Studying the Effects of Training Data on Machine Learning-Based Procedural Content Generation

Sam Snodgrass  
Drexel University  
Philadelphia, PA USA  
sps74@drexel.edu

Adam Summerville  
University of California, Santa Cruz  
Santa Cruz, CA USA  
asummerv@ucsc.edu

Santiago Ontañón  
Drexel University  
Philadelphia, PA USA  
santi@cs.drexel.edu

Abstract

The exploration of Procedural Content Generation via Machine Learning (PCGML) has been growing in recent years. However, while the number of PCGML techniques and methods for evaluating PCG techniques have been increasing, little work has been done in determining how the quality and quantity of the training data provided to these techniques affects the models or the output. Therefore, little is known about how much training data would actually be needed to deploy certain PCGML techniques in practice. In this paper we explore this question by studying the quality and diversity of the output of two well-known PCGML techniques (multi-dimensional Markov chains and Long Short-term Memory Recurrent Neural Networks) in generating Super Mario Bros. levels while varying the amount and quality of the training data.

1 Introduction

Procedural content generation (PCG) studies the algorithmic creation of content (e.g., maps, textures, music, etc.), often for video games. Recently, interest in PCG via Machine Learning (PCGML) (Summerville et al. 2017b) has grown and spawned many level generation techniques (Snodgrass and Ontañón 2014; Dahlskog, Togelius, and Nelson 2014; Summerville, Philip, and Mateas 2015; Guzdial and Riedl 2016). However, while there is work in evaluating level generators, there has not yet been any work in exploring the effects of the quality and quantity of training data on these PCGML techniques. In this paper we explore how using varying amounts and varying quality of training data affects the quality and diversity of the generated levels.

Previously, PCGML techniques have been trained using all of the training data available for the domain at hand. However, there may be scenarios where limited training data is available, or where training data needs to be created specifically for a new domain. In these instances it is important to be able to determine how much training data is needed, and which techniques could be used with the amount and quality of the training data available. The contribution of this work is four-fold:

1. A method for measuring the quality of training data.
2. A method for measuring the level of plagiarism for a PCGML system.
3. A method for measuring the expressivity of a generative system.
4. An experiment examining the effect of the amount and quality of training data on a PCGML system.

The remainder of the paper is organized as follows. First, we formulate our problem statement. In Section 2 we discuss related work, including various PCGML techniques and evaluation approaches. Next, in Section 3, we describe how we represent our training data, and the level generation techniques we use for our experiments. Section 4 explains our experimental evaluation, including how we determine level quality, and how we evaluate our output and models, and finally our results. The paper closes with conclusions and lines of future work.

1.1 Problem Statement

The specific question we address in this paper is how well a PCGML technique performs when provided with varying amounts and quality of training data. Specifically, understanding the effects of using different amounts and types of training data on the quality and diversity of generated levels.

In order to address this question, we propose a way to measure the quality of training data, and use it to study not just the effect of the amount of training data, but also how much the quality of training data affects the PCGML models under consideration.

2 Related Work

There have been several PCGML approaches applied to the domain of platter game level generation. For example, Snodgrass and Ontañón used Markov models to sample levels for Super Mario Bros, Lode Runner, and Kid Icarus (Snodgrass and Ontañón 2016b). Summerville and Mateas (Summerville and Mateas 2016) used long short-term memory neural networks, and Guzdial and Riedl (Guzdial and Riedl 2016) used Bayesian networks, both to generate Super Mario Bros. levels. However, these approaches all used the amount of training data that was available to them, without performing analysis on whether it was the appropriate amount of training data (i.e. whether their models needed all the data provided, if their models could benefit from even more training data, or if higher quality data was needed).
In addition to PCGML techniques, there has also been a lot of work in evaluating content generators and their output. Smith and Whitehead (Smith and Whitehead 2010) explored using various evaluation metrics on output levels in order to measure the expressive range of a generator. Cannosa and Smith (Canossa and Smith 2015) created a larger list of metrics for capturing the expressivity of generators, but did not perform any experimentation with said metrics. Mariño et al. (2015) performed an evaluation of level evaluation metrics, which was expanded on by Summerville et al. (Summerville et al. 2017a). However, these approaches to evaluation have not been used to determine the effects of the training data on PCGML techniques. Snodgrass and Ontaño (Snodgrass and Ontaño 2016b) touched on guiding their model by selecting the training levels, but did not perform an in-depth analysis of the effects of different amounts and types of training data on their models.

While we do not know of any work examining the effect of training data size on a generative machine learning model, there has been research into the effect of dataset size for classification systems. Brain (Brain 1999) examined a number of different classification systems (Naive-Bayes (Kononenko 1993), C4.5 decision trees (Quinlan 2014), MultiBoost (Webb 2000)) on a number of training sets. He decomposed the error into the bias and variance, and generally found that the increase in data reduced variance but typically did not effect bias greatly. Zhu et al. (Zhu et al. 2016) examined a number of mixture models under different conditions of data size and quality. They found that, for a number of models, hyperparameters needed to be tuned for the size of the dataset, and without this tuning an increase in size can result in decreasing performance. They also found that while increasing in size generally enhanced performance it did so at exponentially diminishing returns, indicating that it would either take substantially more data or models better equipped to handle said data. Finally, they found that for their image classification task that higher quality features did not result in markedly different performance.

3 Level Generation Methods

In this section we discuss the level representation and the two PCGML techniques we will use for the remainder of the paper to explore the effects of training data. We chose to use two techniques that have previously been shown to be able to produce high quality content, the LSTM approach of Cannosa and Smith (Canossa and Smith 2015) created a larger list of metrics for capturing the expressivity of generators, but did not perform any experimentation with said metrics. Mariño et al. (2015) performed an evaluation of level evaluation metrics, which was expanded on by Summerville et al. (Summerville et al. 2017a). However, these approaches to evaluation have not been used to determine the effects of the training data on PCGML techniques. Snodgrass and Ontaño (Snodgrass and Ontaño 2016b) touched on guiding their model by selecting the training levels, but did not perform an in-depth analysis of the effects of different amounts and types of training data on their models.

While we do not know of any work examining the effect of training data size on a generative machine learning model, there has been research into the effect of dataset size for classification systems. Brain (Brain 1999) examined a number of different classification systems (Naive-Bayes (Kononenko 1993), C4.5 decision trees (Quinlan 2014), MultiBoost (Webb 2000)) on a number of training sets. He decomposed the error into the bias and variance, and generally found that the increase in data reduced variance but typically did not effect bias greatly. Zhu et al. (Zhu et al. 2016) examined a number of mixture models under different conditions of data size and quality. They found that, for a number of models, hyperparameters needed to be tuned for the size of the dataset, and without this tuning an increase in size can result in decreasing performance. They also found that while increasing in size generally enhanced performance it did so at exponentially diminishing returns, indicating that it would either take substantially more data or models better equipped to handle said data. Finally, they found that for their image classification task that higher quality features did not result in markedly different performance.

3.1 Level Representation

We used the tile-based representation employed by the Video Game Level Corpus (VGLC) (Summerville et al. 2016), an open repository of training data for PCGML techniques. In this representation, a level is represented by an $h \times w$ two-dimensional array, $M$, where $h$ is the height of the level, and $w$ is the width. Each cell of $M$ is mapped to an element of $T$, the set of tile types corresponding to elements within the game levels. Figure 1 shows a section of a Super Mario Bros. level (left) and how we represent that level (right).

Figure 1: A section of Super Mario Bros. level (left) and how we represent that level for use as training data for our techniques (right).

A new level is sampled tile-by-tile according to the trained conditional probability distribution and the previous configuration of tiles. Figure 2 shows the the network structures used by the MdMC approach. Note that $ns_0$ is the starting network structure, which falls back to $ns_2$, and so on.

3.2 Markov chain-based Level Generation

Markov Chains Markov chains (Markov 1971) model transitions between states over time via a conditional probability distribution (CPD), $P(S_t | S_{t-1})$. The set of previous states that influence the CPD are the network structure.

Multi-dimensional Markov chains (MdMCs) are an extension that allow any surrounding state in a multi-dimensional graph to be included in the network structure. By redefining what a previous state can be in this way, the model can more easily capture relations from two-dimensional training data.

Training Training an MdMC requires a network structure and training data, and simply consists of estimating the conditional probability of a given tile type occurring given each configuration of previous tile types according to the network structure. We do this by counting the number of times each tile appears after each previous configuration of tiles.

Sampling A new level is sampled tile-by-tile according to the trained conditional probability distribution and the previous tile configuration at each position. While sampling, the model might encounter a tile configuration that was not seen during training (an unseen state). Unseen states are undesirable, since there is no training data for them, and thus the model will have to generate a tile at random, which often
leads to future unseen states. In order to avoid unseen states, 
two strategies are used: look-ahead and fallback models.

The look-ahead process samples (and resamples a fixed 
number of tiles in advance, trying to ensure that no unseen 
state is reached. If the look-ahead fails and a tile cannot be 
found that results in no unseen states, then the model falls 
back to an MdMC trained with a simpler network structure.
For our experiments we use \( n_{85} \) in Figure 2 as the initial 
network structure, which falls back to \( n_{82} \), and then to \( n_{81} \), 
and finally to \( n_{80} \). This approach is outlined in more detail 
in (Snodgrass and Ontaño 2016b).

In this work we use a state-of-the-art MdMC-based level 
generation approach developed by Snodgrass and Ontaño 
called Violation Location Resampling (VLR) (Snodgrass 
and Ontaño 2016a). VLR can optionally accept a set of 
constraints for the sampled levels to adhere to. At a high 
level, when VLR samples a new level, it first generates a 
new level as the standard MdMC approach would (described 
above), but then, any sections of the level that violate pro-
vided constraints are resampled until all constraints are satis-
fied. In our experiments we pass VLR two constrains: 1) 
checks the playability of the level using an \( A^* \) agent and re-
turns any unplayable sections; 2) checks for any malformed 
structures (e.g., a pipe missing pieces in Super Mario Bros.).

3.3 Long Short-term Memory RNNs

LSTMs  Recurrent Neural Networks (RNNs) operate in a 
manier similar to standard neural networks (i.e. they are 
trained on data and errors are back-propagated to learn 
weight vectors). However, in an RNN the edge vectors are 
not just connected from input to hidden layers to output, they 
are also connected from a node to itself across time. This 
means that back-propagation occurs not just between differ-
ent nodes, but also across time steps.

LSTMs are a neural network topology first proposed by 
Hochreiter and Schmidhuber (Hochreiter and Schmidhuber 
1997) for the purpose of eliminating the vanishing gradient 
problem found in RNNs. LSTMs mitigate the problem via 
nodes that act as a memory mechanism, telling the network 
when to remember and when to forget. The LSTM architec-
ture can be seen in Figure 3.

Training  Torch7 (Collobert, Kavukcuoglu, and Farabet 
2011) was used to train the networks in our experiments, 
based on code from Andrej Karpathy (Karpathy 2015) using 
parameters previously optimized in (Summerville and 
Mateas 2016). Specifically, we trained on sequences of 200 
tiles at a time, in a network with 512 LSTM cells per layer 
and 3 layers. To fight overfitting, dropout was aggressively 
used, with 80% of LSTM cells being dropped at each train-
ing instance.

Following work from Summerville and Mateas (Sum-
merville and Mateas 2016) we used a “Snaking” path (it 
starts from the bottom left, goes bottom-to-top, flips direc-
tions going top-to-bottom, flips, etc.) and “Depth” informa-
tion (a special meta-tile is inserted at the top of a column 
one per each ten columns into the level).

Sampling  To prime the network to begin generation, an 
input seed is passed in with 3 empty columns with a single 
ground tile at the bottom. The generator is then sampled un-
til an end-of-level termination character is found, with each 
newly sampled tile being used as the input for the next step in 
the auto-regression process. Note, we do not enforce con-
straints when sampling with the LSTM.

4 Experimental Evaluation

This section first describes the domain used for experimen-
tation, how we assess the quality of training data, our experi-
mental set-up, our evaluation metrics, and finally our results.

4.1 Domain

We perform our experiments using Super Mario Bros. 
(SMB) as our domain. We chose SMB because of its wide 
use in the field of level generation. Its common use allows 
us to leverage previous work on evaluation metrics, and also 
makes the effects of the training data easier to understand.

We represent Mario levels using a set of 35 tile types. 
Each tile type corresponds to either an enemy type or an 
object in the game, such as blocks or pipes. Note, this rep-
resentation is more expressive than previously used repre-
sentations as it differentiates between the types of enemies, 
and represents several objects that have previously been ig-
nored, such as springs and moving platforms. We use a total 
of 29 levels from Super Mario Bros. and Super Mario Bros. 
2: The Lost Levels. Figure 1 (right) shows a section of an 
SMB level represented using this tile set.

4.2 Training Data Quality

In addition to evaluating the performance of the two chosen 
PCGML methods when using increasing amounts of train-
data. In this paper, we also evaluate the methods using 
training data of varying “quality.”

“Quality” is a subjective term, thus, in this paper we con-
sider levels that are more uniform to be of “lesser quality” 
than levels with move variety. We thus approximate qual-
ity by computing the entropy of the training levels through 
their high-level structures. That is, we split the training lev-
els into \( 4 \times 4 \) tile sections. We then perform \( k \)-medoids on
those sections with \(k = 30\). For the \(k\)-medoids distance metric, we find the positioning of two sections that yields the most overlap in tile types between the sections, and weight that by the area of the overlapping sections. The idea is that this metric provides us with a measure of how structurally similar two sections are. Once the clusters are computed, we represent each level as a histogram containing the number of \(4 \times 4\) level sections belonging to each cluster. Finally, we compute the entropy of those histograms (Wallis 2006), and assume that a higher entropy corresponds to more information in that level, and thus a higher quality training level.

4.3 Training Sets

In order to evaluate the effects of both the quality and quantity of training data on the chosen models, we devise several sets of training data. We first order the training levels from most to least entropic. We then train separate models using the first 16 columns of the most entropic level, using the first 32 columns, using the first 64 columns, using the first 128 columns, using the most entropic level in its entirety, using the two most entropic levels, etc. We then repeat this process using the least to most entropic ordering of the levels. In total, we train 66 MdMC models and 66 LSTM models.

4.4 Evaluation Metrics

We evaluate the levels sampled by our systems using both standard level evaluation metrics and metrics that explore the expressiveness of the systems given the training data.

- **Linearity**: This measures how well the platforms in the level are approximated with a best fit line (Smith and Whitehead 2010). It returns the sum of distances of each solid tile type (i.e., not empty, not enemies) from the best-fit line, normalized by the level length.

- **Leniency**: This approximates the difficulty of the level by summing the gaps (weighted by length) and enemies (weighted by 0.5), and normalizing by the level length (Smith and Whitehead 2010).

- **Enemy Sparsity**: This measures the horizontal spread of enemies through the level by taking the average distance of enemies from the average of enemy \(x\) positions in the level (Summerville et al. 2017a). A large Enemy Sparsity value means enemies are scattered around the level, whereas a low value means enemies are grouped together.

- **Kernel Density Estimation**: The *expressive range* (Smith and Whitehead 2010) of a generator has typically been thought of as a visualization of the metric space covered by the generated content. Most commonly, this has been visualized as a heatmap in 2 dimensions (linearity and leniency, classically (Van der Linden, Lopes, and Bidarra 2013; Snodgrass and Ontaño 2015)) although some have done up to 8 dimensions (2 at a time) (Summerville and Mateas 2016). However, in the literature it is common to refer to the “width” of the expressive range, but this has yet to be done in anything beyond a qualitative visual assessment (Smith et al. 2011).

    From this point on we will use *volume* as the measure we care about, as width is problematic as it is a linear dimension. For instance, a generator that always produced perfectly linear levels that had a very wide range in leniency would still be unlikely to be thought of as very expressive, given that it is completely lacking in 1 dimension. To this end, we will consider the \(n\)-dimensional volume (e.g., area in 2D, standard volume in 3D, etc.) of the generated metric space to be the *size* of the expressive range.

To calculate this volume, we use Kernel Density Estimation (KDE) as calculated by the “ks” R package (Duong and Hazelton 2005). KDE determines a non-parametric function of the density of a sampled space, similar to the binning process of a histogram, but typically smooth given the use of a Gaussian kernel. Figure 4 shows the calculated Linearity and Leniency for the 1000 levels generated by MdMC Most-to-least 29 Level Generator as grey circles. The density estimate is visualized by black contour lines. We then threshold this density estimate for points greater than 0, which we take to be the boundaries of the expressive range. We then form an \(n\)-dimensional grid, and count the number of bins that lie within the expressive \(n\)-volume, multiplying the count with the volume of a single bin. For our experiments we compute the expressive volume of our models using linearity, leniency, and enemy sparsity for the density estimation.

- **Plagiarism**: This measures the percentage of an output level that is directly copied from the training levels. We compute this by first splitting the levels in the training set into overlapping sections of \(n\) columns and removing any duplicates. We call this set of level sections \(T_n\). We then split an output level into overlapping sections of \(n\) columns, but do not remove duplicates. We call this series of level sections \(L_n\). We compute the plagiarism of a level by counting how many \(l \in L_n\) are also in \(T_n\). We then determine the number of columns from the output level that make up the sections that are plagiarized (accounting for overlapping columns). The value returned is the percentage of columns plagiarized in the output level. Notice, we
do not simply count how many individual columns are plagiarized directly because we are interested in seeing how large the sections of plagiarism are as well as how much of the level is plagiarized. For example, given two levels, one of which has 50% of columns plagiarized with $n = 4$, and the second of which also has 50% of columns plagiarized, but with $n = 20$, then we consider the second to be more plagiarized than the first level, because of the large amount of continuous plagiarism (i.e. a section of 20 columns copied directly from the training data is considered worse than 5 separate 4 column sections).

4.5 Results

Figure 5 shows the results of the expressive volume calculations. Notice, what we care about here are the ratios between the expressive volumes of the models, as the actual volume scales will vary for different domains. In general, we see that a small amount of data (< 3 levels) results in a very small expressive volume, which is as we would expect given that there isn’t much variation in the supplied data. A notable exception is the MdMC when using the most to least ordering of levels. This model’s expressive volume levels off after only 1 level. Surprisingly, additional data does not increase the expressive volume of any of the models after about 5 training levels. In the range with sufficient data (> 4 levels) the LSTMs generally have a larger expressive volume (27% greater), but all generators have a much smaller expressive volume than that of the original levels, which is 50% larger than the next closest generator (LSTM Most-to-least 8 levels). This larger volume is present in all of the individual metrics, meaning it is not just a failing in any one particular aspect. We also note that after reaching 5 levels worth of data, the information density of those levels does not effect the expressive volume of the models. Perhaps, this is due to the fact that the variety found in multiple levels, even those of relatively low information content, exceeds that of any one level. Of course, there are certainly degenerate counterexamples (e.g., 5 empty levels would provide no worthwhile information), but in any reasonable practical application it is more important to acquire a sufficient amount of data.

Figure 6 shows how much the MdMC and LSTM models trained with various amounts of training data plagiarize from the training levels. We also display the plagiarism results between the training levels. We compute this by treating each level individually as the output level, and treating the remaining 28 training levels as the training data. Then, for each level we compute the plagiarism of that one level against the other 28 levels. We do this for each training level, and average the values.

An interesting result we see when using the most to least order (top-left and bottom-left) is that the percentage of plagiarism and the size of plagiarized sections increases with the amount of training data. We believe this is due to the fact that as the amount of training data increases, the number of common structures increases, which makes it more likely for something that is sampled to be present in the training data. Alternatively, when using the least to most ordering (top-right and bottom-right), we see more mixed results with fewer training levels resulting in a higher percentage of plagiarism. We believe this is due to the simplicity of those few training levels that are being used. That is, because there are so few differing structures in the initial training levels, the models are likely to copy those few simple structures.

When comparing between the MdMC and LSTM approaches, we see that the LSTM tends to plagiarize a higher percentage of columns and larger sections than the MdMC approach. This is to be expected as the LSTM considers a larger amount of context when generating, keeping four full columns worth of tiles in working memory, which leads to learning of larger structures (and thus more plagiarism of such structures). This is exemplified when only one training level is used, and upwards of 150 column sections are plagiarized from the given training level, due to the ability of the LSTM to nearly memorize the entire level. However, we can see that when only 5 training levels are used, the plagiarism decreases drastically from the one level model.

Furthermore, we can see that the MdMC when trained with all the available training levels (29), plagiarizes a lower

Figure 5: The expressive volume for the 2 different techniques (MdMCs [triangles] and LSTMs [dots]) and 2 different progressions (most-to-least entropy [red] and least-to-most [blue] entropy). We see that the LSTMs in general have higher expressive volume, due to larger variability in the Leniency. The expressive volume of the original levels is shown for reference.
percentage of columns from the training data than the training levels plagiarize from each other, while the LSTM when trained with all levels plagiarizes a higher percentage. Notice, both models plagiarize around the same maximum section size in this case. While the LSTM trained with all levels does plagiarize a higher percentage than the original levels, nearly all trained models (except for LSTM 1 level and MdMC Least-to-Most 16 Columns) are roughly comparable or below the plagiarism of the original to themselves, indicating that neither approach suffers greatly from plagiarism when a sufficient amount of data present.

From the plagiarism and volume estimate results, we see that using all of the available training data is not necessary, and in some cases may even hinder the result. Figure 5 shows that the MdMC’s expressiveness levels off around 6 levels with the least to most ordering and after only 1 level with the most to least ordering, while the LSTM expressiveness levels off after around 10 training levels in both cases. Additionally, the expressiveness of the LSTM models doesn’t change based on the ordering of the training levels, while it does for MdMCs for small amounts of training data. Finally, training with more information dense levels can reduce the amount the models plagiarize from the training data.

5 Conclusions and Future Work

In this paper we explored the effects of the amount and quality of training data on two machine learning-based procedural content generation approaches, multi-dimensional Markov chains (MdMCs) and long short-term memory recurrent neural networks (LSTMs). We found that despite most published results naively using all of the available training levels, there can be benefits to more carefully choosing a smaller subset of the available training data. Specifically, we found that using a smaller subset of training levels (7 for the MdMC and 10 for the LSTM) did not negatively affect the expressiveness of either model. This has important implications for the practical applicability of PCGML techniques, since it means that we do not need large amounts of training data to make them work in a given domain. Additionally, we found that by using a subset of levels with high-entropy in the level structures, the models can be made to plagiarize less from the training levels.

In the future, we would like to investigate these principles in more domains. In particular, more difficult domains such as Lode Runner, which involves solving puzzles. We would be interested in exploring how various training data affects other level generators. Lastly, we would like to devise more formal methods for determining the quality of training data in a particular game for a particular model, which would allow our results to be more easily applied to new models.

References

Fourth Australian Knowledge Acquisition Workshop.


Snodgrass, S., and Onaño, S. 2016b. Learning to generate video game maps using Markov models. *IEEE Transactions on Computational Intelligence and AI in Games*.


