Proceedings of the Seventeenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment
(AIIDE 2021)

Search-Based Exploration and Diagnosis of TOAD-GAN

Maria Edwards, Ming Jiang, Julian Togelius
New York Brooklyn, New York
mariaedwards@nyu.edu, mj1279@nyu.edu, julian@togelius.com

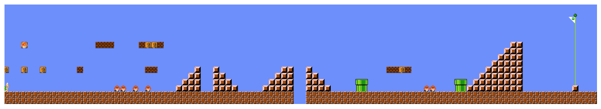
Abstract

Generative Adversarial Networks (GANs) have been used with great success to generate images. They have also been applied to the task of Procedural Content Generation (PCG) in games, particularly for level generation, with various approaches taken to solving the problem of training data. One of those approaches, TOAD-GAN (Token-Based One-Shot Arbitrary Dimension Generative Adversarial Network) (Awiszus, Schubert, and Rosenhahn 2020), can generate levels based on a single training example and has been able to closely reproduce token patterns found in the training sample. While TOAD-GAN is an impressive achievement, questions remain about what exactly it has learned. Can the generator be made to produce levels that are substantially different from the level it has been trained on? Can it reproduce specific level segments? How different are the generated levels? We investigate these questions and others by using the CMA-ES algorithm for Latent Space Evolution. To make the search space feasible, we use a random projection in latent space. We propose the investigation undertaken here as a paradigm for studies into what machine-learned generators have actually learned, and also as a test of a new method for projecting from a smaller search space to a larger latent space.

Introduction

Level generation is an important part of PCG, or procedural content generation (Shaker, Togelius, and Nelson 2016). The use of computer algorithms to produce new, playable levels can provide inspiration to video game designers and expand the space of content available to players. Generative Adversarial Networks, or GANs, can be a very useful tool within this domain. However, their output may not be able to meet all requirements, specifically when training data is not readily available. Further techniques can be used to filter and fine-tune the output of such GANs in order to expand their applicability. Evolutionary search is one of those techniques.

TOAD-GAN (Awiszus, Schubert, and Rosenhahn 2020) is a novel application of a Generative Adversarial Network for single-shot level generation that seeks to present a solution to the problem of scarce training data. It adapts the one-shot approach of SinGAN (Shaham, Dekel, and Michaeli 2019), a generative model that can be learned from a single image of the the natural world, to model levels based on a single training example in token-based games. It does this by downsampling the training levels to a few different scales, taking care to preserve important features, and training a discriminator and generator pair on each different scale. This was implemented on Super Mario Bros (Fig.1) and Mario Kart, and is meant to be widely applicable to arbitrary token-based games.


(a) Mario level 1.

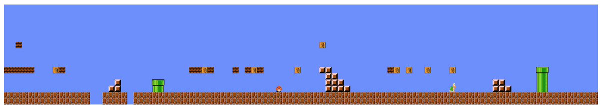

(b) A level generated by TOAD-GAN based on Mario level 1 using a noise vector with a standard gaussian distribution.

Figure 1: Mario level 1 and a visually similar level generated by TOAD-GAN.

However, there are a number of characteristics that TOAD-GAN does not account for. For example, it does not test for playability. A visual examination of some levels generated (Fig.2) reveal notable problems that might prevent a player or AI agent from successfully completing the level. This presents a hurdle to the usefulness of TOAD-GAN because while it produces visually similar new levels, further effort is required to evaluate generated levels for playability.

When examining levels for playability, it can also be noted that there are a wide range of levels that might be generated after training on a single Mario level, and that while many of these levels might be very similar to the training sample, some might also be very different.

A quick visual inspection of levels generated by TOAD-GAN reveals that large areas of missing ground or platform, pipes starting in midair, enemies buried in the ground, and other characteristics that might limit usability are common in the resulting output. Further examination is necessary to illuminate the full range of levels that might be generated by TOAD-GAN given a training sample.

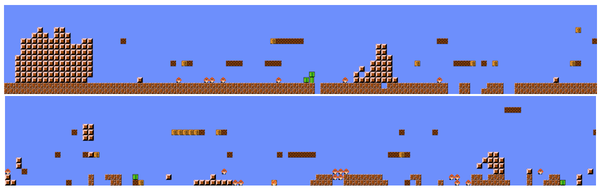


Figure 2: Unplayable levels generated by TOAD-GAN after training on Mario level 1. Both levels were generated using a noise vector with a standard gaussian distribution.

This work seeks to examine that range, and in doing so to explore the opportunity for using evolutionary search to find levels satisfying specified requirements within the latent space of a GAN. This approach can expand the applicability of GANs, especially in domains where the limited availability of training data means that a trained GAN will still generate a significant number of samples that are not usable.

Further, this work seeks to extend the applicability of evolutionary search by proposing a technique for projection from a small search space to a larger latent space. In this technique, a randomly initialized neural network takes a smaller input vector and outputs a much larger noise vector. This reduction of a higher-dimensional space to a smaller one allows for an efficient exploration of that space using methods that might otherwise be computationally prohibitive. Here, this method allows evolutionary search to be used to explore the latent space of TOAD-GAN.

Related Work

PCG for Level Generation

Procedural content generation for token-based game levels and specifically for Super Mario Brothers has a been undertaken using a wide range of methods. Some researchers have used Monte Carlo Tree Search to guide Markov chain generation for Super Mario Brothers (Snodgrass and Ontañón 2014). Others have used Markov chains to learn statistical patterns from human-authored maps, and used those trained Markov chains to generate Super Mario Brothers levels (Summerville, Philip, and Mateas 2021). Still others have generated levels for the same game by stitching together previously defined mini-level scenes showcasing various game mechanics (Green et al. 2020). This is just a small sampling of the methods that have been used to generate Super Mario Brothers levels.

Machine Learning and Neural Networks have also been applied to the task of level generation in the field of PCG via Machine Learning (PCGML), training neural networks on human authored levels in order to generate new content (Summerville et al. 2018; Liu et al. 2020). Reinforcement Learning (RL) has been applied in this field as well, treating the task of level design as a game played by the RL agent (Khalifa et al. 2020).

Research into using GANs for PCG has often acknowledged the problem posed by lack of adequate training data (Volz et al. 2018; Torrado et al. 2019; Awiszus, Schubert, and Rosenhahn 2020). GANs consist of a generator tasked with mapping random noise vectors to output of a specified format such as an image, and a discriminator tasked with determining whether a sample is real, part of the training data, or false, part of the generated data. In training, the generator seeks to produce more and more realistic samples and the discriminator seeks to become better and better at detecting fake samples, creating an adversarial relationship where one network is successful only when the other is not. However, without adequate training data, a discriminator may simply memorize the patterns of a few specific examples and reject all other patterns, rendering the model ineffective.

A variety of approaches, including that of TOAD-GAN, have attempted to mitigate this risk, with a few described below.

The architecture used by TOAD-GAN was first implemented for use with images of nature in SinGAN (Shaham, Dekel, and Michaeli 2019). This architecture takes a single training image and examines it at different scales, training a lightweight convolutional GAN at each scale to model the distribution of patches within the image. Training a GAN to genenerate images by progressively working at different scales has also been done in the past as applied to the CelebA dataset (Karras et al. 2017).

Other researchers (Torrado et al. 2019) have proposed a GAN with an architecture utilizing conditional embeddings to reduce memory usage and speed up training, and self-attention to model long-range dependencies. This model aimed to produce levels to meet functional requirements such as playability, rather than simply pattern-based similarity. This approach applied bootstrapping techniques as well, so that successfully generated levels were added to the training corpus, increasing the variety within that corpus and reducing the likelihood of problems arising from insufficient training data.

Latent Space Examination

Rather than altering the architecture of the GAN itself, some have applied search-based methods to further examine the latent space of a convolutional GAN. As mentioned above, Volz et al. applied this to the problem of using a GAN to generate Mario levels (Volz et al. 2018). This work applies the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen and Ostermeier 2001; Hansen 2016). First, Mario levels are generated using a Deep Convolutional GAN (DCGAN) based on a single training sample, and then CMA-ES is used to search through the space of those levels for samples satisfying various fitness requirements

This approach builds on latent variable evolution (LVE) (Bontrager, Togelius, and Memon 2017), which trains a neural network to generate images and searches over the latent variables of the network, or the space of input vectors, for input that will produce the right output. LVE was originally applied to fingerprints, and then used to generate levels for Super Mario Bros (Volz et al. 2018). In this approach CMA-ES samples the population of that latent space, made up of random noise vectors, and creates a covariance matrix to calculate how each variable in those vectors affects the fitness of the final output. In this way CMA-ES is used to learn the distribution of more fit output samples. LVE has also been extended to Latent Variable Illumination by using CMA-ME, a quality-diversity version of CMA-ES (Fontaine et al. 2020).

Representations of High-Dimensional Spaces

A number of different techniques have been used in the past to represent high dimensional information in a compressed manner. Often these methods are applied in order to use evolutionary methods on otherwise too-large spaces.

Some have used autoencoders to represent a high-dimensional environment observation as a smaller vector in order to train neural network controllers for playing Doom using CMA-ES. In this application, a neural network is used to learn how to efficiently compress the high-dimensional observation vector. CMA-ES is then used to search the space of the weights of the behavior-generating network for combinations that can successfully perform in the VizDoom ”health gathering” environment (Alvernaz and Togelius 2017).

Others have represented network weights as a set of coefficients to be transformed into weight values using an inverse Fourier-type transform (Koutník et al. 2013). This approach assumes a correlation between nearby weight coefficients. The researchers then evolve the weights of large recurrent neural network using this compression method for application to the TORCS racing simulator.

Further research has been done to propose novel methods for generating state representations in the context of separating representation learning from policy learning while using reinforcement learning to play Atari games. (Cuccu, Togelius, and Cudré-Mauroux 2018). These methods maintain a dictionary of centroids in the observation space which are then used for encoding, adding to that dictionary as training proceeds for online learning. In this case, the technique allows for playing of Atari games with small networks of only 6 to 18 neurons.

Methods

This project undertakes to explore the latent space of the TOAD-GAN level generator. Exploration is done using CMA-ES, which builds a covariance matrix and calculates how each variable in a noise vector affects the fitness of the final output in order to learn the distribution of output samples with higher fitness scores (Hansen and Ostermeier 2001; Hansen 2016). For the purposes of this exploration we define a number of fitness functions in order to find levels meeting specific criteria, and apply CMA-ES to find levels according to each of those criteria.

Random Network Projection

As described above, TOAD-GAN takes a single training sample, scales it down to a specified number of sizes, and trains a separate generator/discriminator pair at each size. In generating a sample, each of these generators requires a noise vector. Furthermore, TOAD-GAN uses a one-hot-encoded version of a tokenized level. Since Mario level 1

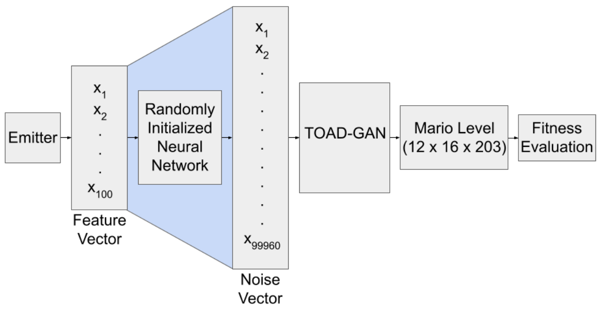


Figure 3: CMA-ES is used to explore the space of input to a randomly initialized neural network, which then allows for the exploration of the latent space of TOAD-GAN.

uses 12 tokens, each scaled Mario level is encoded with 12 channels. In the experiments publicized in (Awiszus, Schubert, and Rosenhahn 2020) based on Mario level 1, three down-scaled sizes were used, for a total of 4 trained generators and a final full-scale level size of 16 x 203 tokens encoded into 12 channels. The final size of the noise vector required given these parameters is 99,960. This large size makes calculating a covariance matrix infeasible due to resources available.

In order to explore this very large latent space with the computational resources available, an additional step is needed to translate a manageable number of features into a large enough noise vector to supply generators at 4 scales. For this purpose, a neural network is created with a single dense layer to take a smaller number of input features and output a vector of size (1, 99960). The number of input features used here is 100, since this is small enough that the amount of memory required by the covariance matrix is reasonable. This neural network is randomly initialized and not trained, so its output remains random.

To use this network, a generation of 100-element feature vectors is passed from an optimizing emitter to a fitness function. The fitness function then passes each of these vectors to the randomly initialized neural network, where a larger vector is generated from each. Each larger vector is passed to a trained set of TOAD-GAN generators. Each generator then takes a portion of the larger noise vector, and together the generators create a Mario level. Finally, the resulting level is then evaluated for fitness as defined below, as well as for various behavioral characteristics. The results of this evaluation for each level created in a generation are then passed back to the emitter, which updates the covariance matrix and optimizes the next generation of levels to attempt a higher fitness score as shown in figure 3.

TOAD-GAN is trained using noise vectors from a standard Gaussian distribution. Examination of the output of the randomly initialized neural network used here shows that it has a mean of 0 and a standard deviation between .5 and .6. One might expect that this noise, being different from the noise TOAD-GAN is trained on, might be less likely to result in generated levels that are similar to their training sample. In order to examine a wider sampling of the possible output of TOAD-GAN, experiments are run using the output of the random network as it is generated, with a standard deviation between .5 and .6, and also using the output of the random network after standardizing that output. In experiments run using standardized noise vectors, the output of the random neural network is passed through scikit-learn's StandardScaler, giving it a standard deviation of 1 and a mean of 0, before being passed to TOAD-GAN.

Fitness Evaluation

In these experiments, generated levels are examined first for playability. Previous researchers (Volz et al. 2018) have used the Mario AI competition framework[1] and its stable of successful trained playing agents to test for playability. Voltz et al. specifically referenced the A* agent by Robin Baumgarten that won the 2009 version of this competition. Here, the same agent is used. Experiments are run with percentage of level completed by the agent as a fitness measure, as well as with percentage of level left incomplete by the agent as a fitness measure.

It must be acknowledged that while this agent's completion of a level can be taken as proof of playability, there are also a number of levels that this agent is not able to complete, which a human player or another agent could finish. Many different agents exist, including some that are specifically made to be human-like (Shaker et al. 2013), but they are not integrated into the current version of the Mario AI Framework.

Generated levels are also evaluated for similarity to the training level. The different metrics listed below are used, and experiments are done seeking to both maximize and to minimize each metric, looking for levels that are either more similar to or more different from the training sample.

The first similarity metric used here is Tile-Pattern KL Divergence, which has been used in the past to measure the similarity of patterns in generated and original token-based levels (Lucas and Volz 2019), (Awiszus, Schubert, and Rosenhahn 2020). This metric adapts Kullback-Leibler Divergence to the domain of Mario levels (Lucas and Volz 2019). In the original TOAD-GAN experiments as published in (Awiszus, Schubert, and Rosenhahn 2020), results across a number of Mario levels and generated samples are found to have an average .33 Tile-Pattern KL divergence. In order to further explore the meaning of these results, Tile-Pattern KL Divergence is used here as a fitness measure.

Another metric used here is Hamming Distance, which is a measure of the number of tokens which differ between two levels (Torrado et al. 2019).

Normalized Compression Distance (NCD) is the third metric used here to compare two tokenized levels. NCD measures the similarity between two levels after both are compressed (Dahlskog et al. 2014). Here, the training level and output level are compressed using GZIP, and the resulting difference between the two levels is calculated according to the below, where $Z(x)$ is the size of a given tokenized level, x, represented as a string and then compressed.

[1]https://github.com/amidos2006/Mario-AI-Framework

Fitness	Std. Noise	Level	% Completed
Playability	no	1	100
Playability	yes	1	100
Playability	yes	3	99.57
Unplayability	no	1	5.80
Unplayability	yes	1	7.58
Unplayability	yes	3	6.04

Table 1: Average percentage of level completed by A* agent over 10000 Generations.

$$NCD(x, y) = \frac{Z(x+y) - min(Z(x), Z(y))}{max(Z(x), Z(y))}$$

Experiments are run for 10,000 generations, each optimizing for a different fitness measure and also recording a number of different behavioral characteristics. All experiments are run using Mario Level 1 as available in the Mario AI Framework as the training sample, and a 3-layer TOAD-GAN model that was trained for 4,000 iterations. Further experiments are then run using Mario Level 3 and a standardized noise vector as the training level so that results may be compared.

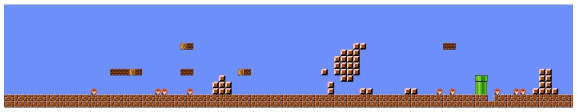
Results

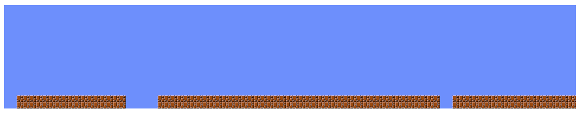
In these experiments, CMA-ES is used to search the latent space of TOAD-GAN first for playable and unplayable levels, then for levels that are more or less similar to the training sample, and finally for levels that exhibit visual characteristics not yet observed in generated samples. Each of these experiments are done using standardized noise vectors, and also using noise vectors that have not been standardized, so that the range of the GAN may be more fully explored.

Optimizing for Playability

In experiments examining generated levels for playability, the percentage of a level completed by the A* agent is used as a fitness measure, and the performance of the most fit member of each generation is recorded. Similarly, in experiments with percentage of level left incomplete by the A* agent as a fitness measure, the performance of the most fit member of each generation is recorded. Results can be seen in figure 1.

Results here indicate that CMA-ES is more successful when optimizing for playable Mario levels than unplayable Mario levels, implying that this level generator may be more likely to produce playable levels than not. This is consistently true whether the noise vector generated by the random network and input to TOAD-GAN is standardized or not, indicating that evolutionary search is successfully able to explore the latent space of the GAN using both methods. A visual inspection of the results tells us that there are a wide range of levels generated when optimizing for playability (Fig. 4), as well as a wide range of levels generated when optimizing for unplayability (Fig. 5), Given that range, further specifications may be needed to find usable levels.


(a) A level generated after training on Mario Level 1 and using the standardized output of the random neural network


(b) A level generated after training on Mario Level 1 and using the unstandardized output of the random neural network

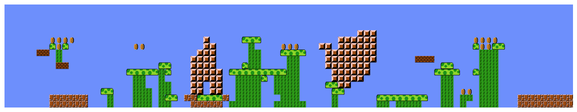
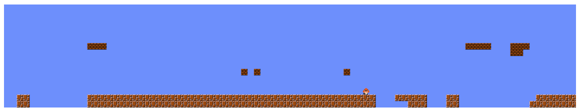
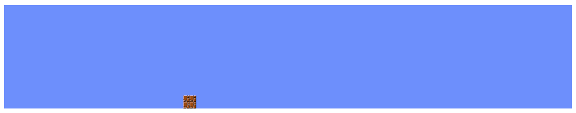

(c) A level generated after training on Mario Level 3 and using the standardized output of the random neural network

Figure 4: Levels generated by TOAD-GAN while optimizing for playability.


(a) A level generated after training on Mario Level 1 and using the standardized output of the random neural network


(b) A level generated after training on Mario Level 1 and using the unstandardized output of the random neural network

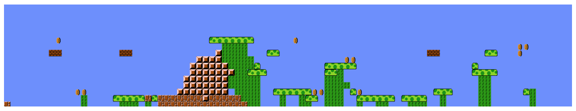

(c) A level generated after training on Mario Level 3 and using the standardized output of the random neural network

Figure 5: Levels generated while optimizing for unplayability.

Metric	Std. Noise	Level	Min	Max
Hamming Dist.	No	1	0.02	0.65
Hamming Dist.	Yes	1	0.02	0.35
Hamming Dist.	Yes	3	0.04	0.61
NCD	No	1	0.59	0.92
NCD	Yes	1	0.53	0.86
NCD	Yes	3	0.59	0.93
TPKL Div.	No	1	0.49	5.97
TPKL Div.	Yes	1	0.29	3.70
TPKL Div.	Yes	3	0.35	4.11

Table 2: Minimum and Maximum values found for each measure of similarity.

Optimizing for Similarity and Dissimilarity

The next set of experiments examines the similarity of generated levels to the training level using Tile-Pattern KL Divergence, Hamming Distance, and Normalized Compression distance as measures of similarity. A higher score for each of these measures indicates a less similar level, so experiments are run seeking to both minimize and maximize these scores. As for each of the fitness measures used here, one set of experiments is run using output directly from the random neural network to pass to TOAD-GAN, and another set is run first standardizing that output and then passing it to TOAD-GAN.

The maximum and minimum values found for each comparison metric are given in Table 2. These values demonstrate that standardizing the noise vector before passing it to TOAD-GAN results in levels that are more similar to the training sample according to every metric.

Heatmaps generated while maximizing each of these similarity metrics and recording the other two metrics as behavioral characteristics are shown in Figures 6a, 6b, 6c. These heatmaps are generated without standardizing the noise vector input to TOAD-GAN. In them, higher scores as indicated by brighter squares tend to be concentrated towards the upper right corner of each map, indicating that a higher score for one similarity metric often coincides with higher scores for the other two.

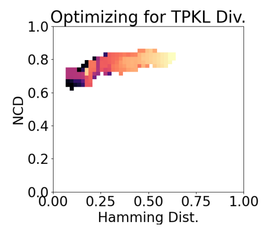
On examination, the most visually dissimilar levels generated based on level 1 while optimizing for each of these measures of similarity often appear to have a similar prevalence of pyramid blocks and may also be missing the identifiable ground or platform areas that would make them usable (Fig. 7). This is true of levels generated with and without standardizing the noise vector input to TOAD-GAN. Levels generated while maximizing dissimilarity for level 3 (Fig. 8) may appear similarly strange but do not necessarily include the same token types, as level 3 is made up largely of tree platforms, which are not present in level 1.

Optimizing Based on Visual Observations

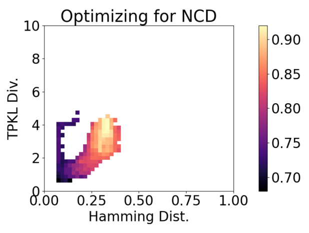
Once playability, Tile-Pattern KL Divergence, Hamming Distance, and Normalized Compression Distance have been explored, a visual inspection of generated levels is done, and a number of other characteristics are examined. Further experiments are then done to see if levels can be generated that exhibit characteristics not yet seen.

The first fitness measure considered here is the presence of token types not found in the training sample. It is expected that TOAD-GAN would generate levels using exclusively token types present in the training sample. This is demonstrated when in 10,000 generations of experiments, not a single level is generated that includes a token not found in the original level, confirming that this is one boundary of a TOAD-GAN generator trained on a given sample. This is consistent in experiments run using noise vectors that are standardized and noise vectors that are not, as well as in experiments run using level 1 as a training sample and those using level 3 as a training sample.

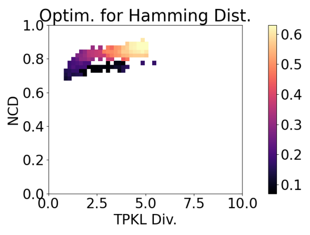
Another characteristic considered is the presence of par-



(a) Using high TPKL Divergence as fitness.

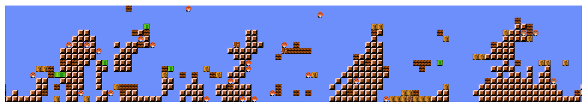


(b) Using high NCD as fitness.

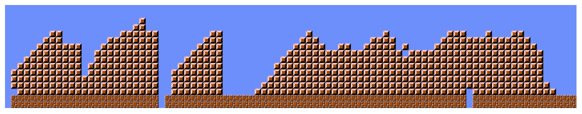


(c) Using high Hamming Distance as fitness.

Figure 6: Similarity measures as fitness

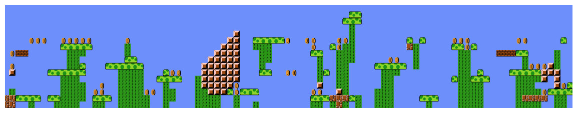


(a) A level generated with an unstandardized noise vector that has a normalized compression distance of 0.92 from its training sample.

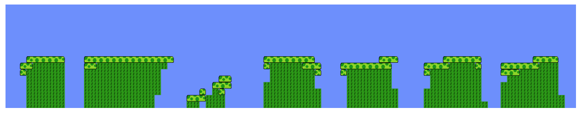


(b) A level generated with a standardized noise vector that has a hamming distance of 0.35 from its training sample.

Figure 7: Levels generated based on Mario world 1 level 1 that have some of the highest scores in differing from the training sample.

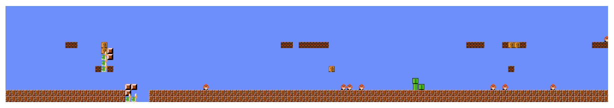


(a) A level generated with a standardized noise vector that has a normalized compression distance of 0.93 from its training sample.

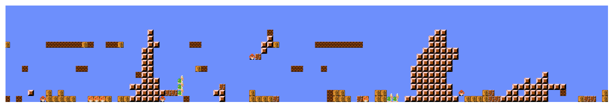


(b) A level generated with a standardized noise vector that has a Tile-Pattern KL Divergence of 4.19 from its training sample.

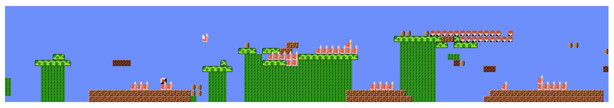
Figure 8: Levels generated based on Mario world 1 level 3 that have some of the highest scores in differing from the training sample.



(a) A level generated based on Mario level 1 from standardized noise vectors.



(b) A level generated based on Mario level 1 from an unstandardized noise vectors.



(c) A level generated based on Mario level 3 from a standardized noise vectors.

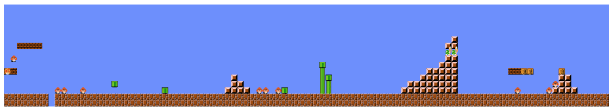
Figure 9: Using evolutionary search to optimize for number of Koopas.

ticular types of tokens, such as enemy tokens. The original Mario level 1 has 15 enemies present. Of these, 14 are Goombas and 1 is a green Koopa Troopa. CMA-ES is used here to explore the range of enemies that might be generated with this training sample.

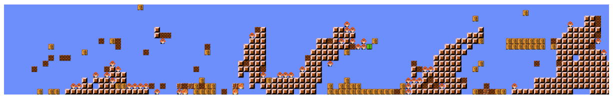
Experiments are done using total number of enemies as a fitness function, as well as a using total number of green Koopas as a fitness function. Results are shown in figure 9. When the noise vector fed to the GAN is not standardized, evolutionary search achieves a maximum of 9 green Koopas or 88 enemies. When the noise vector is standardized, levels with a maximum of 8 green Koopas or 65 enemies are found when using level 1 as the training sample, and no green Koopas but a maximum of 69 enemies are found when using level 3 as the training sample.

One can also observe that in the original Mario level 1, enemies are placed on ground tokens, platform tokens, and in midair. In order to see if TOAD-GAN can produce levels with new enemy placements, this characteristic is used as a fitness metric. Fitness here is defined as the number of enemies placed on pyramid tokens. Some of the resulting levels are shown in figure 10. There are a visible number of enemies placed on pyramid blocks in many of the resulting levels. This is true in levels generated based on level 1, but also in those generated based on level 3, showing that levels can be found that meet this criteria even when the training samples have differing numbers of pyramid blocks and types of enemies present.

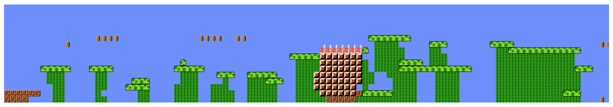
Finally, one of the most immediately apparent differences between levels generated by TOAD-GAN and original Mario levels is the presence of malformed elements, such as areas of the ground that have holes in them or pipes that start in midair. Both of these characteristics are then used as fitness functions for individual experiments. When holes in the ground are used as a fitness function, we define fitness as a value starting from 0 and negatively progressing with each ground token that is present in one of the two ground



(a) A level generated using a standardized noise vector based on Mario level 1

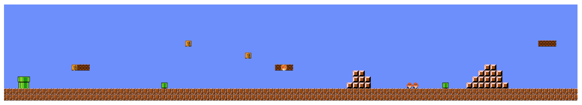


(b) A level generated using an unstandardized noise vector based on Mario level 1

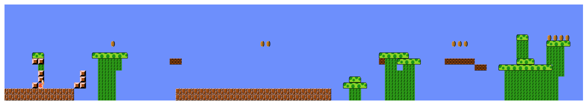


(c) A level generated using a standardized noise vector based on Mario level 3

Figure 10: Levels generated to optimize for number of enemies placed on pyramid blocks.



(a) A level produced using Mario level 1 as the training sample



(b) A level produced using Mario level 3 as the trainging sample

Figure 11: Levels generated while optimizing to avoid holes in the ground and pipes starting in midair, using standardized noise vectors.

levels but not the other. When pipes in midair are used as a fitness function, fitness is a value starting from 0 and negatively progressing with each pipe token that has a ”sky” token present immediately beneath it. Additional attempts are made to find levels that exhibit neither holes in the ground nor pipes starting in midair by combining both of these metrics and using that sum as the fitness measure.

A visual inspection of these results (Fig. 11) indicates that many of them are reasonably functional, and do not exhibit as many obvious visual malformations as seen in other experiments.

This of course does not cover the full range of behavior demonstrated by a trained TOAD-GAN level generator, but can provide some insight into what that behavior might be, where it might result in generated levels that are very different from the training level or for some reason unusable, and whether evolutionary search can be used to impose desired constraints on the space of the levels generated by a given GAN.

Discussion

In this work, CMA-ES is used to search the latent space of a trained TOAD-GAN model for more or less playable levels, as well as levels exhibiting higher or lower scores for Hamming Distance, Normalized Compression Distance, and Tile-Pattern KL Divergence as compared to the training level.

Results indicate that when noise vectors are standardized before being passed into TOAD-GAN, the generated levels are more similar to the training samples than when those noise vectors are not standardized. This is to be expected, as TOAD-GAN is trained on standardized noise vectors. However, allowing the search space to include noise vectors that are not standardized means that we are able to further explore the further reaches of a GAN's capabilities, and see some levels that are more dissimilar than expected to the training samples.

On observing the levels generated in these experiments, it is apparent that levels generated by TOAD-GAN always use the same token types as the training sample, and also produce enemy counts, types and placements similar to the training sample. However, they also exhibit many visual malformations such as pipes starting in midair or holes in the ground that would not be found in a human-designed Mario level. When CMA-ES is used to search for levels that contradict these observations, levels can be found that exhibit higher numbers of enemies in general, higher number of specific enemy types, and new enemy placements. Levels are also found that exhibit lower numbers of holes in the ground and pipes starting in midair. This indicates that evolutionary search can be used alongside GANs to confirm whether those GANs can generate levels that meet a specified criteria and to and to find those levels.

It is important to note that some bias towards the tokens and patterns of the training level was expected to be found in the trained TOAD-GAN model. Any successfully trained model will by necessity be biased. The question is what exactly the bias consists in, and the methods presented here can reveal this.

Conclusion

In this paper, we employ evolutionary search to illuminate the latent space of a trained level generator. In doing so we offer a useful method for understanding and characterizing what a generator has learned. The methods proposed here could be applied to understand and contrast other generators of functional content, as well as to search for levels meeting specific criteria in the latent space of a generator.

To make the search space feasible here, a random projection in latent space is used. This is a technique that allowed us to combine two otherwise incompatible previous methods to gain deeper insight into one of them. It is also a technique that may have wider applications in fields such as evolutionary search, where some methods may not be suitable for searching a larger space. This project can suggest a new way to address those problems, thus opening the door for future work.

References

Alvernaz, S.; and Togelius, J. 2017. Autoencoder-augmented Neuroevolution for Visual Doom Playing. *CoRR* abs/1707.03902. URL http://arxiv.org/abs/1707.03902.

Awiszus, M.; Schubert, F.; and Rosenhahn, B. 2020. TOAD-GAN: Coherent Style Level Generation from a Single Example. *CoRR* abs/2008.01531. URL https://arxiv.org/abs/2008.01531.

Bontrager, P.; Togelius, J.; and Memon, N. D. 2017. Deep-MasterPrint: Generating Fingerprints for Presentation Attacks. *CoRR* abs/1705.07386. URL http://arxiv.org/abs/1705.07386.

Cuccu, G.; Togelius, J.; and Cudré-Mauroux, P. 2018. Playing Atari with Six Neurons. *CoRR* abs/1806.01363. URL http://arxiv.org/abs/1806.01363.

Dahlskog, S.; Horn, B.; Shaker, N.; Smith, G.; and Togelius, J. 2014. A Comparative Evaluation of Procedural Level Generators in the Mario AI Framework. In *Proceedings of Foundations of Digital Games*.

Fontaine, M. C.; Liu, R.; Togelius, J.; Hoover, A. K.; and Nikolaidis, S. 2020. Illuminating mario scenes in the latent space of a generative adversarial network. In *Proceedings of AAAI*.

Green, M. C.; Mugrai, L.; Khalifa, A.; and Togelius, J. 2020. Mario Level Generation From Mechanics Using Scene Stitching. *CoRR* abs/2002.02992. URL https://arxiv.org/abs/2002.02992.

Hansen, N. 2016. The CMA Evolution Strategy: A Tutorial. *CoRR* abs/1604.00772. URL http://arxiv.org/abs/1604.00772.

Hansen, N.; and Ostermeier, A. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation* 9(2): 159–195.

Karras, T.; Aila, T.; Laine, S.; and Lehtinen, J. 2017. Progressive Growing of GANs for Improved Quality, Stability, and Variation. *CoRR* abs/1710.10196. URL http://arxiv.org/abs/1710.10196.

Khalifa, A.; Bontrager, P.; Earle, S.; and Togelius, J. 2020. PCGRL: Procedural Content Generation via Reinforcement Learning. *CoRR* abs/2001.09212. URL https://arxiv.org/abs/2001.09212.

Koutník, J.; Cuccu, G.; Schmidhuber, J.; and Gomez, F. 2013. Evolving Large-Scale Neural Networks for Vision-Based Reinforcement Learning. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, 1061–1068. New York, NY, USA: Association for Computing Machinery. ISBN 9781450319638. doi:10.1145/2463372.2463509. URL https://doi.org/10.1145/2463372.2463509.

Liu, J.; Snodgrass, S.; Khalifa, A.; Risi, S.; Yannakakis, G. N.; and Togelius, J. 2020. Deep learning for procedural content generation. *Neural Computing and Applications* 1–19.

Lucas, S. M.; and Volz, V. 2019. Tile Pattern KL-Divergence for Analysing and Evolving Game Levels. *CoRR* abs/1905.05077. URL http://arxiv.org/abs/1905.05077.

Shaham, T. R.; Dekel, T.; and Michaeli, T. 2019. SinGAN: Learning a Generative Model from a Single Natural Image. *CoRR* abs/1905.01164. URL http://arxiv.org/abs/1905.01164.

Shaker, N.; Togelius, J.; and Nelson, M. J. 2016. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer.

Shaker, N.; Togelius, J.; Yannakakis, G. N.; Poovanna, L.; Ethiraj, V. S.; Johansson, S. J.; Reynolds, R. G.; Heether, L. K.; Schumann, T.; and Gallagher, M. 2013. The turing test track of the 2012 Mario AI Championship: Entries and evaluation. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, 1–8. doi:10.1109/CIG.2013.6633634.

Snodgrass, S.; and Ontañón, S. 2014. Experiments in Map Generation using Markov Chains. In *Proceedings of Foundations of Digital Games*.

Summerville, A.; Philip, S.; and Mateas, M. 2021. MCMCTS PCG 4 SMB: Monte Carlo Tree Search to Guide Platformer Level Generation. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 11(3): 68–74. URL https://ojs.aaai.org/index.php/AIIDE/article/view/12816.

Summerville, A.; Snodgrass, S.; Guzdial, M.; Holmgård, C.; Hoover, A. K.; Isaksen, A.; Nealen, A.; and Togelius, J. 2018. Procedural content generation via machine learning (PCGML). *IEEE Transactions on Games* 10(3): 257–270.

Torrado, R. R.; Khalifa, A.; Green, M. C.; Justesen, N.; Risi, S.; and Togelius, J. 2019. Bootstrapping Conditional GANs for Video Game Level Generation. *CoRR* abs/1910.01603. URL http://arxiv.org/abs/1910.01603.

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. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 221–228.