Neural Cellular Automata for Diverse Tree Growing

Cedric Caspar [1] **Nicolas Blumer** [1] **Oleh Kuzyk** [1] **Piyushi Goyal** [1]

Abstract

Neural Cellular Automata (NCA) have gained some traction in learning captivating 2D and 3D structures while being very parameter efficient. This paper proposes an extension to the dynamics of NCA, exploring the capabilities and behaviors that arise when training NCA on several 3D structures, in our case trees from Minecraft. To this end, we examined several methods to create additional inputs for the NCAs, from (variational) auto-decoders, feeding in noise, or adding the ID of the tree to help adapt NCAs to a multi-sample environment and evaluate their potential. Remarkably, we observed that NCA, despite having access only to local information, can infer the input size and learn distinct structures based on it.The results obtained show that using VADs leads to the best convergence in the loss. Code.

1. Introduction

The fundamental appeal of Cellular Automata (CA) lies in their simplicity, yet the capacity to generate intricate patterns and behaviors. While traditional CA models have been extensively studied, the integration of depth through multiple layers, termed deep cellular automata, introduces a novel and relatively unexplored dimension to this paradigm. Cellular automata (CA) have long served as a versatile framework for simulating complex systems and exploring emergent phenomena. NCAs are extremely parameter efficient as every cell (voxel in our case) has the same weights and are usually implemented via CNNs (Gilpin, 2019). his parameter efficiency is a key motivation for their selection, aligning with our objective of achieving high efficiency. This weight sharing also enables working with different input sizes. Additionally, the models are very shallow, when compared to more conventional deep learning architectures. The modeling prowess arises from applying the model over multiple time steps. This research seeks to delve into the dynamics of deep CA growth, by extending existing NCA models from (Sudhakaran et al., 2021). They train NCAs to generate or even regenerate structures in the particular setting of Minecraft, from buildings and trees to redstone machines. Since any particular model is only trained to generate one structure at a time, there is a limited use case for such models, as they have basically the same function as a building plan. We extend their idea to make NCA generalize to several structures (trees) making them a potentially useful tool for game designers. The aim is to have an NCA that can be reused to create more interesting, diverse worlds without the need for a lot of manually-designed trees. The resulting method can then be applied to other structures like houses or furniture.

Figure 1. Example of the generated tree, Sampled from VAD trained on the large(20 trees) dataset

2. Literature Review

Some existing approaches for 3D object generation include GANs and implicit methods. For instance, (Wu et al., 2016; Kench & Cooper, 2021; Zhu et al., 2018) have demonstrated the power of Generative Adversarial Networks (GANs). Moreover, other works such as from (Chabra et al., 2020; Zheng et al., 2021; Xu et al., 2019; Liu & Liu, 2021; Park et al., 2019) have exhibited the use of implicit methods for generating 3D shapes.

Our class of methods, Cellular Automata (CA), had first been suggested in the 1960s by (Neumann & Burks, 1966). The work suggests applying the biological process of generation and regeneration to artificial generation using computing machines. At every time step, the status of each cell can be represented as a state, which is then transitioned into the next state per the update rule.

The next advancement to this came when (Li & Yeh, 2002)

[1]EqualContribution, Department of Computer Science, ETH Zurich, Switzerland. Correspondence to: Cedric Caspar <ccaspar@student.ethz.ch>.

Figure 2. Evolution of a single tree over time, using a variational auto-decoder (embedding dimension = 4), trained on 8 trees

suggested that the update rules in CAs can be replaced with neural networks, thereby giving it the name of Neural Cellular Automata (NCA). Quite recently, the work presented by (Mordvintsev et al., 2020) extends NCA to generate a fixed image only from a single pixel. The core concept behind NCA involves employing a Convolutional Neural Network (CNN) to acquire the transition rule for a 2D Cellular Automaton (CA) characterized by a continuous and multi-dimensional state space, conceptualized as an image. Further development was suggested by (Ruiz et al., 2021) where the NCA could generate multiple images and objects instead of just one. This was done by making the weight of the network a function of the target. (Zhang et al., 2021) generalized this model further by adapting it to an arbitrary starting point/pixel.
Thus, the models could now generate multiple images without keeping a fixed single pixel. The work by (Sudhakaran et al., 2021) extends the application of NCAs into the 3D realm, incorporating 3D convolutions within the proposed neural network architecture. Minecraft serves as the chosen environment for their automaton, given its rich 3D voxel environment and structures. Their results showcase the remarkable capability of NCAs to grow complex entities, including castles, apartment blocks, and trees, some comprising over 3,000 blocks.
(Zadeh et al., 2021) introduces the Variational Auto-Decoder (VAD), which in contrast to the Variational Auto-Encoder (VAE), omits the encoder component, utilizing a proposed efficient inference method for learning generative models with reduced computational cost. This method uses no additional parameters other than the embeddings themselves, making the method very efficient, which is why it is used in our approach.

3. Methodology

Our primary goal is to use limited or few parameters to generate a variety of trees for the Minecraft environment. Where (Sudhakaran et al., 2021) uses models trained only on a single tree instance in the dataset, our objective is to augment the existing code of that paper in order to create different structures, in our case trees, with a single NCA. This NCA should ideally be able to generate a variety of shapes, sizes, and types of voxelized trees and still be controllable. We perform a multitude of experiments to determine what kind of model or training process works best for adapting a single-sample NCA into a randomized multi-sample NCA. We conducted the following experiments:

Experiment - Pools: During training (Sudhakaran et al., 2021) store the final states of the sequences as starting points for the next iteration. In our multi-sampled case, we test whether a single pool for all samples suffices as having separate ones inflates memory usage drastically.

Experiment - Id: The simplest embedding scheme provides the model with an identifier of the current training sample. The model may be able to extract enough information to diversity and not regress to a general solution.

Experiment - Embedding: Another embedding scheme provides the model directly with multi-dimensional embeddings per sample. They can be manually chosen or randomly sampled before training. This is still a deterministic setting as these embeddings stay constant during the entire training process.

Experiment - Noise: On the other hand, it has been shown that noisy input is beneficial especially when the outcome should be noisy as well. Therefore, we test using random embedding dimensions sampled from a standard normal distribution for each training step. The original models intentionally overfitted on the structures but in our case, this limits the possible variety in which noisy inputs are correct.

Experiment - Variational: Finally, we allow the model to learn its own embeddings. An auto-decoder is used to avoid additional weights for an encoder as that would not be suitable for our relatively small dataset. Both standard embeddings from an auto-decoder and variational embeddings from a variational auto-decoder are being explored. In the latter case, the training error gets backpropagated and adjusts the mean and variance per sample per embedding dimension.

During implementation, we discovered that the version and libraries used by (Sudhakaran et al., 2021) were deprecated and are not compatible anymore. So we were forced to decouple the project from Minecraft and transform it into a pure 3D voxel training and generation pipeline. Only the '.nbt' files remain for loading the samples and the visualizations were created with Open3D.

(Hendrycks & Gimpel, 2023) shows an empirical evaluation of the GELU nonlinearity against the ReLU and ELU activations and finds performance improvements across all considered tasks. Hence, we replaced RELUs in the base code with GELUs. Additionally, we added gradient clipping to our training procedure as the training was very unstable and we wanted to avoid a sudden collapse through a bad step.

The model architecture of NCAs is really small by design. The surroundings are perceived with a single convolution with a window size of 3. Further processing is done with a small number of size 1 convolutions (up to 4) which are meant to represent linear layers with weight sharing. The loss formula used while training was the same as that used in (Sudhakaran et al., 2021). It is as follows:

$$Loss = \frac{1}{N}\sum_{i=1}^{N}(\sum_{c=1}^{M}(CE(\hat{y}_{i,c}, c)) + IOU(\hat{y}_i, y_i))$$

CE refers to cross entropy loss and IOU is the Intersect Over Union (IOU) cost, which measures the absolute difference between the non-"air" blocks in the target and the generated entities. Each cell is associated with an "alive" channel and an alpha value as markers to distinguish it from others - "alive" would mean the cell or its neighbours have $alpha >= 0.1$ otherwise "dead". "Dead" cells are nullified by setting the "air" block channel to 1 and everything else to 0. For our variational decoders, we added the KL divergence to a multivariate normal distribution with mean 0 and variance 1. The weight of this additional loss is a hyper-parameter for our model, but we mainly used 0.1 for our experiments to not have the variational loss dominate everything.

3.1. Dataset

For our samples we select existing custom-built trees from (Planet Minecraft). This dataset consists of 370 custom trees that can be downloaded and used by the players to polish their builds. The first set consists of 8 trees with the same type of wood and different fir shapes. The biggest tree has dimensions (14, 29, 13).

The second set consists of 20 trees with multiple types. Some trees are notably bigger with dimensions (27, 25, 22).

Overall training can be done locally on outdated hardware. The only issue is the required memory on GPUs as the 3D nature of the task we are always memory bound. Especially with the second dataset 8GB memory is exceeded on the GPU and we had to cut back on the tree sizes.

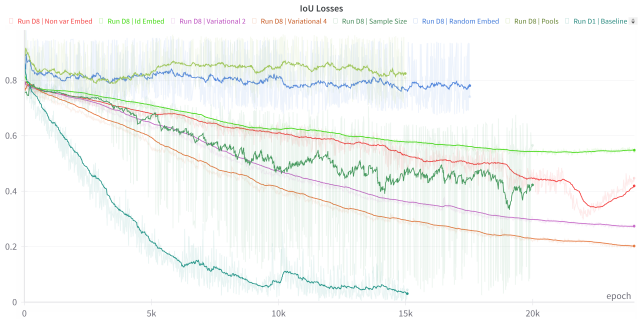

Figure 3. Comparison of IoU loss of methods trained on the small (8 trees) dataset.

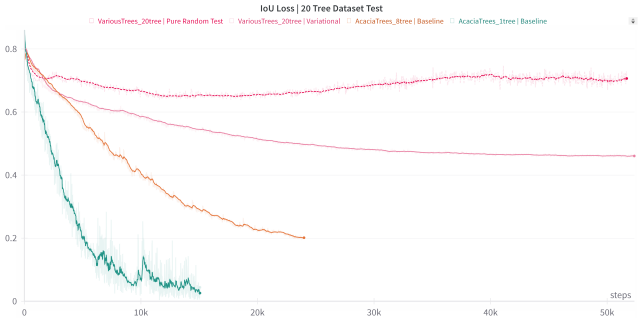

Figure 4. Comparison of IoU loss of methods trained on the large (20 trees) dataset.

4. Evaluation

A unique problem that we encountered in our project was the lack of quantitative evaluation metrics. There are no suitable metrics to measure or compare the quality of trees generated by our models. Something like the distance of a generated tree to the closest tree in the dataset would be an option, but this would not capture the variance we want to create. We decided to focus our quantitative analysis on the most useful component of the loss used for the training of the IOU. Even that is not very representative as the models trained on the second dataset have a high loss while still generating some nice-looking trees like Figure 1. It is also unclear what the best endpoint of the IoU curve actually is because at some point overfitting starts to destroy variability. Therefore, most evaluation must be done by manually comparing generated trees. This is not exhaustive and some randomness is involved, but should still allow for some comparison between the methods as some were not able to generate anything useful. The example evolution sequence of the output can be seen in Figure 2.

The loss curves for each model were monitored with the AI developer platform Weights & Biases for ease of comparison.

embedding: 1011, size: 11, 22, 11

embedding: 0000, size: 11, 22, 11

embedding: 1011, size: 18, 18, 30

Figure 5. Comparing different embeddings to different window sizes

5. Results

One challenge with neural cellular automata (NCAs) is their inherent instability in training, marked by a steady increase of the loss during certain runs without recovery, the dotted curve in Figure 4 for example. Notably, this issue appears less prominent in methods incorporating learned embeddings, where the noise is more controlled.

It can be observed in Figure 6 that the class loss is easier in general but decreases faster for models that use the variational decoders and auto-decoders. Similarly, the IoU loss shown in Figure 3 is the best when using variational auto-decoders (orange curve for 4 and pink curve for 2 embedding dimensions). It also converges for the model where we give the tree ID in the input channels (red curve).

For the second dataset, the IoU curves are shown in Figure 4. While they don't reach the same baseline as the other dataset it still manages to generate trees like shown in Figure 1 but the variety is severely limited.

Upon inspection of the learned embeddings in variational models, it appeared that the means are very close to 0, and the variances close to 1 which means that the model has no good way of differentiating between the different trees and no disentanglement is learned. This suggests that the learned embeddings are not much better than just sampling random noise, however, this is refuted by the purely random embedding experiment shown in Figure 4 (red dotted curve). Despite this obscurity, variational encodings led to the best model both in terms of loss and qualitative analysis of the generated trees.

Upon further investigation of impact of the embeddings was the discovery made, that the model learns different shapes based on the provided 3D window size it operates on. Since the different trees have different dimensions the cells can presumably communicate the distance to an edge to other cells' hidden states. The impact of altering the field size for growth surpasses that of changing embeddings as shown in Figure 5. Different embeddings only change minor details in the final shapes and not the shape as was the goal.

6. Conclusions

In this project, our aim was to extend the versatility of a single NCA to work for various 3D shapes. We tested our hypothesis by applying NCAs to generate voxelized trees from the gaming environment Minecraft. While our results show qualitative promise, they require further refinement. We believe that further research in this area and the use of certain advanced techniques could lead to a state-of-the-art model that can be extensively used in the gaming industry. One challenge we faced was the lack of suitable metrics to evaluate the generated trees, as our goal is diversity rather than exact replication. This opens up avenues for further exploration within the same framework.

Further research with more complex models is essential as our simple model does not have sufficient capability to generalize to very different shapes. In addition, bigger, more diverse data sets should then be used to create even more variety. But most importantly the remaining entanglement of the embedding space and reliance on window size need to be addressed and controlled. Finally, one could try to make the automata learn the life cycle of a tree, though the required data does not seem to exist at the moment which would allow for in-game seasonal animations and more. In essence, this exploration into the versatility of Neural Cellular Automata offers a compelling glimpse into the potential of procedural content generation for diverse 3D shapes in gaming, laying the groundwork for exciting future advancements.

References

Chabra, R., Lenssen, J. E., Ilg, E., Schmidt, T., Straub, J., Lovegrove, S., and Newcombe, R. Deep local shapes: Learning local sdf priors for detailed 3d reconstruction. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX 16*, pp. 608–625. Springer, 2020.

Gilpin, W. Cellular automata as convolutional neural networks. *Phys. Rev. E*, 100:032402, Sep 2019. doi: 10.1103/PhysRevE.100.032402. URL https://link.aps.org/doi/10.1103/PhysRevE.100.032402.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus), 2023.

Kench, S. and Cooper, S. J. Generating 3d structures from a 2d slice with gan-based dimensionality expansion, 2021.

Li, X. and Yeh, A. Neural-network-based cellular automata for simulating multiple land use changes using gis. *International Journal of Geographical Information Science*, 16:323–343, 06 2002. doi: 10.1080/13658810210137004.

Liu, F. and Liu, X. Voxel-based 3d detection and reconstruction of multiple objects from a single image. *Advances in Neural Information Processing Systems*, 34:2413–2426, 2021.

Mordvintsev, A., Randazzo, E., Niklasson, E., and Levin, M. Growing neural cellular automata. *Distill*, 2020. URL https://distill.pub/2020/growing-ca/.

Neumann, J. V. and Burks, A. W. *Theory of Self-Reproducing Automata*. University of Illinois Press, USA, 1966.

Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. Deepsdf: Learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Planet Minecraft, 2019. URL https://www.planetminecraft.com/project/tree-bundle-370-custom-trees-download/. Prebuild Minecraft Tree set.

Ruiz, A. H., Vilalta, A., and Moreno-Noguer, F. Neural cellular automata manifold, 2021.

Sudhakaran, S., Grbic, D., Li, S., Katona, A., Najarro, E., Glanois, C., and Risi, S. Growing 3d artefacts and functional machines with neural cellular automata, 2021.

Wu, J., Zhang, C., Xue, T., Freeman, W. T., and Tenenbaum, J. B. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems*, pp. 82–90, 2016.

Xu, Q., Wang, W., Ceylan, D., Mech, R., and Neumann, U. Disn: Deep implicit surface network for high-quality single-view 3d reconstruction. *Advances in neural information processing systems*, 32, 2019.

Zadeh, A., Lim, Y.-C., Liang, P. P., and Morency, L.-P. Variational auto-decoder: A method for neural generative modeling from incomplete data, 2021.

Zhang, D., Choi, C., Kim, J., and Kim, Y. M. Learning to generate 3d shapes with generative cellular automata, 2021.

Zheng, Z., Yu, T., Dai, Q., and Liu, Y. Deep implicit templates for 3d shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1429–1439, 2021.

Zhu, J., Xie, J., and Fang, Y. Learning adversarial 3d model generation with 2d image enhancer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

A. Appendix

Here are some additional Plots, for which we did not have enough space given the page requirements.

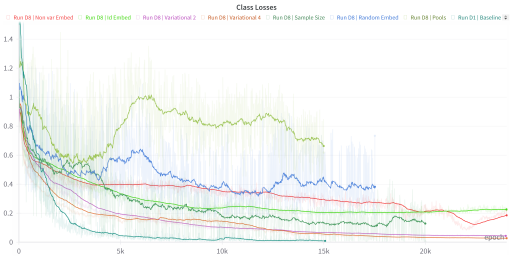

Figure 6. Class Losses (Cross-Entropy) for different models on the small (8 trees) dataset.

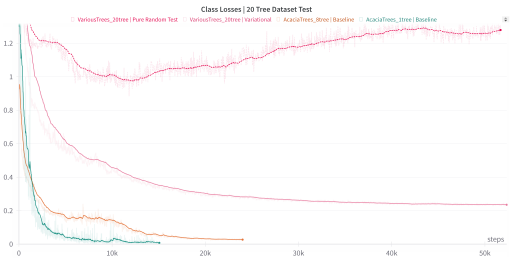

Figure 7. Class Losses (Cross-Entropy) for different models on the large (20 trees) dataset.