3D Growing Neural Cellular Automaton

June 2, 2022

Fabrizio Rossi   Matteo Orsini

Abstract

The work of Mordvintsev et al. (2020) proposes a method to model the morphogenesis process using a Neural Cellular Automaton in order to reconstruct 2D images. In our project, we extended this concept into the 3D domain using voxels. In order to do so, we created our dataset composed of different 3D shapes with various sizes and trained a model able to reconstruct the 3D target shapes with good results.

GitHub Repo: https://github.com/SkyLionx/DL2021

1. Introduction

Morphogenesis is a biological process that causes cells, tissues, and organs to develop and persist their shape. Starting from a simple initial form, cells grow to construct a more complex one. In order to do this, cells define new connections with their neighbors and, communicating, they decide what to build and when to stop growing. This very complex process is derived from simple rules which are encoded in the genome of each cell.
An application of this process is regeneration, which can be seen in some creatures, such as salamanders, that can fully regenerate vital organs, limbs, eyes, or even parts of the brain! Thus, studying this process can be useful in the computer science field in order to construct machines or agents that are able to regenerate and repair themselves.

Our project is inspired by Mordvintsev et al. (2020), who propose a method to address this task in the 2D domain using images. In fact, the method proposed, starting from a simple image with only one pixel, called seed, is able to regenerate the full target image provided during training. In order to model this biological process, they used a **Cellular Automaton** (CA) which was trained with the goal to learn an **update rule**. The latter is applied iteratively to each cell which, based on the state information of the immediate neighborhood, will update the previous cell state, eventually reconstructing the target shape. Thanks to the fact that this last operation can be modeled using the convolutional operator and the update rule can be formulated in a differentiable way, we implemented a deep model able to perform morphogenesis on the 3-dimensional Euclidean space using voxels.

Email: Fabrizio Rossi <rossi.1815023@studenti.uniroma1.it>, Matteo Orsini <orsini.1795119@studenti.uniroma1.it>.

Deep Learning and Applied AI 2021, Sapienza University of Rome, 2nd semester a.y. 2020/2021.

2. Related Work

The literature is full of works that try to model biological processes in order to understand nature and take inspiration from its solutions. Cellular Automata were originally proposed by Von Neumann (1966) but they became popular thanks to Conway's Game of Life (Gardner, 1970), which shows that, despite using very simple rules, a Cellular Automaton is able to model surprisingly complex behaviors.

In more recent years, CAs were enhanced by the power of neural networks, which are used in the already mentioned work proposed by Mordvintsev et al. (2020). A similar work to the latter was proposed by Miller (2004) but addressing the same problem using evolutionary algorithms instead of CAs.
Finally, the very recent work of Sudhakaran et al. (2021) is the most similar to ours. In fact, they also address the 3D domain starting from the method proposed by Mordvintsev et al. (2020). In their work they use the Minecraft environment to generate static structures and machines, which are also able to regenerate themselves.

3. Method

In this section, we will describe the method we used to implement the CA for the 3D domain, underlining the differences with the original work.

3.1. Cell state

A single cell is represented by a vector of 16 real values, which encode its state. The first four values of the cell state represent the RGBA color of the cell, where the alpha represents also the *"liveness"* of the cell. The rest of the state doesn't have a predefined meaning and it is going to be learned by the model.
Cells having an alpha value higher than 0.1 and their neigh-

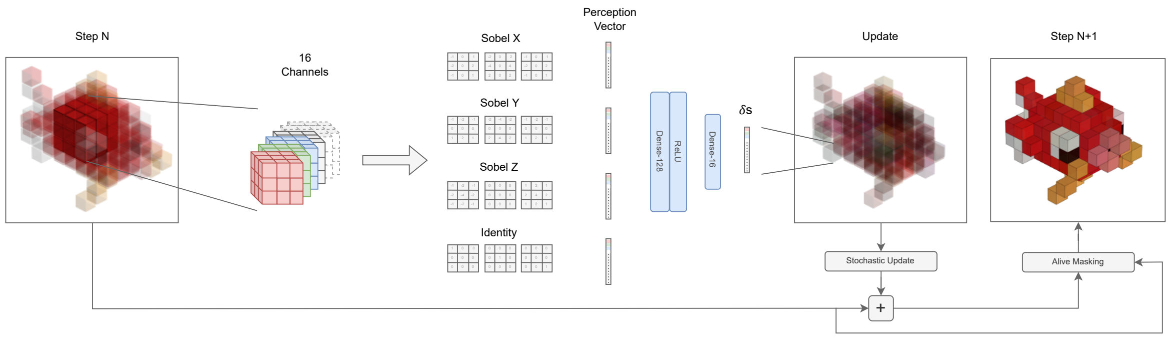

Figure 1. Representation of a single iteration of the CA

bors are considered *living* cells, instead cells with a lower value are considered *dead* cells and in each time step their state is explicitly set to 0.
Moreover, in the original work a cell corresponds to a single pixel, instead in our case it corresponds to a voxel in the 3D space.

3.2. Perceive

The update rule of the CA is composed by several phases. The first phase is the perception phase which has the goal to let each cell perceive its surrounding environment. As in the original work, to model this step we used the Sobel operator which is convolved to each state channel.
In our case, as shown in Figure 1, we used three 3D kernels, one for each direction (x, y and z). Then, for each cell, as the original paper, we concatenated all the results of the convolutions with the identity of the cell state, creating the so-called *perception vector* of size $16 * 3 + 16 = 64$.

3.3. Update

The following step has the goal to learn the actual update rule to apply to each perception vector. In order to implement it, we used the popular building blocks of Deep Learning. We kept the same small network of the original work, shown in Figure 1, which uses 1×1 convolutions and the ReLU activation function to introduce non-linearity in the model, without using an activation function on the last layer. The output of the network is then summed to the initial cell state using a stochastic procedure to mask out some of the updates. Finally, to avoid the execution of the update rule on dead cells, a living mask is computed and it is used to set the state of those cells to 0.

3.4. Training procedure

The training procedure of the CA starts with a so-called *seed* as input, which is simply an empty grid with the state of the central cell set to all 1s except the RGB values which are set to 0. For each epoch, the update rule is applied to the seed for a random number of iterations, chosen from a specified interval, in order to grow the seed into the target. At the end of each epoch the L2 loss between the RGBA values of the target and the processed grid is computed. Since the steps are all differentiable, the loss is then back-propagated through time, for all the iterations.

3.5. Persist

The authors of the original work noticed that applying the update rule for a number of iterations higher than the interval used in training leads to very strange behaviors: for example some of the patterns could ”explode” filling the grid or collapse on themselves. Thus, as they proposed, we changed the training procedure introducing a **pool** which is initially filled with all seeds.
So, in each epoch, some exemplars are sampled from the pool and fed as input to the model. Then, once processed by the model, they are substituted to the original ones. This enables the model to learn not only how to grow from the initial seed to the target, but also to reach it from an intermediate state. Moreover, to prevent that the model forgets how to update the starting seed to reach the target, in each epoch the exemplar with the highest loss in the pool sample is replaced with the initial seed.

3.6. Damage

In addition, in the original paper, to improve the regenerative capabilities and robustness of the model, during training some of the exemplars in the pool sample were ”damaged” removing a part of their shape. In practice, the 3 exemplars with lowest loss were damaged using random circles in the 2D case, and random boxes in the 3D domain.

During our experiments, we noticed that simply generating 3D random boxes often resulted in shapes with no voxels in common with the target shape, producing undamaged models. So, we decided to perform a number of tries in order to produce a valid damage. A damage is considered valid if at least two corners, or a corner and the center of

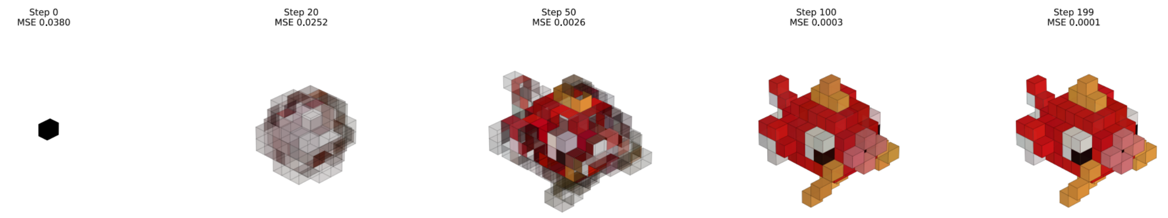

Figure 2. Iterations of our model trained with Magikarp. The output of the model at certain steps is represented with the corresponding MSE difference between the current output and the target shape.

the damage, hit a voxel of the target shape. In case no valid damage is found, a random damage is used.

3.7. Rotate

Finally, the original paper shows also how rotating the *perceptive field* of the cells, we are able to grow a rotated version of the target shape, specifying an arbitrary rotation, without retraining the model. This is achieved rotating the Sobel kernels using a rotation matrix, which in our case is defined by three angles (pitch, roll and yaw).

4. Results

In order to test our model we created an ad-hoc dataset which is composed by several 3D models with different sizes created by hand using an online voxel editor (Voxelator). The models are very different from each other in terms of colors and shapes: they range from a simple box, used for testing purposes, to a reproduction of the popular Magikarp from Pokémon and some Minecraft animals.
In order to visualize the models and animate the growth of the CA, we tried to use Plotly but it could not handle the amount of voxels we had in our models. So, we created a small visualizer using JavaScript and the framework Three.js which was integrated inside the Colab environment thanks to IPython. Moreover, we also used the 3D functionalities offered by Matplotlib for the creation of the static images and videos.

Our model, as the original one, has several hyperparameters. The number of **iterations** for each epoch is uniformly chosen in the range $[64, 96]$. The number of **convolutional filters** used in the model was set to 128, the **pool size** to 1024 and, finally, the **batch size** to 8. Moreover, after a number of experiments we found that an acceptable number of **epochs** to learn a complex shape is 3000 with a **learning rate** of 8e-4, different from the one used in the original method.
In Figure 2 is shown the progression of our model on the Magikarp. As we can see the model starts at step 0 with the empty seed, around step 100 has almost perfectly reconstructed the target shape and then, looking at the next steps, we can see that the model persists the state reached.

Additional materials, in the form of videos, were attached to the report in order to show how our model performs with all the shapes we created. For each shape a video is generated showing the persist, damage and rotate cases. In some cases the output of the model seems correct but the MSE value computed is higher than expected: this is because the model reconstructing the target may output a shifted version of it, causing this behavior which can also be observed in the original 2D implementation.

5. Discussion and Conclusions

The dataset we created allowed us to inspect the performance difference between a smaller shape and a bigger one, which wasn't investigated in the original paper, since they used the same width and height for each image. The model showed very good results reconstructing both simple and complex shapes, even if in our opinion for some of them is possible to achieve more robust results increasing the number of epochs. Unfortunately, we weren't able to train the model for a number of epochs greater than 3000 since it would take more than 6 hours for the bigger shapes. Moreover, in the majority of cases the model is also able to reconstruct both the original target shape when given as input a damaged shape (i.e. a 3D model with some missing parts), and a rotated version of the target shape based on the angles given as input.

As interesting applications of this work, we thought about using this model as a compression method, since the model encodes the target shape and it is able to reconstruct it starting from a predefined initial seed. Thus, by simply sending the few weights of the model we can reconstruct the target shape at the destination without the original one.

Instead, as possible future studies, we thought about training a single model which is able to encode not only one target shape but several ones, learning the invariances between them. Another interesting study could also be based on the interaction between different organisms, i.e. agents using different update rules, in order to simulate a more complex environment.

References

Gardner, M. Mathematical games. *Scientific American*, 223 (4):120–123, 1970. ISSN 00368733, 19467087. URL http://www.jstor.org/stable/24927642.

Miller, J. F. Evolving a self-repairing, self-regulating, french flag organism. In Deb, K. (ed.), *Genetic and Evolutionary Computation – GECCO 2004*, pp. 129–139, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-24854-5.

Mordvintsev, A., Randazzo, E., Niklasson, E., and Levin, M. Growing neural cellular automata. *Distill*, 2020. doi: 10.23915/distill.00023. https://distill.pub/2020/growing-ca.

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. *CoRR*, abs/2103.08737, 2021. URL https://arxiv.org/abs/2103.08737.

Von Neumann, J. *Theory of Self-Reproducing Automata*. University of Illionois Press, Champain, IL, 1966.