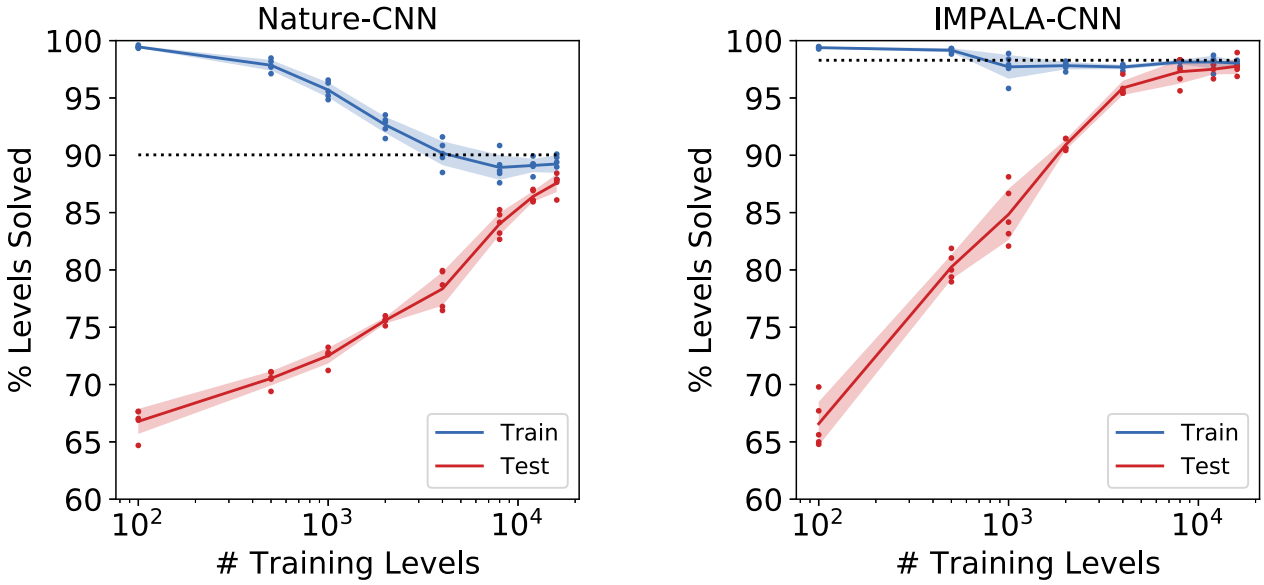
We propose the CoinRun environment to evaluate the generalization performance of trained agents. The goal of each CoinRun level is simple: collect the single coin that lies at the end of the level. The agent controls a character that spawns on the far left, and the coin spawns on the far right. Several obstacles, both stationary and non-stationary, lie between the agent and the coin. A collision with an obstacle results in the agent's immediate death. The only reward in the environment is obtained by collecting the coin, and this reward is a fixed positive constant. The level terminates when the agent dies, the coin is collected, or after 1000 time steps.

We designed the game CoinRun to be tractable for existing algorithms. That is, given a sufficient number of training levels and sufficient training time, our algorithms learn a near optimal policy for *all* CoinRun levels. Each level is generated deterministically from a given seed, providing agents access to an arbitrarily large and easily quantifiable supply of training data. CoinRun mimics the style of platformer games like Sonic, but it is much simpler. For the purpose of evaluating generalization, this simplicity can be highly advantageous.

Levels vary widely in difficulty, so the distribution of levels naturally forms a curriculum for the agent. Two different levels are shown in Figure 1. See Appendix A for more details about the environment and Appendix B for additional screenshots. Videos of a trained agent playing can be found here, and environment code can be found here.

3.2. CoinRun Generalization Curves

Using the CoinRun environment, we can measure how successfully agents generalize from a given set of training lev-



(a) Final train and test performance of Nature-CNN agents after 256M timesteps, as a function of the number of training levels.

(b) Final train and test performance of IMPALA-CNN agents after 256M timesteps, as a function of number of training levels.

Figure 2. Dotted lines denote final mean test performance of the agents trained with an unbounded set of levels. The solid line and shaded regions represent the mean and standard deviation respectively across 5 seeds. Training sets are generated separately for each seed.

els to an unseen set of test levels. Train and test levels are drawn from the same distribution, so the gap between train and test performance determines the extent of overfitting. As the number of available training levels grows, we expect the performance on the test set to improve, even when agents are trained for a fixed number of timesteps. At test time, we measure the zero-shot performance of each agent on the test set, applying no fine-tuning to the agent's parameters.

We train 9 agents to play CoinRun, each on a training set with a different number of levels. During training, each new episode uniformly samples a level from the appropriate set. The first 8 agents are trained on sets ranging from of 100 to 16,000 levels. We train the final agent on an unbounded set of levels, where each level is seeded randomly. With 2^{32} level seeds, collisions are unlikely. Although this agent encounters approximately 2M unique levels during training, it still does not encounter any test levels until test time. We repeat this whole experiment 5 times, regenerating the training sets each time.

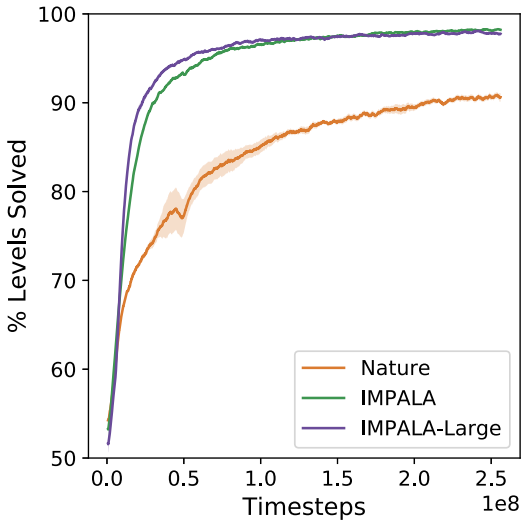
We first train agents with policies using the same 3-layer convolutional architecture proposed by (Mnih et al., 2015), which we henceforth call Nature-CNN. Agents are trained with Proximal Policy Optimization (Schulman et al., 2017; Dhariwal et al., 2017) for a total of 256M timesteps across 8 workers. We train agents for the same number of timesteps independent of the number of levels in the training set. We average gradients across all 8 workers on each mini-batch. We use $\gamma = .999$, as an optimal agent takes between 50 and 500 timesteps to solve a level, depending on level difficulty. See Appendix D for a full list of hyperparameters.

Results are shown in Figure 2a. We collect each data point by averaging the final agent's performance across 10,000 episodes, where each episode samples a level from the appropriate set. We can see that substantial overfitting occurs when there are less than 4,000 training levels. Even with 16,000 training levels, overfitting is still noticeable. Agents perform best when trained on an unbounded set of levels, when a new level is encountered in every episode. See Appendix E for performance details.

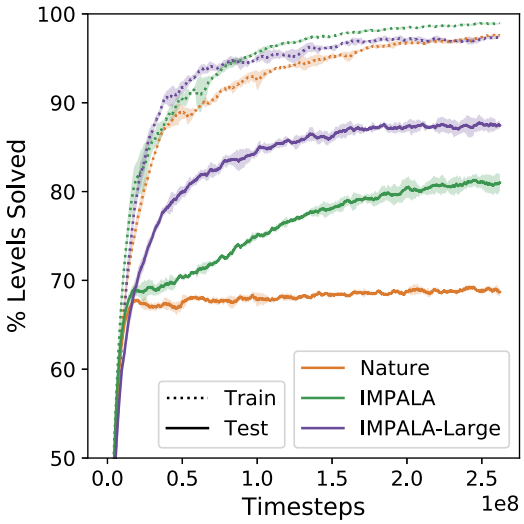
Now that we have generalization curves for the baseline architecture, we can evaluate the impact of various algorithmic and architectural decisions.

4. Evaluating Architectures

We choose to compare the convolutional architecture used in IMPALA (Espeholt et al., 2018) against our Nature-CNN baseline. With the IMPALA-CNN, we perform the same experiments described in Section 3.2, with results shown in Figure 2b. We can see that across all training sets, the IMPALA-CNN agents perform better at test time than



(a) Performance of Nature-CNN and IMPALA-CNN agents during training, on an unbounded set of training levels.



(b) Performance of Nature-CNN and IMPALA-CNN agents during training, on a set of 500 training levels.

Figure 3. The lines and shaded regions represent the mean and standard deviation respectively across 3 runs.

Nature-CNN agents.

To evaluate generalization performance, one could train agents on the unbounded level set and directly compare learning curves. In this setting, it is impossible for an agent to overfit to any subset of levels. Since every level is new, the agent is evaluated on its ability to continually generalize. For this reason, performance with an unbounded training set can serve as a reasonable proxy for the more explicit train-to-test generalization performance. Figure 3a shows a comparison between training curves for IMPALA-CNN and Nature-CNN, with an unbounded set of training levels. As we can see, the IMPALA-CNN architecture is substantially more sample efficient.

However, it is important to note that learning faster with an unbounded training set will not always correlate positively with better generalization performance. In particular, well chosen hyperparameters might lead to improved training speed, but they are less likely to lead to improved generalization. We believe that directly evaluating generalization, by training on a fixed set of levels, produces the most useful metric. Figure 3b shows the performance of different architectures when training on a fixed set of 500 levels. The same training set is used across seeds.

In both settings, it is clear that the IMPALA-CNN architecture is better at generalizing across levels of CoinRun. Given the success of the IMPALA-CNN, we experimented with several larger architectures, finding a deeper and wider variant of the IMPALA architecture (IMPALA-Large) that performs even better. This architecture uses 5 residual blocks instead of 3, with twice as many channels at each layer. Results with this architecture are shown in Figure 3.

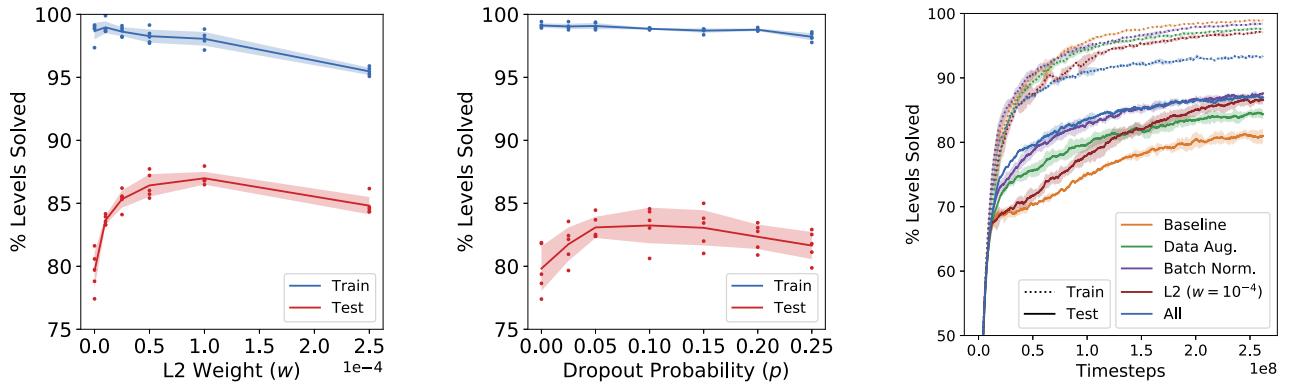
It is likely that further architectural tuning could yield even greater generalization performance. As is common in supervised learning, we expect much larger networks to have a higher capacity for generalization. In our experiments, however, we noticed diminishing returns increasing the network size beyond IMPALA-Large, particularly as wall clock training time can dramatically increase. In any case, we leave further architectural investigation to future work.

5. Evaluating Regularization

Regularization has long played an significant role in supervised learning, where generalization is a more immediate concern. Datasets always include separate training and test sets, and there are several well established regularization techniques for reducing the generalization gap. These regularization techniques are less often employed in deep RL, presumably because they offer no perceivable benefits in the absence of a generalization gap – that is, when the training and test sets are one and the same.

Now that we are directly measuring generalization in RL, we have reason to believe that regularization will once again prove effective. Taking inspiration from supervised learning, we choose to investigate the impact of L2 regularization, dropout, data augmentation, and batch normalization in the CoinRun environment.

Throughout this section we train agents on a fixed set of 500 CoinRun levels, following the same experimental procedure shown in Figure 3b. We have already seen that sub-



(a) Final train and test performance after 256M timesteps as a function of the L2 weight penalty. Mean and standard deviation is shown across 5 runs.

(b) Final train and test performance after 512M timesteps as a function of the dropout probability. Mean and standard deviation is shown across 5 runs.

(c) The effect of using data augmentation, batch normalization and L2 regularization when training on 500 levels. Mean and standard deviation is shown across 3 runs.

Figure 4. The impact of different forms of regularization.

stantial overfitting occurs, so we expect this setting to provide a useful signal for evaluating generalization. In all subsequent experiments, figures show the mean and standard deviation across 3-5 runs. In these experiments, we use the original IMPALA-CNN architecture with 3 residual blocks, but we notice qualitatively similar results with other architectures.

5.1. Dropout and L2 Regularization

We first train agents with either dropout probability $p \in [0, 0.25]$ or with L2 penalty $w \in [0, 2.5 \times 10^{-4}]$. We train agents with L2 regularization for 256M timesteps, and we train agents with dropout for 512M timesteps. We do this since agents trained with dropout take longer to converge. We report both the final train and test performance. The results of these experiments are shown in Figure 4. Both L2 regularization and dropout noticeably reduce the generalization gap, though dropout has a smaller impact. Empirically, the most effective dropout probability is $p = 0.1$ and the most effective L2 weight is $w = 10^{-4}$.

5.2. Data Augmentation

Data augmentation is often effective at reducing overfitting on supervised learning benchmarks. There have been a wide variety of augmentation transformations proposed for images, including translations, rotations, and adjustments to brightness, contrast, or sharpness. (Cubuk et al., 2018) search over a diverse space of augmentations and train a policy to output effective data augmentations for a target dataset, finding that different datasets often benefit from different sets of augmentations.

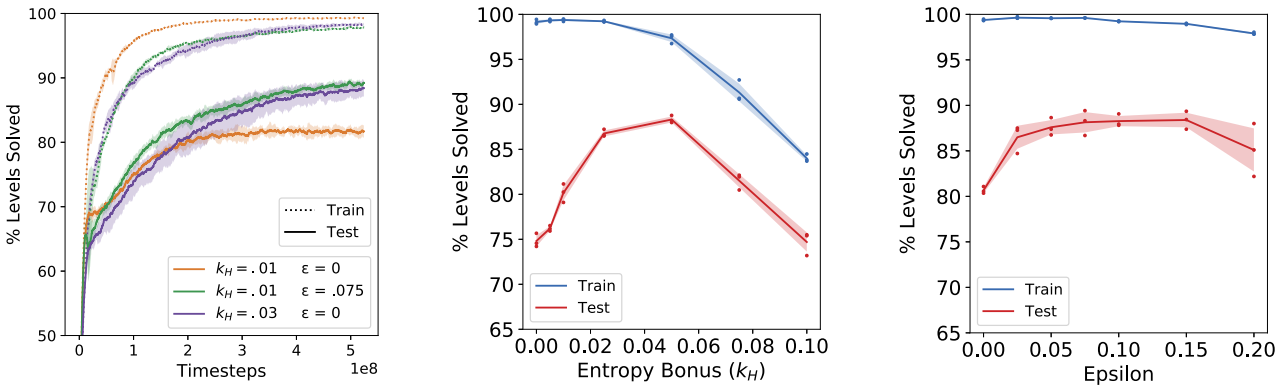
We take a simple approach in our experiments, using a slightly modified form of Cutout (Devries and Taylor, 2017). For each observation, multiple rectangular regions of varying sizes are masked, and these masked regions are assigned a random color. See Appendix C for screenshots. This method closely resembles domain randomization (Tobin et al., 2017), used in robotics to transfer from simulations to the real world. Figure 4c shows the boost this data augmentation scheme provides in CoinRun. We expect that other methods of data augmentation would prove similarly effective and that the effectiveness of any given augmentation will vary across environments.

5.3. Batch Normalization

Batch normalization (Ioffe and Szegedy, 2015) is known to have a substantial regularizing effect in supervised learning (Luo et al., 2018). We investigate the impact of batch normalization on generalization, by augmenting the IMPALA-CNN architecture with batch normalization after every convolutional layer. Training workers normalize with the statistics of the current batch, and test workers normalize with a moving average of these statistics. We show the comparison to baseline generalization in Figure 4c. As we can see, batch normalization offers a significant performance boost.

5.4. Stochasticity

We now evaluate the impact of stochasticity on generalization in CoinRun. We consider two methods, one varying the environment's stochasticity and one varying the policy's stochasticity. First, we inject environmental stochasticity by following ϵ-greedy action selection: with probability ϵ at each timestep, we override the agent's preferred action with a random action. In previous work, ϵ-greedy ac-



(a) Comparison of ϵ-greedy and high entropy bonus agents to baseline during training.

(b) Final train and test performance for agents trained with different entropy bonuses.

(c) Final train and test performance for ϵ-greedy agents trained with different values of ϵ.

Figure 5. The impact of introducing stochasticity into the environment, via epsilon-greedy action selection and an entropy bonus. Training occurs over 512M timesteps. Mean and standard deviation is shown across 3 runs.

tion selection has been used both as a means to encourage exploration and as a theoretical safeguard against overfitting (Bellemare et al., 2012; Mnih et al., 2013). Second, we control policy stochasticity by changing the entropy bonus in PPO. Note that our baseline agent already uses an entropy bonus of $k_H = .01$.

We increase training time to 512M timesteps as training now proceeds more slowly. Results are shown in Figure 5. It is clear that an increase in either the environment's or the policy's stochasticity can improve generalization. Furthermore, each method in isolation offers a similar generalization boost. It is notable that training with increased stochasticity improves generalization to a greater extent than any of the previously mentioned regularization methods. In general, we expect the impact of these stochastic methods to vary substantially between environments; we would expect less of a boost in environments whose dynamics are already highly stochastic.

5.5. Combining Regularization Methods

We briefly investigate the effects of combining several of the aforementioned techniques. Results are shown in Figure 4c. We find that combining data augmentation, batch normalization, and L2 regularization yields slightly better test time performance than using any one of them individually. However, the small magnitude of the effect suggests that these regularization methods are perhaps addressing similar underlying causes of poor generalization. Furthermore, for unknown reasons, we had little success combining ϵ-greedy action selection and high entropy bonuses with other forms of regularization.

6. Additional Environments

The preceding sections have revealed the high degree overfitting present in one particular environment. We corroborate these results by quantifying overfitting on two additional environments: a CoinRun variant called CoinRun-Platforms and a simple maze navigation environment called RandomMazes.

We apply the same experimental procedure described in Section 3.2 to both CoinRun-Platforms and RandomMazes, to determine the extent of overfitting. We use the original IMPALA-CNN architecture followed by an LSTM (Hochreiter and Schmidhuber, 1997), as memory is necessary for the agent to explore optimally. These experiments further reveal how susceptible our algorithms are to overfitting.

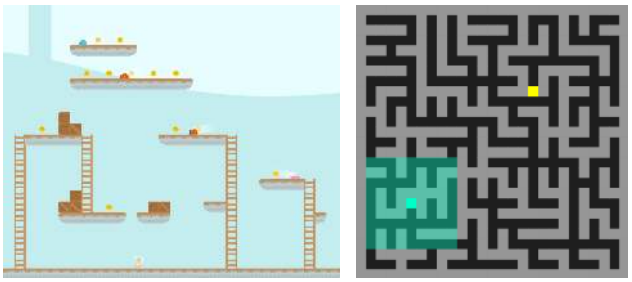


Figure 6. Levels from CoinRun-Platforms (left) and RandomMazes (right). In RandomMazes, the agent's observation space is shaded in green.

6.1. CoinRun-Platforms

In CoinRun-Platforms, there are several coins that the agent attempts to collect within the 1000 step time-limit. Coins

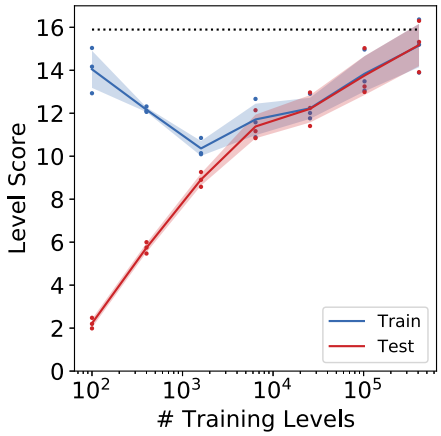


Figure 7. Final train and test performance in CoinRun-Platforms after 2B timesteps, as a function of the number of training levels.

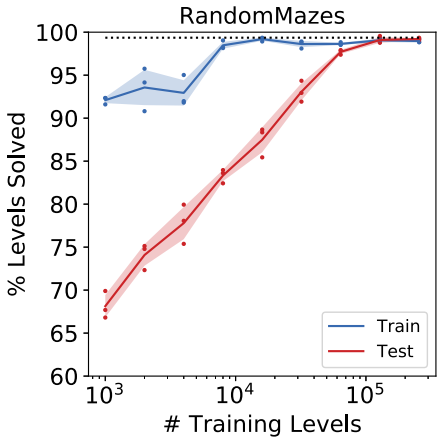


Figure 8. Final train and test performance in RandomMazes after 256M timesteps, as a function of the number of training levels.

are randomly scattered across platforms in the level. Levels are a larger than in CoinRun, so the agent must actively explore, sometimes retracing its steps. Collecting any coin gives a reward of 1, and collecting all coins in a level gives an additional reward of 9. Each level contains several moving monsters that the agent must avoid. The episode ends only when all coins are collected, when time runs out, or when the agent dies. See Appendix B for environment screenshots.

As CoinRun-Platforms is a much harder game, we train each agent for a total of 2B timesteps. Figure 7 shows that overfitting occurs up to around 4000 training levels. Beyond the extent of overfitting, it is also surprising that agents' training performance increases as a function of the number of training levels, past a certain threshold. This is notably different from supervised learning, where training performance generally decreases as the training set becomes larger. We attribute this trend to the implicit curriculum in the distribution of generated levels. With additional training data, agents are more likely to learn skills that generalize even across training levels, thereby boosting the overall training performance.

6.2. RandomMazes

In RandomMazes, each level consists of a randomly generated square maze with dimension uniformly sampled from 3 to 25. Mazes are generated using Kruskal's algorithm (Kruskal, 1956). The environment is partially observed, with the agent observing the 9×9 patch of cells directly surrounding its current location. At every cell is either a wall, an empty space, the goal, or the agent. The episode ends when the agent reaches the goal or when time expires after 500 timesteps. The agent's only actions are to move to an empty adjacent square. If the agent reaches the goal, a constant reward is received. Figure 8 reveals particularly strong overfitting, with a sizeable generalization gap even when training on 20,000 levels.

6.3. Discussion

In both CoinRun-Platforms and RandomMazes, agents must learn to leverage recurrence and memory to optimally navigate the environment. The need to memorize and recall past experience presents challenges to generalization unlike those seen in CoinRun. It is unclear how well suited LSTMs are to this task. We empirically observe that given sufficient data and training time, agents using LSTMs eventually converge to a near optimal policy. However, the relatively poor generalization performance raises the question of whether different recurrent architectures might be better suited for generalization in these environments. This investigation is left for future work.

7. Conclusion

Our results provide insight into the challenges underlying generalization in RL. We have observed the surprising extent to which agents can overfit to a fixed training set. Using the procedurally generated CoinRun environment, we can precisely quantify such overfitting. With this metric, we can better evaluate key architectural and algorithmic decisions. We believe that the lessons learned from this environment will apply in more complex settings, and we hope to use this benchmark, and others like it, to iterate towards more generalizable agents.

References

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012.

E. D. Cubuk, B. Zoph, D. Mané, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation policies from data. *CoRR*, abs/1805.09501, 2018. URL http://arxiv.org/abs/1805.09501.

T. Devries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017. URL http://arxiv.org/abs/1708.04552.

P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. Openai baselines. https://github.com/openai/baselines, 2017.

L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. *CoRR*, abs/1802.01561, 2018.

J. Farebrother, M. C. Machado, and M. Bowling. Generalization and regularization in DQN. *CoRR*, abs/1810.00123, 2018. URL http://arxiv.org/abs/1810.00123.

M. J. Hausknecht and P. Stone. The impact of determinism on learning atari 2600 games. In *Learning for General Competency in Video Games, Papers from the 2015 AAAI Workshop, Austin, Texas, USA, January 26, 2015.*, 2015. URL http://aaai.org/ocs/index.php/WS/AAAIW15/paper/view/9564.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

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

N. Justesen, R. R. Torrado, P. Bontrager, A. Khalifa, J. Togelius, and S. Risi. Illuminating generalization in deep reinforcement learning through procedural level generation. *CoRR*, abs/1806.10729, 2018.

J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society, 7*, pages 48–50, 1956.

P. Luo, X. Wang, W. Shao, and Z. Peng. Towards understanding regularization in batch normalization. *CoRR*, abs/1809.00846, 2018. URL http://arxiv.org/abs/1809.00846.

M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. J. Hausknecht, and M. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *J. Artif. Intell. Res.*, 61:523–562, 2018. doi: 10.1613/jair.5699. URL https://doi.org/10.1613/jair.5699.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman. Gotta learn fast: A new benchmark for generalization in RL. *CoRR*, abs/1804.03720, 2018. URL http://arxiv.org/abs/1804.03720.

C. Packer, K. Gao, J. Kos, P. Krähenbühl, V. Koltun, and D. Song. Assessing generalization in deep reinforcement learning. *CoRR*, abs/1810.12282, 2018.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, abs/1703.06907, 2017. URL http://arxiv.org/abs/1703.06907.

A. Zhang, N. Ballas, and J. Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *CoRR*, abs/1806.07937, 2018a.

C. Zhang, O. Vinyals, R. Munos, and S. Bengio. A study on overfitting in deep reinforcement learning. *CoRR*, abs/1804.06893, 2018b. URL http://arxiv.org/abs/1804.06893.