

# Differentiable Programming of Reaction-Diffusion Patterns

Alexander Mordvintsev<sup>1</sup>, Ettore Randazzo<sup>1</sup> and Eyvind Niklasson<sup>1</sup>

<sup>1</sup>Google  
moralex@google.com

## Abstract

Reaction-Diffusion (RD) systems provide a computational framework that governs many pattern formation processes in nature. Current RD system design practices boil down to trial-and-error parameter search. We propose a differentiable optimization method for learning the RD system parameters to perform example-based texture synthesis on a 2D plane. We do this by representing the RD system as a variant of Neural Cellular Automata and using task-specific differentiable loss functions. RD systems generated by our method exhibit robust, non-trivial “life-like” behavior.

## Introduction

Multicellular organisms build and maintain their body structures through the distributed process of local interactions among tiny cells working toward a common global goal. This approach, often called *self-organisation*, is drastically different from the way most human technologies work. We are just beginning to scratch the surface of integrating some of nature’s “best practices” into technology.

In 1952, Alan Turing wrote his seminal work, “The Chemical Basis of Morphogenesis” (Turing, 1952), in which he proposed that pattern formation in living organisms can be controlled by systems of chemical substances called *morphogens*. Simultaneous processes of chemical Reactions-Diffusion (RD) of these substances provide the sufficient biocomputation platform to execute distributed algorithms of pattern formation (Kondo and Miura, 2010; Landge et al., 2019).

Even a system of just two interacting substances (e.g. Gray-Scott) can produce a great diversity of patterns and interesting behaviours (Pearson, 1993). As is often the case with complex emergent behaviours of simple systems, it is very difficult find model parameters that produce a particular, predefined behaviour. Most of the time researchers use hand-tuned reaction rules, random and grid search over the parameter space for the combinations with the desired properties. Procedural texture synthesis is one of the best known applications for this type of parameter tuning (Witkin and Kass, 1991; Turk, 1991). In this paper, we propose to use

differentiable optimization as an alternative to such a trial-and-error process.

The task of determining the sets of parameters that lead to desired behaviors becomes even more important as we enter the realm of artificial life and synthetic biology. The work of Scalise and Schulman (2014) is a remarkable example of an attempt to design a flexible modular framework for RD-based spatial programming. The authors demonstrate (at least in simulation) a way to combine a set of basic computational primitives, implemented as RD systems of DNA strands, into multistage programs that generate non-trivial spatial structures. We argue that these programs cannot yet be called “self-organizing systems” due to two important limitations. First, they rely on a predefined initial state that defines the global coordinate system on which the program operates through chemical gradients or precise locations of chemical “seeds”. Second, program execution is a feed-forward process that does not imply homeostatic feedback loops that make the patterns robust to external perturbations or a changed initial state.

Another very related line of research in *artificial life* is Lenia (Chan, 2020), which aims to find rules and configurations for space-time-state-continuous cellular automata that demonstrate life-like homeostatic and replicating behaviours.

Figures in video form and a reference implementation is available here<sup>1</sup>.

## Differentiable Reaction-Diffusion Model

We study a computational model that can be defined by the following system of PDEs:

$$\frac{\partial x_i}{\partial t} = c_i \nabla^2 x_i + f_\theta(x_0, \dots, x_{n-1})$$

$x_0, \dots, x_{n-1}$  are scalar fields representing the “concentrations”  $n$  “chemicals” on a 2D plane.  $c_i$  are per-“chemical” diffusion coefficients, and  $f_\theta: \mathbb{R}^n \rightarrow \mathbb{R}^n$  is a function defining the rate of change of “chemical concentrations” due to

<sup>1</sup>[https://selfforglive.github.io/alife\\_rd\\_textures/](https://selfforglive.github.io/alife_rd_textures/)

local “reactions”. Chemical terms are used here in quotes because, in the current version of our model, the function  $f_\theta$  need not obey any actual physical laws, such as the law of conservation of mass or the law of mass action. “Concentrations” can also become negative.

## Reaction-Diffusion as a Neural CA

The objective of this paper is to train a model that can be represented by a space-time continuous PDE system. Yet, we have to discretize the model to run it on a digital computer. The discretized model can be treated as a case of Neural Cellular Automata (NCA) model (Mordvintsev et al., 2020; Randazzo et al., 2020).

Our model and the training procedure are heavily inspired by the texture-synthesizing Neural CA described by Niklasson et al. (2021). Similarly, we discretize space into cells and time into steps, use explicit Euler integration of the system dynamics and backpropagation-through-time to optimize the model parameters. Our model differs from the previous texture-synthesizing NCA:

- CA iteration does not have the *perception* phase. The “reaction” part of cell update (modelled by  $f_\theta$ ) depends on the current state of the cell only.
- There is an isotropic diffusion process that runs in parallel with “reaction” and is modelled by channelwise convolution with a Laplacian kernel. This is the only method of communication between cells. Thus, the Neural RD model is completely isotropic: in addition to translation, the system behaviour is invariant to rotational and mirroring transformations.
- We do not use stochastic updates, all cells are updated at every step.

Thus, the system described here can be seen as a stripped version of the Neural CA model. These restrictions are motivated by a number of practical and philosophical arguments described below.

**Model simplicity and prospects of physical implementation** The discussion section of “Growing NCA” article by Mordvintsev et al. (2020) contains speculations about potential of physical implementation of the Neural CA as a grid of tiny independent computers (microcontrollers). Neural CA consists of a grid of discrete cells that are sophisticated enough to persist and update individual state vectors, and communicate within the neighborhood in a way that differentiates between neighbours, adjacent to different sides of the cell. This implies the existence of global alignment between cells, so that they agree where up and left are, and separate communication channels to send the information in different directions. In contrast, diffusion based communication doesn’t require any dedicated structures, and

“just happens” in nature due to the Brownian motion of molecules.

Furthermore, states of Neural CA cells are only modified by their learned update rules and are not affected by any environmental processes. Cells have clear separation between what’s inside and outside of them, and can control which signals should be let through. However, many natural phenomena of self-organisation can be effectively described as a PDE system on a continuous domain. Individual elements that constitute the domain are considered negligibly small, and the notion of their individual updates is meaningless.

In the experiments section we cover some practical advantages of the proposed RD model with respect to Neural CA. In particular we demonstrate the generalization into arbitrary mesh surfaces and even volumes.

**Reaction-Diffusion CA update rule** The discrete update rule can be written as follows:

$$\begin{aligned} x_i^{t+1} &= x_i^t + c_i K_{lap} * x_i^t d + f_\theta(x_0^t, \dots, x_{n-1}^t) r \\ d &= \Delta_t / \Delta_h^2, \quad r = \Delta_t \end{aligned} \quad (1)$$

$K_{lap}$  is a 3x3 Laplacian convolution kernel,  $c_i$  and  $\theta$  are parameters that control the CA behaviour, and the coefficients  $r$  and  $d$  control the rates of reaction and diffusion, encapsulating temporal and spatial discretization step sizes  $\Delta_t$  and  $\Delta_h$ . By varying these coefficients we can validate if the learned discrete CA rule approximates the continuous PDE and does not over-fit to the particular discretization. During training we use  $r = d = 1.0$ .

The function  $f_\theta(\mathbf{x}) = act(\mathbf{x}W_0 + b_0)W_1$  is a small two layer neural network with parameters  $\theta : (W_0 \in \mathbb{R}^{n \times h}, W_1 \in \mathbb{R}^{h \times n}, b_0 \in \mathbb{R}^h)$  and a non-linear element-wise activation function  $act$  (see the experiments section). In our experiments we use  $n = 32, h = 128$ , so the system models the dynamics of 32 “chemicals”, and the total number of network parameters equals to 8320. Per-“chemical” diffusion coefficients  $c_i$  can be learned (in this case we set  $c_i = sigmoid(\hat{c}_i)$  to make sure that diffusion rate stays in 0..1 range), or fixed to specific values.

## Texture synthesis

Reaction-Diffusion models are a well-known tool for texture synthesis. Typically, manual parameter tuning has been used to design RD systems that generate desired visual patterns (Witkin and Kass, 1991; Turk, 1991). We propose an example-based training procedure to learn a RD rule for a given example texture image. This procedure closely follows the work “Self-Organising Textures” (SOT) (Niklasson et al., 2021), with a few important modifications. Our goal is to learn a RD update function whose continuous application from a *starting state* would produce a pattern similar to the provided texture sample. The procedure is summarised in algorithm 1.

---

**Algorithm 1:** Texture RD system training

---

```

 $pool \leftarrow N_{pool}$  randomly sampled seed states ;
for  $step \leftarrow 1$  to  $N_{train}$  do
     $batch \leftarrow$  sample  $N_{batch}$  random pool indices ;
     $x \leftarrow pool[batch]$ ;
    if  $step \bmod R_{seed} = 0$  then
        |  $x[0] \leftarrow$  random seed state
    end
    for  $i \leftarrow 1$  to  $Uniform(I_{min}, I_{max})$  do
        |  $x \leftarrow RD_\theta(x)$ ;
    end
     $loss \leftarrow calcLoss(x)$  ;
    update  $\theta$  using the gradient of  $loss$  ;
     $pool[batch] \leftarrow x$ ;
end

```

|             |       |           |    |             |      |
|-------------|-------|-----------|----|-------------|------|
| $N_{train}$ | 20000 | $I_{min}$ | 32 | $N_{pool}$  | 1024 |
| $R_{seed}$  | 32    | $I_{max}$ | 96 | $N_{batch}$ | 4    |

---

**Seed states** SOT uses seed states filled with zero values. Stochastic cell updates provide the sufficient variance to break the symmetry between cells. We use synchronous explicit Euler integration scheme that updates all cells simultaneously, so the non-uniformity of the seed states is required to break the symmetry. We initialize the grid with a number of sparsely scattered Gaussian blobs (fig.1).

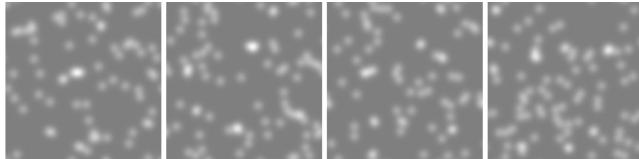


Figure 1: Random *seed states*.

Periodic injection of seed states into training batches is crucial to prevent the model from forgetting how to develop the pattern from the seed, rather than improving already existing one. We observed that it is sufficient to inject the seed much less often than in SOT. We use  $R_{seed} = 32$  in this experiment.

**Rotation-invariant texture loss** Similar to SOT, we interpret the first 3 “chemicals” as RGB colors and use them to calculate texture loss. Our texture loss is also based on matching Gram matrices of pre-trained VGG network activations (Gatys et al., 2015), but has an important modification to account for the rotational invariance of isotropic RD. Consider the texture `lined_0118` from the figure 2. The target pattern is anisotropic, but RD, unlike NCA, has no directional information and cannot produce a texture that exactly matches the target orientation. We construct the rotation-invariant texture loss by uniformly sampling  $N_{rot}$

target images rotated by angles  $0^\circ \dots 360^\circ$  and computing the corresponding texture descriptions. This computation only occurs in the initialization phase and does not slow down the training. At each loss evaluation, the texture descriptors from RD are matched with all target orientations and the best match is taken for each sample in the batch.

Per-“chemical” diffusion coefficients in Eq. (1) are set to  $c_{0..7} = 1/8$ ,  $c_{8..15} = 1/4$ ,  $c_{16..23} = 1/2$ ,  $c_{24..31} = 1$ , so that the “substances” are split into 4 groups of varying diffusivity. Therefore, RGB colors correspond to slowly diffusing channels  $x_{0..2}$ . We experimented with learned diffusion coefficients, but it did not seem to bring substantial improvement, so we kept fixed values for the simplicity. In all of the texture synthesis experiments we use wrap-around (torus) grid boundary conditions.

RD network uses a variant of Swish (Ramachandran et al., 2017) elementwise activation function:  $act(x) = x\sigma(5.0 * x)$ , where  $\sigma$  is a sigmoid function.

We trained seven texture-synthesizing RD models (fig.2). Six were using 128x128 images from DTD dataset (Cimpoi et al., 2014), and the last was using a 48x48 lizard emoji image, replicated four times. Models were trained for 20000 steps using the Adam optimiser (Kingma and Ba, 2015) with learning rate 1e-3 decayed by 0.2 at steps 1000 and 10000. We also used the gradient normalization trick from SOT to improve training stability. Training a single RD model took about 1 hour on the NVIDIA P100 GPU.

## Results

In spite of the constrained computational model, trained RD systems were capable of reproducing (although imperfectly) distinctive features of the target textures. All models except `chequered_0121` seemed to be isotropic, which manifested in the random orientation of the resulting patterns, depending on the randomized seed state. `chequered_0121` always produced diamond-oriented squares, which suggests overfitting to the particular discrete grid. Below we investigate this behaviour more carefully.

Witkin and Kass (1991) proposed using anisotropic diffusion for anisotropic texture generation with RD. In our experiments, we demonstrated the capability of fully isotropic RD systems to produce locally anisotropic textures through the process of iterative alignment, that looks surprisingly “life-like”. Figure 3 shows snapshots of grid states at different stages of pattern evolution. We recommend watching supplementary videos to get a better intuition of RD system behaviours.

**Do we really learn a PDE?** We decided to validate that the discrete Neural CA that we use to simulate RD system is robust enough to be used with different grid resolution than used during training. This would confirm that the system we trained really approximates a space-time continuous

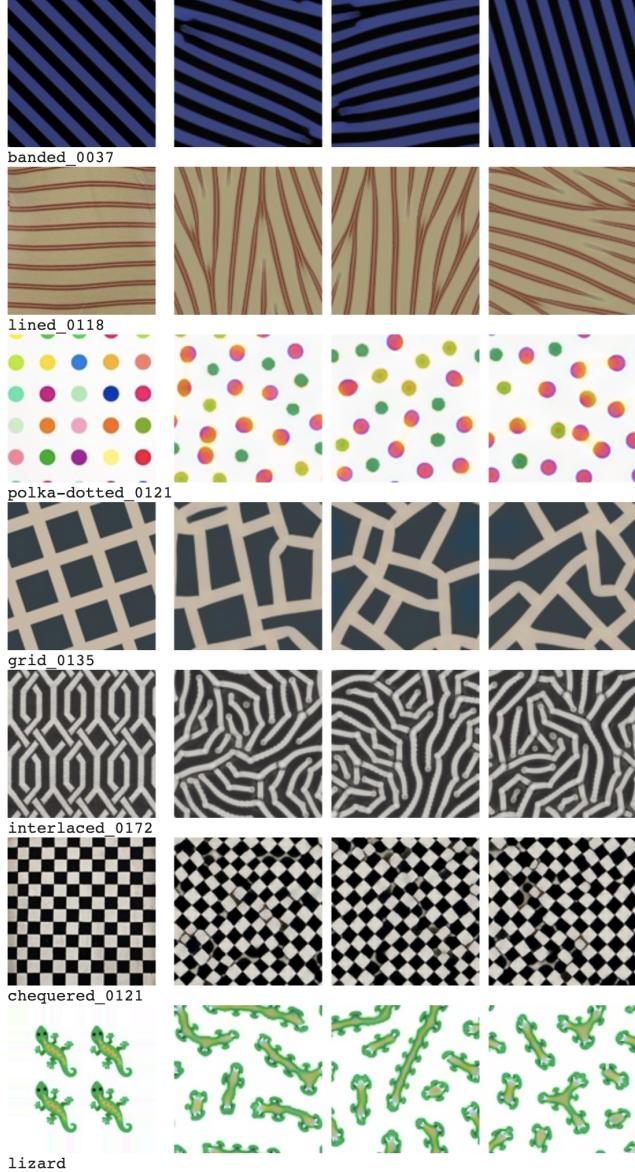


Figure 2: Left column: sample texture images; other columns: different 5000-step runs of the learned RD models. Almost all models replicate features of the target texture in rotation-invariant fashion. Only chequered\_0121 overfitted to exploit the underlying raster grid structure to always produce diamond-oriented checker squares.

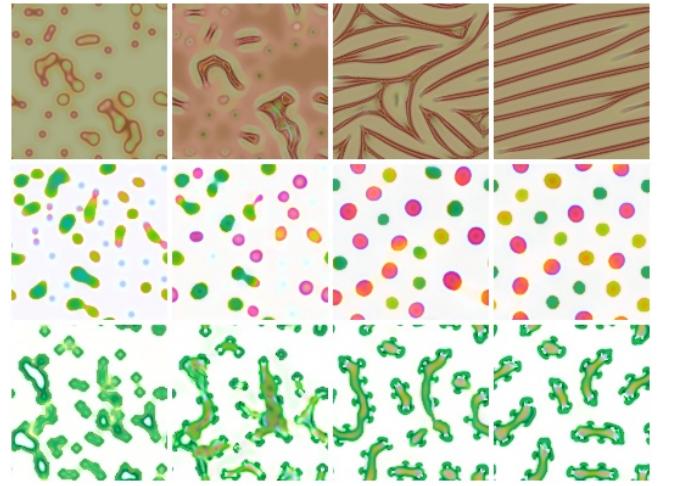


Figure 3: Stages of texture development at time steps 50, 100, 1000 and 10000. Anisotropic textures form by local symmetry breaking and refinement. Resulting patterns look consistent over long time periods, but never reach full stability. For example, dots in the middle row asynchronously change colors.



Figure 4: Running RD system on a non-uniform  $r$  grid. All models except the rightmost seem to be capable of operating on a modified grid, preserving the key pattern characteristics. chequered\_0121 overfitted to the particular grid resolution and is unable to produce right corners at larger scales. Sometimes it even develops instabilities and explodes.

PDE without overfitting to the particular grid discretization. One way to execute the RD on a finer grid is to decrease the  $\Delta_h$  coefficient in (1). This leads to the quick growth of the diffusion term, making the simulation unstable. We can mitigate this by reducing the  $\Delta_t$  as well. In practice we keep  $d = 1.0$ , but decrease  $r$ , so that  $\Delta_h = \sqrt{r}$ . Thus, setting  $r = 1/4$  corresponds to running on a twice finer grid, and should produce patterns magnified two times. Decrease of  $r$  may also be interpreted as reaction speed slowdown or diffusion acceleration.

Fig. 4 shows results of RD system evaluation on a grid having non-uniform  $r$ : in the center  $r = 1$ , slowly decreasing to  $r \approx 1/9$  at the boundary. Most of the trained models were capable of preserving their behaviour independent of the grid resolution. chequered\_0121 model was the only



Figure 5: RD texture models can easily be applied to mesh surfaces. We treat each vertex of the Stanford Bunny model as a cell, and allow the associated state vectors to diffuse along the mesh edges. This enables consistent texturing of dense meshes without constructing UV-maps or tangent coordinate frames. Please see the supplementary videos for the system dynamics and more views.

exception. The grid-overfitting hypothesis was confirmed by the fact that the model could not form right angles at the grid corners, and even developed instabilities in the fine resolution grid areas.

**Generalization beyond 2D plane** RD-system can be applied to any manifold that has a diffusion operation defined on it. This enables much more extreme out-of-training generalization scenarios, than those possible for Neural CA. For example, applying texture synthesis method by Niklasson et al. (2021) to a 3d mesh surface would require defining smooth local tangent coordinate systems for all surface cells (for example, located at mesh vertices). This is necessary to compute partial derivatives of cell states with some variant of generalized of Sobel filters. In contrast, Neural RD doesn't require tangents, and can be applied to an arbitrary mesh by state diffusion over the edges of the mesh graph (see fig. 5). Even more surprisingly, RD models that were trained to produce patterns on a 2d plane, can be applied to spaces of higher dimensionality by just replacing the diffusion operator with a 3d equivalent. Figure 6 shows examples of volumetric texture synthesis by 2d models.

## Discussion

Reaction-Diffusion is known to be an important mechanism controlling many developmental processes in nature (Kondo and Miura, 2010; Landge et al., 2019). We think that mastering RD-system engineering is an important prerequisite for

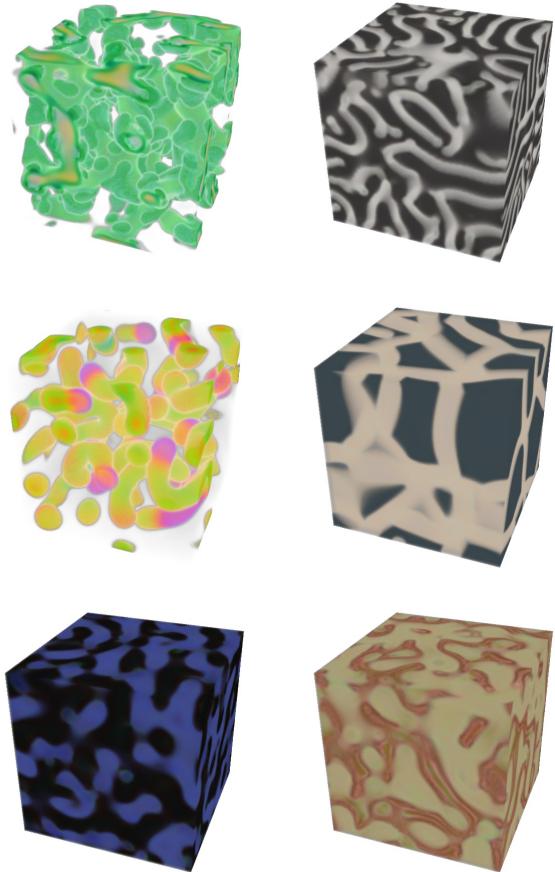


Figure 6: 2D to 3D generalization of Reaction-Diffusion models. The models that were trained on 2D plane with 2D image loss and executed on a 3D space. For some patterns, individual slices though the volume have textures similar to the target images, while for others (e.g. the last row) the similarity is less convincing. We treat white color as transparent to visualize the structure of lizards and polka-dotted patterns. Please see supplementary videos for model dynamics.

making human technology more life-like: robust, flexible, and sustainable. This work demonstrates the applicability of Differentiable Programming to the design of RD-systems. We think this is an important stepping stone to transform RD into a practical engineering tool.

To achieve this, some limitations should be addressed in future work. First, it is crucial to find such optimization problem formulations that would produce physically plausible RD systems. Second, further research in the area of differentiable objective formulations is needed to make this approach applicable to a broader range of design problems for self-organizing systems.

## References

- Chan, B. (2020). Lenia and expanded universe. *ArXiv*, abs/2005.03742.
- Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., and Vedaldi, A. (2014). Describing textures in the wild. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3606–3613.
- Gatys, L. A., Ecker, A. S., and Bethge, M. (2015). Texture synthesis using convolutional neural networks. In *NIPS*.
- Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.
- Kondo, S. and Miura, T. (2010). Reaction-diffusion model as a framework for understanding biological pattern formation. *Science*, 329:1616 – 1620.
- Landge, A. N., Jordan, B. M., Diego, X., and Müller, P. (2019). Pattern formation mechanisms of self-organizing reaction-diffusion systems. *Developmental Biology*, 460:2 – 11.
- Mordvintsev, A., Randazzo, E., Niklasson, E., and Levin, M. (2020). Growing neural cellular automata. *Distill*. <https://distill.pub/2020/growing-ca>.
- Niklasson, E., Mordvintsev, A., Randazzo, E., and Levin, M. (2021). Self-organising textures. *Distill*. <https://distill.pub/2021/textures>.
- Pearson, J. E. (1993). Complex patterns in a simple system. *Science*, 261(5118):189–192.
- Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Swish: a self-gated activation function. *arXiv: Neural and Evolutionary Computing*.
- Randazzo, E., Mordvintsev, A., Niklasson, E., Levin, M., and Greydanus, S. (2020). Self-classifying mnist digits. *Distill*. <https://distill.pub/2020/selforg/mnist>.
- Scalise, D. and Schulman, R. (2014). Designing modular reaction-diffusion programs for complex pattern formation.
- Turing, A. M. (1952). The chemical basis of morphogenesis. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 237(641):37–72.
- Turk, G. (1991). Generating textures on arbitrary surfaces using reaction-diffusion. *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*.
- Witkin, A. and Kass, M. (1991). Reaction-diffusion textures. *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*.