Illuminating Generalization in Deep Reinforcement Learning through Procedural Level Generation

Niels Justesen
IT University of Copenhagen
Copenhagen, Denmark
noju@itu.dk

Ruben Rodriguez Torrado
New York University
Brooklyn, USA
rrt264@nyu.edu

Philip Bontrager
New York University
Brooklyn, USA
philipjb@nyu.edu

Ahmed Khalifa
New York University
Brooklyn, USA
ahmed.khalifa@nyu.edu

Julian Togelius
New York University
Brooklyn, USA
julian@togelius.com

Sebastian Risi
IT University of Copenhagen
Copenhagen, Denmark
sebastian.risi@gmail.com

Abstract

Deep reinforcement learning (RL) has shown impressive results in a variety of domains, learning directly from high-dimensional sensory streams. However, when neural networks are trained in a fixed environment, such as a single level in a video game, they will usually overfit and fail to generalize to new levels. When RL models overfit, even slight modifications to the environment can result in poor agent performance. In this paper, we explore how procedurally generated levels during training increase generality. We show that for some games procedural level generation enables generalization to new levels within the same distribution. Additionally, it is possible to achieve better performance with less data by manipulating the difficulty of the levels in response to the performance of the agent. The generality of the learned behaviors is also evaluated on a set of human-designed levels. Our results show that the ability to generalize to human-designed levels highly depends on the design of the level generators. We apply dimensionality reduction and clustering techniques to visualize the generators' distributions of levels and analyze to what degree they can produce levels similar to those designed by a human.

Introduction

Deep reinforcement learning (RL) has shown remarkable results in a variety of domains, in particular, learning policies for video games (Justesen et al. 2017). However, there is increasing evidence that suggests that agents easily overfit to their particular training environment, resulting in policies that do not generalize well to related problems or even different instances of the same problem. Even small game modifications can often lead to dramatically reduced performance, leading to the suspicion that these networks learn reactions to particular situations rather than general strategies (Kansky et al. 2017; Zhang et al. 2018).

This paper has four contributions. **First**, we show that deep reinforcement learning overfits to a large degree on 2D arcade games when trained on a fixed set of levels. These results are important because similar setups are particularly popular to use as benchmarks in deep reinforcement learning research (e.g. the Arcade Learning Environment (Bellemare et al. 2013)). Our findings suggest that policies trained in such settings merely memorize certain action sequences rather than learning general strategies to solve the game. **Second**, we show that it is possible to overcome such overfitting by introducing Procedural Content Generation (PCG) (Shaker, Togelius, and Nelson 2016), more specifically procedurally generated levels, in the training loop. However, we show that this can lead to overfitting on a higher level, such as the distribution of generated levels presented during training. This paper investigates both types of overfitting and the effect of several level generators for multiple games. **Third**, we introduce a particular form of PCG-based reinforcement learning, which we call *Progressive PCG*, where the difficulty of levels/tasks is increased gradually to match the agent's performance. While similar techniques of increasing difficulty have been used before, they have not been combined with a PCG-based approach in which agents are *evaluated on a completely new level every time a new episode begins.* Our approach applies constructive level generation techniques, rather than pure randomization, and this paper studies the effect of several level generation methods. **Fourth**, we analyze distributions of procedurally generated levels using dimensionality reduction and clustering to understand whether it resembles human-designed levels.

It is important to note that the primary goal of this paper is not to achieve strong results on human levels, but rather to gain a deeper understanding of overfitting and generalization in deep RL, which is an important and neglected area in AI research. We believe this paper makes a valuable contribution in this regard, suggesting that a PCG-based approach could be an effective tool to study these questions from a fresh perspective. We also see this study relevant for robotics, where an ongoing challenge is to learn in simulated environments to then generalize to real-world scenarios.

Related Work

Within supervised learning, it is generally accepted that accuracy (and other metrics) are reported on a testing set that is

separate from the training set. In contrast, in reinforcement learning research it is common to report results on the very same task a model was trained on. However, several recent learning-focused game AI competitions, such as the Visual Doom (Kempka et al. 2016) AI Competition, The General Video Game AI Learning Track (Liu, Perez-Lebana, and Lucas ; Rodriguez Torrado et al. 2018) and the OpenAI Retro Contest[1] evaluate the submitted controllers on levels that the participants did not have access to. However, none of them are based on procedurally generated levels. The only game AI competition to prominently feature procedural level generation is the Mario AI Competition which did not have provisions for learning agents (Togelius et al. 2013).

Randomization of objects in simulated environments has shown to improve generality for robotic grasping to such a degree that the robotic arm could generalize to realistic settings as well (Tobin, Zaremba, and Abbeel 2017). Low-fidelity texture randomization during training in a simulated environment has allowed for autonomous indoor flight in the real world (Sadeghi and Levine 2016). Random level generation has been applied to video games to enable generalization of reinforcement learning agents (Beattie et al. 2016; Graves et al. 2016; Groshev et al. 2017). Several RL approaches exist that manipulate the reward function instead of the structure of the environment to ease learning and ultimately improve generality, such as Hindsight Experience Replay (Andrychowicz et al. 2017) and Rarity of Events (Justesen and Risi 2018).

The idea of training agents on a set of progressively harder tasks is an old one and has been rediscovered several times within the wider machine learning context. Within evolutionary computation, this practice is known as incremental evolution (Gomez and Miikkulainen 1997; Togelius and Lucas 2006). For example, it has been shown that while evolving neural networks to drive a simulated car around a particular race track works well, the resulting network has learned only to drive that particular track; but by gradually including more difficult levels in the fitness evaluation, a network can be evolved to drive many tracks well, even hard tracks that could not be learned from scratch (Togelius and Lucas 2006). Essentially the same idea has later been independently invented as curriculum learning (Bengio et al. 2009). Similar ideas have been formulated within a coevolutionary framework as well (Brant and Stanley 2017).

Several machine learning algorithms also gradually scale the difficulty of the problem. Automated curriculum learning includes intelligent sampling of training samples to optimize the learning progress (Graves et al. 2017). Intelligent task selection through asymmetric self-play with two agents can be used for unsupervised pre-training (Sukhbaatar et al. 2017). The POWERPLAY algorithm continually searches for new tasks and new problems solvers concurrently (Schmidhuber 2013) and in Teacher-Student Curriculum Learning (Matiisen et al. 2017) the teacher tries to select sub-tasks for which the slope of the learning curve of the student is highest. Reverse curriculum generation automatically generates a curriculum of start states, further and further away from the goal, that adapts to the agent's performance (Florensa et al. 2017).

A protocol for training reinforcement learning algorithms and evaluate generalization and overfitting, by having large training and test sets, was proposed in (Zhang et al. 2018). Their experiments show that training on thousands of levels in a simple video game enables the agent to generalize to unseen levels. Our (contemporaneous) work here differs by implementing an adaptive difficulty progression along with near endless content generation. By testing on several more complex games we also explore what types of problems that agents are able to generalize.

There has also been some work where a single network has been trained to play multiple games simultaneously, such as the IMPALA system (Espeholt et al. 2018). In that work, the same games were used for training and testing, and it is in principle possible that the network simply learned individual behaviors for all of these games within the shared model.

General Video Game AI Framework

We are building on the General Video Game AI framework (GVG-AI) which is a flexible framework designed to facilitate the advance of general AI through video game playing (Perez-Liebana et al. 2016). There are currently over 160 games written for GVG-AI using which are specified using the declarative video game description language (VGDL) (Schaul 2013), originally proposed in (Ebner et al. 2013). The game definition specifies objects in the game and interaction rules such as rewards and effects of collisions. A level is defined as an ASCII grid where each character represents an object. This allows for quick development of games and levels making the framework ideal for research purposes (Perez-Liebana et al. 2018).

The GVGAI framework has been integrated with the OpenAI Gym environment (Rodriguez Torrado et al. 2018) which provides a unified RL interface across several different environments (Brockman et al. 2016) as well as state-of-the-art RL implementations (Dhariwal et al. 2017). While GVG-AI originally provides a forward model that allows agents to use search algorithms, the GVG-AI Gym only provides the pixels of each frame, the incremental reward, and whether the game is won or lost.

Parameterized Level Generator

For this paper, constructive level generators are built for several games in GVG-AI: Boulderdash, Frogs, Solarfox and Zelda. These games were selected as they can be considered hard; most of the GVG-AI tree search agents do not perform well and can thus be considered hard (Bontrager et al. 2016). Constructive level generators (Shaker, Togelius, and Nelson 2016) are widely used in game development because they are relatively fast and easy to debug. The constructive level generators incorporate game knowledge during the generation process to make sure the output level is directly playable without testing. Our generators are designed after analyzing the core components in the human-designed levels for each game. All the generators are parameterized using maximum

[1] https://contest.openai.com/

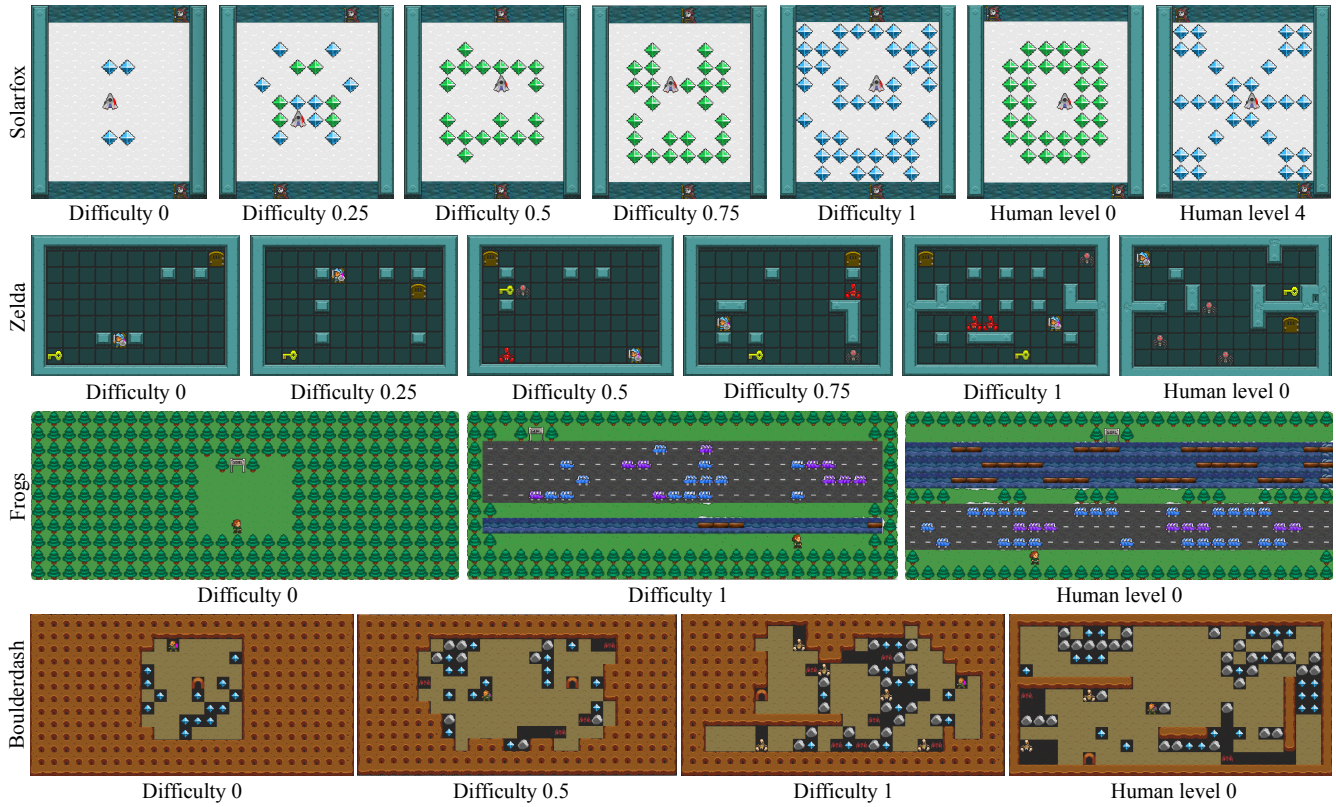


Figure 1: Procedurally generated levels for Solarfox, Zelda, Frogs, and Boulderdash with various difficulty between 0 and 1. Human-designed levels are shown for each game as well.

width, maximum height, and a difficulty parameter. The generators are handmade in this work to remove the variability of general level generators (Khalifa et al. 2016) and allow a better analysis of training on procedurally generated levels.

Boulderdash Level Generator: This game is a GVG-AI port of "Boulder Dash" (First Star Software, 1984). Here the player tries to collect at least ten gems and then go to the exit door while avoiding falling boulders and attacking enemies. The level generation in Boulderdash works as follows: (1) Generate the layout of the map using Cellular Automata (Johnson, Yannakakis, and Togelius 2010). (2) Add the player to the map at a random location. (3) Add the exit door at a random location away from the player. (4) Add at least ten gems to the map at random locations. (5) Add enemies to the map at random locations in a similar manner to the third step.

Frogs Level Generator: Frogs is a GVG-AI port of "Frogger" (Konami, 1981). In Frogs, the player tries to move upwards towards the goal without getting killed either by drowning in water or by getting run over by a car. The level generation in Frogs follow these steps: (1) Add the player at the lowest empty row in the level. (2) Add the goal at the highest row in the level. (3) Assign the intermediate rows either as roads, water, or forest. (4) Add cars to the roads and wood logs to water.

Solarfox Level Generator: Solarfox is a GVG-AI port of "Solar Fox" (Midway Games, 1981).In Solarfox, the player is always moving in one of the four directions (North, South, East, and West). The player tries to collect all the gems in the level while avoiding hitting the borders or the enemy bullets coming either from the north or the south. The level generation for Solarfox follow these steps: (1) Add the player in the middle of the map. (2) Add some gems either in the upper half, left half, or upper left quarter. (3) Replicate the same pattern of gems on the remaining parts of the map.

Zelda Level Generator: Zelda is a GVG-AI port of the dungeon system in "The Legend of Zelda" (Nintendo, 1986). In Zelda, the goal is to grab a key and exit through a door without getting killed by enemies. The player can use their sword to kill enemies for higher scores. The level generation in Zelda works as follows: (1) Generate the map layout as a maze using Prim's Algorithm (Buck 2015). (2) Remove some of the solid walls in the maze at random locations. (3) Add the player to a random empty tile. (4) Add the key and exit door at random locations far from the player. (5) Add enemies in the maze at random locations far away from the player.

We can control the difficulty of the generators using a *difficulty parameter* in [0;1] passed to the generator during the generation process. Figure 1 shows the effect of the difficulty parameter in the four games. Increasing the difficulty has three effects. First, the active level size increases (except in Zelda and Solarfox where the level size is fixed), which is the area in the level where the player can move through.

Second, the number of objects that can kill the player and/or the number of objects that the player can collect is increased. Third, the layout of the level gets more complex to navigate. The level generator search space is extremely big around 10^8 at low difficulty to 10^{24} at high difficulties. Difficult levels have more possible configurations as they typically have more elements.

Procedural Level Generation for Deep RL

In a supervised learning setting, generality is obtained by training a model on a large dataset, typically with thousands of examples. Similarly, the hypothesis in this paper is that RL algorithms should achieve generality if many variations of the environments are used during training, rather than just one. This paper presents a novel RL framework wherein a new level is generated *whenever a new episode begins*, which allows us to algorithmically design the new level to match the agent's current performance. This framework also enables the use of search-based PCG techniques, that e.g. learn from existing level distributions (Volz et al. 2018), which remove the dependency on domain knowledge.

When the learning algorithm is presented with new levels continuously during training, it must learn general strategies in the game to improve. Learning a policy this way is more difficult than learning one for just a single level and it may be infeasible if the game rules and/or generated levels have sparse rewards. To ease the learning, the difficulty of the generated levels is controlled by our learning algorithm. In this way, the level generator will initially create easy levels and progressively increase the difficulty as the agent learns.

Our proposed method, **Progressive PCG** (PPCG), uses a level generator with a difficulty setting between 0 and 1. The difficulty of levels is never measured. Instead, the difficulty setting is only used as input to level generator. We implemented a simple control mechanism where levels initially are requested with difficulty 0. If the agent wins an episode, the difficulty will be incremented such that future levels during training are harder. The difficulty is increased by α for a win and decreased by the same amount for a loss. In our experiments, we used $\alpha = 0.01$. For distributed learning algorithms, the difficulty setting is shared across all processes such that the outcome of all episodes influences the difficulty of future training levels. We compare PPCG to a simpler method, also using procedurally generated levels, but with a constant difficulty level. We refer to this approach as **PCG X**, where *X* is the difficulty level.

Experiments

To evaluate our approach, we employ the reinforcement learning algorithm *Advantage Actor-Critic* (A2C) (Mnih et al. 2016). We use the implementation of A2C from the Open AI Baselines and run it on the GVG-AI Gym framework. The neural networks in this paper have the same architecture originally used from Mnih et al. (Mnih et al. 2016) with three convolutional layers and a single fully-connected layer. The output consists of both a policy and value output in contrast to DQN. A2C is using 12 parallel workers, a step size $t_{max} = 5$, no frame skipping as in (Rodriguez Torrado et al. 2018), and a constant learning rate of 0.007 with the RMS optimizer (Ruder 2016). The code for our experiments will be made available online[2].

We compare four different training approaches in Zelda. **Lv X**: Training level is a single human-designed level where *X* denotes which of the five human-designed levels. **Lv 0-3**: Several human-designed levels (level 0, 1, 2, and 3) which are sampled randomly during training. **PCG X**: Procedurally generated training levels with a constant difficulty *X*. **Progressive PCG (PPCG)**: procedurally generated training levels where the difficulty is adjusted to the performance of the agent.

Each training setting was repeated four times and tested on two sets of 30 pre-generated levels with either difficulty 0.5 and 1 as well as the five human-designed levels. The training plots on Figure 2 and the test results in Table 1 are averaged across the four trained models where each model was tested 30 times on each test setup (thus a total of 120 episodes per training setup). All four training approaches were tested on Zelda. Only PCG and PPCG were tested on Solarfox, Frogs, and Boulderdash. The trained agents are also compared to an agent taking uniformly random actions.

Results

Training on a few Human-designed Levels

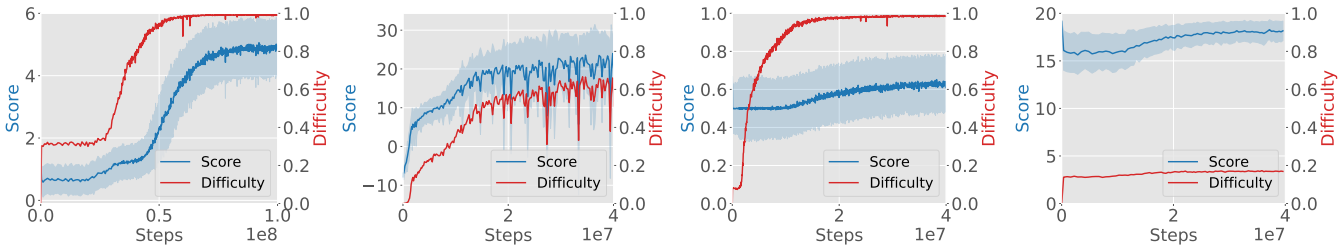
Policies trained on just one level in Zelda (*Lv 0* and *Lv 4* in Table 1) reach high scores on the training level but has poor performance on all test levels; human-designed and procedurally generated. It is clear that these are prone to memorization and cannot adapt well to play new levels. The scores on the training levels are close to the maximum scores achievable while the scores on the test levels are lower than the random policy, a clear indication of overfitting in reinforcement learning. Policies trained on four human-designed levels in Zelda also achieve high scores on all four training levels. The testing scores are marginally higher than when trained on a single level, on both the human-designed level 4 and the PCG levels.

Training on Procedurally Generated Levels

Agents trained on procedurally generated levels with a fixed difficulty learned a general behavior within the distribution of procedurally generated levels, with mediocre scores in Zelda, Solarfox, and Boulderdash, while in Frogs it never progressed at all.

It has previously been shown that DQN and A2C fail to learn anything on just one level in Frogs using using 1 million training steps (Rodriguez Torrado et al. 2018). While PCG 1, here with 40 million steps, also fails to learn Frogs, PPCG achieves a score of 0.57 (57% win rate) in the test set of procedurally generated levels with difficulty 1 (comparable to human levels in difficulty - see Figure 1). For Zelda, PCG 1 was able to achieve strong scores while PPCG is slightly better. Interestingly, for the two cases where PPCG is able to reach difficulty 1 during training (Frogs and Zelda), it produces better results than PCG 1 on PCG

[2] https://github.com/njustesen/a2c_gvgai



(a) PPCG in Zelda (b) PPCG in Solarfox* (c) PPCG in Frogs (d) PPCG in Boulderdash

Figure 2: Smoothed mean scores and level difficulties during training across five repetitions of Progressive PCG in Zelda, Solarfox, Frogs, and Boulderdash. Opaque coloring show one standard deviation. *Only three repetitions of PPCG and one of PCG 1 for Solarfox.

Training	PCG 0.5	PCG 1	Lv 0	Lv 1	Lv 2	Lv 3	Lv 4
Zelda							
Max.	4.40	6.87	8.00	8.00	8.00	10.00	8.00
Random	0.38	0.22	0.26	0.17	-0.11	-0.07	0.18
60M steps:							
Level 0	0.28	0.51	6.97	-0.45	-0.53	0.07	-0.58
Level 4	0.56	0.07	-0.51	0.99	0.04	-0.35	5.93
Level 0-3	1.98	2.37	6.95	7.17	7.20	8.17	**1.91**
PCG 0.5	3.45	4.00	2.21	2.28	0.92	**2.27**	0.15
PCG 1	0.27	3.56	2.40	1.37	1.49	2.88	-0.62
PPCG	3.44	4.28	2.67	3.35	2.43	1.89	0.96
100M steps:							
PCG 1	3.05	4.38	2.49	1.54	1.18	2.04	-0.29
PPCG	**3.82**	**4.51**	**2.71**	**3.74**	**2.84**	1.90	0.88
Solarfox*							
Max.	30.83	51.83	32.00	32.00	34.00	70.00	62.00
Random	-3.68	-4.55	-5.49	-4.80 ,	-5.41	2.03	1.13
40M steps:							
PCG 1	**20.70**	**32.43**	**22.00**	**21.83**	**26.00**	**43.96**	**28.16**
PPCG	16.08	21.40	16.87	10.26	12.02	27.37	20.00
Frogs							
Max.	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Random	0.01	0.00	0.00	0.00	0.00	0.00	0.00
40M steps:							
PCG 1	0.01	0.00	0.00	0.00	0.00	0.00	0.00
PPCG	**0.81**	**0.57**	0.00	0.00	0.00	0.00	0.00
Boulderdash							
Max.	31.50	29.80	48.00	52.00	58.00	48.00	44.00
Random	6.29	3.71	0.85	2.58	3.5	0.65	2.66
60M steps:							
PCG 1	**14.63**	**8.32**	**5.39**	**10.28**	**5.85**	**5.08**	**8.27**
PPCG	11.78	4.86	3.44	0.98	0.68	0.41	3.32

Table 1: Test results of A2C using several training settings, including; a single human-designed level (*Level 0* and *Level 4*), several human-designed levels (*Level 0-3*), procedurally generated leves with a fixed difficulty (*PCG 0.5* and *PCG 1*), and *PPCG* that progressively adapts the difficulty of the levels to match the agent's performance. *Random* refers to results of an agent taking uniformly random actions and *Max.* shows the maximum possible score. Scores are in red if training level are the same as the test level. The best scores for a game, that is not marked red, are in bold. *Only three repetitions of PPCG and one of PCG 1 were made for Solarfox so far.

1 levels. As PPCG never requests any difficult levels during training, this is expected. In Boulderdash, the agents trained with PCG 1 reaches decent scores (8.34 on average) on levels with difficulty 1. PPCG learns to achieve high scores during training but it fails to win the game when the difficulty is around 0.2.

Generalization on Human-Designed Levels

From the results, it is clear that by introducing procedurally generated levels we can obtain behaviors that can generalize to unseen levels within the training distribution. It is, however, interesting whether they also generalize to the five human-designed levels in GVG-AI.

In Zelda, the performance of PCG and PPCG in human-designed levels are descent but lower than in the procedurally generated levels. In Frogs, PCG and PPCG are unable to win any of the human-designed levels indicating a clear discrepancy between the two level distributions. In Boulderdash, PCG 1 achieved on average 5.08–10.28 points (out of 20) in the five human-designed levels compared to 8.32–14.63 on the procedurally generated levels. PPCG perform worse since it never reached a difficulty level during training similar to the human-designed levels. Similarly, in Solarfox, PCG 1 achieved on average a higher score on the five human-designed levels compared to PPCG agents as the PPCG agents didn't reach difficulty of 1.0 during training. PCG 1, however, shows remarkable generalization with no clear drop in performance.

Qualitative Analysis of Agent Replays

In Zelda, it is clear that PPCG has learned to reliably strike down and avoid enemies but is only rarely able to collect the key and exit through the door. Whether this is due to the difficulties of navigating in tricky mazes or lack of motivation towards the key and door is unknown. One of the assumptions for the difficulty to improve in Zelda, is that the agent relies on enemies' movement to reach new locations where the key or/and the door is sometimes close by and easy to reach. In Solarfox, PCG 1 has learned to effectively pick up the diamonds and avoid the fireballs, occasionally getting hit while trying to get away from them. This behavior is remarkably human-like. Sometimes the agent is able to win the human-designed levels which is quite impressive. As PPCG

perfomance is lacking in the hard levels we notice that it jiggles around the starting location to collect nearby diamonds. We think that behavior arose because the easy procedurally generated levels have diamonds near the starting location. In Frogs, PPCG is able to navigate towards goal. Sometimes it looses when crossing the water when only few logs are available. We suspect that navigation in this game is learned more easily than in other games as the goal in Frogs is always in the top and thus moving up is a simple heuristic to learn. In Boulderdash, PCG 1 learned basic skills in fighting and picking up nearby diamonds, also under boulders, while it does not seem to be capable of long-term planning. It often fails to fight and move boulders and thus dies rather quickly in most level. Since it often dies from boulders and enemies, it can explain why PPCG never reached a difficulty higher than 0.2. It simply gets killed early when boulders and enemies are introduced.

Exploring the Distribution of Generated Levels

The learned policies ability to generalize to human-designed levels highly depends on the game and the design of the level generators. We do not expect agents to play well on levels that are dissimilar from their training distribution. To investigate the distribution of the procedurally generated levels, and how the structure of these levels correlate with human-designed levels, we have generated 1000 levels for each game with difficulty 1. The high-dimensional structure of each level have been compressed to two dimensions using principal component analysis (PCA), whereafter the transformed points are clustered using density-based spatial clustering of applications with noise (DBSCAN). The transformed space of levels are visualized in Figure 3. For PCA to work on GVG-AI levels, they have been transformed into a 3D array of shape (tile_type, height, width) containing 0's and 1's and then reshaped into a 1D array. The human-designed levels were included in both the transformation and clustering processes.

The generated levels for Solarfox are clustered in three wide groups; (1) levels with only green diamonds, (2) levels with both green and blue diamonds, and (3) levels with only blue diamonds. None of the human-designed levels use both types of diamonds and thus only belong to two of the clusters. For Zelda, only one cluster is discovered without outliers. Interestingly, PCG 1 and PPCG generalizes best to the human-designed levels that are closest to the centroid (level 0 and 1) and worst to levels further from the centroid (level 2 and 4). The generated levels in Frogs have been clustered into 19 groups. This is due to the high structural effect of roads and rivers that goes across the level. It is noticeable how level 4 is the most distant outlier. This is because level 4 has a river on the starting row which our generator is constrained not to create. Level 0–3 are near the same small cluster while the generated levels are spread across many isolated clusters. It is not exactly clear why PCG 1 and PPCG fail to play on all the human-designed Frogs levels but the structure of the level space is remarkably different from the other games. In Boulderdash, similarly to Zelda, only one cluster emerges, but all human-designed levels are distant outliers. This clustering is a result of the fixed amount of open space in the human-designed levels with padding of only one tile; the generated levels are more varied and cave-like.

Discussion

The results of our experiments affirm the original concern with the way reinforcement learning research is often evaluated and reported. When it is reported that an agent has learned to play a game, it may simply mean that the policy has found actions for a small subspace of the possible observations the game can offer. This boils down to the network mapping seen observations in this subspace to actions without learning general concepts of the game. Table 1 shows this with the huge disparity between the red and black numbers; the difference in performance on the training levels and test levels. If the goal of the agent is to learn how to play a game, then this work shows that it must be evaluated in several test environments.

Our results demonstrate the effectiveness of using procedurally generated levels in the training loop while it also presents us with new challenges. For PPCG, there is a design challenge in how to scale the complexity of the environment to smoothen the learning curve. In Frogs, it was very effective to apply padding to easy levels, creating smaller levels in the beginning, while it was not sufficient for Boulderdash. Another challenge is how to ensure that the distribution of procedurally generated levels match another distribution, in this case human-designed levels. We have provided a tool using dimensionality reduction and clustering which can be used to improve the design of constructive level generators or perhaps to guide search-based level generators in future work. While our results vary across the four games, we have the opportunity to analyze when our approach works and when it fails. We believe that search-based PCG is an interesting area for future work that could ultimately lead to RL agents with more general policies. We believe that this study is also relevant for robotics; learning to generalize from simulation to real-world scenarios where pure randomization of the environment is insufficient.

Conclusion

We explored how policies learned with deep reinforcement learning generalize to levels that were not used during training. The results demonstrate that agents trained on just one or a handful of levels often fail to generalize to new levels. This paper presented a new approach that incorporates a procedural level generator into the reinforcement learning framework, in which a new level is generated for each episode. Training on a large set of levels requires much more training time compared to training on just one level. Our experiments show that training on hard procedurally generated levels can be infeasible in some games. Our *Progressive PCG* (PPCG) approach increases the difficulty of the generated levels when the agent wins during training and decreases the difficulty when it loses. This technique was able to achieve a win rate of 57%, compared to 1% with-

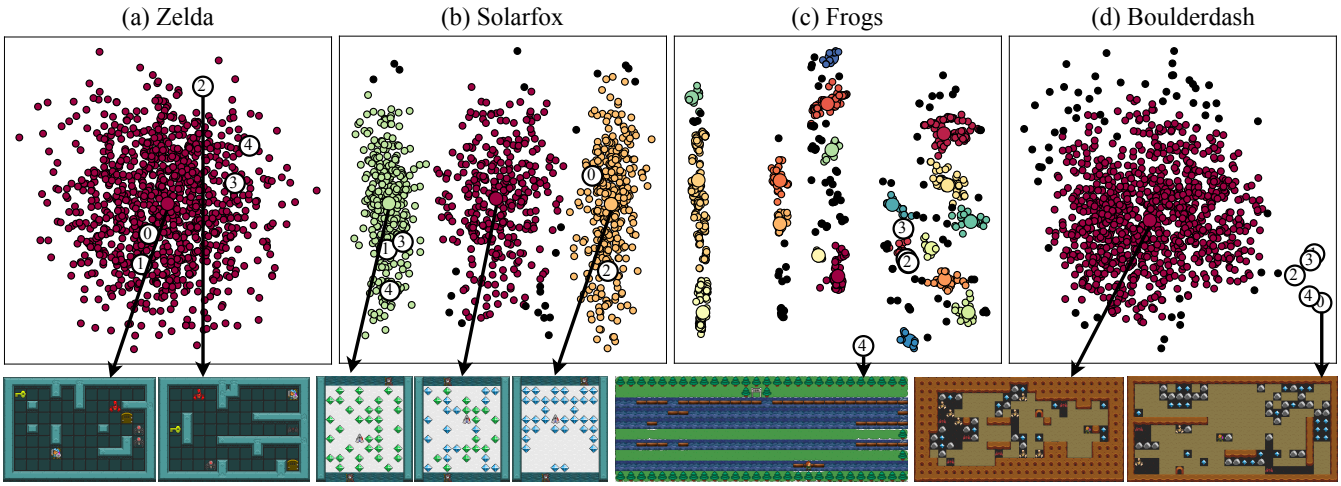


Figure 3: Visualization of the level distribution and how they correlate to human-designed levels (white circles). Levels were reduced to two dimensions using PCA and clustered using DBSCAN ($\epsilon = 0.5$ and a minimum of 10 samples per cluster). Outliers are black and centroids are larger.

out, in difficult Frogs levels. Additionally, in Zelda this approach was superior across procedurally generated levels and human-designed levels. In Solarfox and Boulderdash, the level difficulty of PPCG never reached the maximum during training and here procedurally generated levels with a fixed difficulty setting was best. This results of this paper also highlights the important challenge of ensuring that the training distribution resembles test distributions. We have provided a tool that can assist with the second challenge, that uses dimensionality reduction and clustering to visualize the difference between two distributions of video game levels.

Acknowledgements

Niels Justesen was financially supported by the Elite Research travel grant from The Danish Ministry for Higher Education and Science. Ahmed Khalifa acknowledges the financial support from NSF grant (Award number 1717324 - "RI: Small: General Intelligence through Algorithm Invention and Selection.").

References

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, O. P.; and Zaremba, W. 2017. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, 5048–5058.

Beattie, C.; Leibo, J. Z.; Teplyashin, D.; Ward, T.; Wainwright, M.; Küttler, H.; Lefrancq, A.; Green, S.; Valdés, V.; Sadik, A.; et al. 2016. Deepmind lab. *arXiv preprint arXiv:1612.03801*.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47:253–279.

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, 41–48. ACM.

Bontrager, P.; Khalifa, A.; Mendes, A.; and Togelius, J. 2016. Matching games and algorithms for general video game playing. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 122–128.

Brant, J. C., and Stanley, K. O. 2017. Minimal criterion coevolution: a new approach to open-ended search. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 67–74. ACM.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

Buck, J. 2015. *Mazes for Programmers: Code Your Own Twisty Little Passages*. Pragmatic Bookshelf.

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

Ebner, M.; Levine, J.; Lucas, S. M.; Schaul, T.; Thompson, T.; and Togelius, J. 2013. Towards a video game description language.

Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Doron, Y.; Firoiu, V.; Harley, T.; Dunning, I.; et al. 2018. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*.

Florensa, C.; Held, D.; Wulfmeier, M.; Zhang, M.; and Abbeel, P. 2017. Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300*.

Gomez, F., and Miikkulainen, R. 1997. Incremental evolution of complex general behavior. *Adaptive Behavior* 5(3-4):317–342.

Graves, A.; Wayne, G.; Reynolds, M.; Harley, T.; Danihelka, I.; Grabska-Barwińska, A.; Colmenarejo, S. G.; Grefenstette, E.; Ramalho, T.; Agapiou, J.; et al. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature* 538(7626):471.

Graves, A.; Bellemare, M. G.; Menick, J.; Munos, R.; and Kavukcuoglu, K. 2017. Automated curriculum learning for neural networks. *arXiv preprint arXiv:1704.03003*.

Groshev, E.; Goldstein, M.; Tamar, A.; Srivastava, S.; and Abbeel, P. 2017. Learning generalized reactive policies using deep neural networks. *arXiv preprint arXiv:1708.07280*.

Johnson, L.; Yannakakis, G. N.; and Togelius, J. 2010. Cellular automata for real-time generation of infinite cave levels. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, 10. ACM.

Justesen, N., and Risi, S. 2018. Automated curriculum learning by rewarding temporally rare events. *arXiv preprint arXiv:1803.07131*.

Justesen, N.; Bontrager, P.; Togelius, J.; and Risi, S. 2017. Deep learning for video game playing. *arXiv preprint arXiv:1708.07902*.

Kansky, K.; Silver, T.; Mély, D. A.; Eldawy, M.; Lázaro-Gredilla, M.; Lou, X.; Dorfman, N.; Sidor, S.; Phoenix, S.; and George, D. 2017. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. *arXiv preprint arXiv:1706.04317*.

Kempka, M.; Wydmuch, M.; Runc, G.; Toczek, J.; and Jaśkowski, W. 2016. ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games*, 341–348. Santorini, Greece: IEEE. The best paper award.

Khalifa, A.; Perez-Liebana, D.; Lucas, S. M.; and Togelius, J. 2016. General video game level generation. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, 253–259. ACM.

Liu, J.; Perez-Lebana, D.; and Lucas, S. M. The single-player gvgai learning framework technical manual.

Matiisen, T.; Oliver, A.; Cohen, T.; and Schulman, J. 2017. Teacher-student curriculum learning. *arXiv preprint arXiv:1707.00183*.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 1928–1937.

Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Lucas, S. M.; and Schaul, T. 2016. General Video Game AI: Competition, Challenges and Opportunities. In *Thirtieth AAAI Conference on Artificial Intelligence*, 4335–4337.

Perez-Liebana, D.; Liu, J.; Khalifa, A.; Gaina, R. D.; Togelius, J.; and Lucas, S. M. 2018. General video game ai: a multi-track framework for evaluating agents, games and content generation algorithms. *arXiv preprint arXiv:1802.10363*.

Rodriguez Torrado, R.; Bontrager, P.; Togelius, J.; Liu, J.; and Perez-Liebana, D. 2018. Deep reinforcement learning for general video game ai. In *Computational Intelligence and Games (CIG), 2018 IEEE Conference on*. IEEE.

Ruder, S. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Sadeghi, F., and Levine, S. 2016. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*.

Schaul, T. 2013. A video game description language for model-based or interactive learning. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, 1–8. IEEE.

Schmidhuber, J. 2013. Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. *Frontiers in psychology* 4:313.

Shaker, N.; Togelius, J.; and Nelson, M. J. 2016. *Procedural content generation in games*. Springer.

Sukhbaatar, S.; Lin, Z.; Kostrikov, I.; Synnaeve, G.; Szlam, A.; and Fergus, R. 2017. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*.

Tobin, J.; Zaremba, W.; and Abbeel, P. 2017. Domain randomization and generative models for robotic grasping. *arXiv preprint arXiv:1710.06425*.

Togelius, J., and Lucas, S. M. 2006. Evolving robust and specialized car racing skills. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, 1187–1194. IEEE.

Togelius, J.; Shaker, N.; Karakovskiy, S.; and Yannakakis, G. N. 2013. The mario ai championship 2009-2012. *AI Magazine* 34(3):89–92.

Volz, V.; Schrum, J.; Liu, J.; Lucas, S. M.; Smith, A.; and Risi, S. 2018. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. *arXiv preprint arXiv:1805.00728*.

Zhang, C.; Vinyals, O.; Munos, R.; and Bengio, S. 2018. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*.