Asynchronicity in Neural Cellular Automata

Eyvind Niklasson, Alexander Mordvintsev and Ettore Randazzo

Google Research
eyvind@google.com

Abstract

Cellular Automata have intrigued curious minds for the better part of the last century, with significant contributions to their field from the likes of Von Neumann et al. (1966), John Conway (Gardner (1970)), and Wolfram and Gad-el Hak (2003). They can simulate and model phenomena in biology, chemistry, and physics (Chopard and Droz (1998)). Recently, Neural Cellular Automata (NCA) have demonstrated a capacity to learn complex behaviour, including constructing a target morphology (Mordvintsev et al. (2020)), classifying the shape they occupy (Randazzo et al. (2020)), or segmentation of images (Sandler et al. (2020)). As a computational model, NCA have appealing properties. They are parallelisable, fault tolerant and partially robust to operating on manifolds other than those used during training. A strong parallel exists between training NCA and system identification of a partial differential equation (PDE) satisfying certain boundary value conditions. In the original work by Mordvintsev et al. (2020), asynchronicity in cell updates is justified by a desire to have purely local communication between cells. We demonstrate that asynchronicity is not just an ideological feature of the model and is in fact necessary to learn a well-behaved PDE and to allow the model to be used in arbitrary integrators.

Introduction

Neural Cellular Automata (NCA) have been applied to a diverse set of tasks, and have shown a propensity to learn complex behaviour despite small model sizes. In a biological setting, similar tasks must be performed in the presence of noise, both in the environment and internal to the execution of the cell's functions, and without a global clock. Thus, one would expect a biological model to be robust, self-correcting and impervious to order of execution among cells.

The training regime for NCA is straightforward and the underlying models are often quite small by the standards of deep learning today (on the order of $1e3$ parameters). The authors' use of asynchronicity in the update step is not introduced as an attempt to improve convergence, but rather to ensure the underlying model operates in a completely local fashion. Synchronous updates in a system generally require a shared global clock. In a biological system or artificial distributed system, this is not always feasible or desired. Synchronicity can be achieved through clock synchronization algorithms in-silico, or signals globally diffusing over cells clusters biologically, but a preferable solution is for cells to themselves learn an optimal version of such behaviour or to operate in a fashion that is largely invariant to global time.

Mordvintsev et al. (2020) note that NCA learn a dynamical system, and Niklasson et al. (2021) make this link more clear by framing the training of NCA as both learning and numerically solving a partial differential equation. We expand on the analogy and discuss some of the subtleties associated with it. We demonstrate that asynchronicity during training not only satisfies an ideological requirement of the NCA to operate in a truly local fashion, but also makes the learned PDE more well behaved, allowing the model to be robust to alternative implementations and numerical integration schemes.

We use the model as described in Mordvintsev et al., without alive-masking and with the batch-sampling mechanism, and use the target image of a lizard emoji, hex-code *u1f98e* from the Noto-Emoji font. We train the model for 8,000 steps. An implementation of experiments is available here[1].

NCA as PDEs

Training an NCA is equivalent to simultaneously constructing a partial differential equation (PDE) and solving it with numerical methods. The NCA constructs a PDE of the form:

$$\frac{\partial \mathbf{s}}{\partial t} = f(\mathbf{s}, \nabla_{\mathbf{x}}\mathbf{s})$$

where $\mathbf{s}$ represents a k dimensional vector, whose first three components correspond to the visible RGB color channels: $\tilde{\mathbf{s}}(\mathbf{x}, \mathbf{t}) = (s^0 = R, s^1 = G, s^2 = B, s^3, ..., s^{k-1})$. The RGB channels are used to construct the loss against the target image. $\nabla_{\mathbf{x}}\mathbf{s}$ denotes a matrix of per-component gradients over $\mathbf{x}$, f is a function parameterised by a neural network. The evolution of the system starts with an initial state $\mathbf{s}_0$ and evolves in time according to f. Later work by Niklasson et al. (2021) introduce minor alterations, such as providing the network with a laplacian term, $\nabla^2_{\mathbf{x}}\mathbf{s}$, as well.

[1] https://colab.research.google.com/github/google-research/self-organising-systems/blob/master/notebooks/async.ipynb

To numerically solve PDEs, one must discretize the spatio-temporal domain, provide the discrete version of gradient operators, and specify the integration algorithm. A chosen discretization method in space is referred to as a "stencil" in the field of finite difference methods; we use Sobel filters (Sobel and Feldman (1973)) in $\vec{x}$ and $\vec{y}$ (termed K_x and K_y, respectively). The evolution of the NCA state $\mathbf{s}_t(x, y)$, where x and y are integer cell coordinates (indexing a Cartesian 2D raster grid), is now given by:

$$\mathbf{p}_t = concat(\mathbf{s}_t, K_x * \mathbf{s}_t, K_y * \mathbf{s}_t) \quad (1)$$

$$\mathbf{s}_{t+1} = \mathbf{s}_t + f(\mathbf{p}_t)\delta_{x,y,t}\lambda_{x,y,t} \quad (2)$$

where $concat$ denotes stacking along the state-vector dimension, and $f(\mathbf{p}_t)$ operates element-wise on this grid. The cell update rate is denoted by $\delta_{x,y,t}$ and the step size is denoted by $\lambda_{x,y,t}$, set to 1.0 during training. We present two cases; we consider NCA to be 'synchronous' if we choose $\delta_{x,y,t} = 1.0$; every cell updates at every time step. As in Mordvintsev et al. (2020), we consider NCA to be 'asynchronous' if each $\delta_{x,y,t}$ is a random variable $\delta_{x,y,t} \sim B(1, p)$; in other words each cell randomly samples whether or not to do an update with probability p. We choose $p = 0.5$ for our experiments. The synchronous case can be interpreted as an integration step of the PDE with an explicit Euler integration method in time, with step-size $\lambda_{x,y,t}$. The asynchronous case can be seen as a stochastic PDE.

Gradient descent using the Adam ((Kingma and Ba, 2014)) optimizer is used to iteratively modify the parameters defining the PDE until the numerical solution of the PDE matches the boundary value conditions we have chosen. The task of finding a PDE or dynamical system to match or model a desired behaviour is known as a system identification problem.

In Mordvintsev et al. (2020), these boundary value conditions consist of an initial state of a single black cell, and a final state of the image of a lizard.

Traditionally, PDEs can be analysed in a multitude of ways, such as investigating error when numerically integrated to determine the stiffness of the equation. A key thing to bear in mind is that our NCA, by construction, find a PDE that satisfies our boundary value conditions **when numerically integrated with our chosen method and step-size.** This leaves open the possibility of converging to PDEs which satisfy our conditions but only when numerically integrated with the specific integrator used during training. Ideally, we would want our NCA to be decoupled from the exact implementation of numerical integration we have chosen, and instead output a 'true' derivative of the dynamical system at that point in time.

Is our PDE well behaved?

We choose to consider our PDE well-behaved if we can integrate it with smaller-step sizes than used during training, without increasing loss or the pattern becoming unstable.

In Figure 1 we see that the synchronous training of NCA results in NCA that maintain a stable version of the target pattern, with low loss, when the step-size is the same as what was used during training. However, for any other step-size, the loss increases and for larger step-sizes the pattern destabilizes and the loss becomes very large. Figure 2 shows the corresponding visual output. The increases in loss for the synchronous NCA are subtle but visible as deterioration in the re-creation of the lizard pattern.

Using asynchronous training for the NCA results in a very stable behaviour across different step-sizes, with an increase in loss at higher step-sizes.

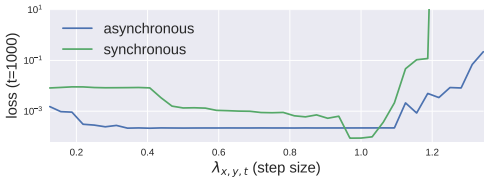

Figure 1: Loss at 1,000 iterations. The asynchronous model appears to learn a more well-behaved PDE, behaving the same when integrated with step-sizes smaller than training, and is also somewhat robust to larger step-sizes. The synchronous model performs significantly worse in both cases.

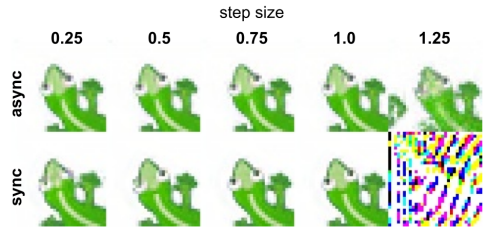

Figure 2: Zoomed representation of lizard after 800 iterations. The synchronously trained NCA degrades both for larger and for smaller step-sizes, with features such as the lizard's eyes failing to be reconstructed.

We hypothesize the reason asynchronous training improves the PDE behaviour is that at any given update, a cells' neighbours may differ by multiple time-steps. Thus, during training, the model is exposed to neighbours at a range of different time-steps, and must then be able to recover and continue the time-evolution of the model, preventing it from overfitting to a single timestep.

Asynchronously trained models representing well-behaved PDEs open the door to more optimized integrators, employing schemes such as adaptive step-sizes. This work also highlights an important caveat to approaches that incorporate numerical integration methods with learned models, such as the family of work around Neural ODEs (Chen et al. (2018)); that a learned model can easily exploit properties of the numerical integration scheme and care has to be taken to learn the true derivative of a process of interest.

References

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. (2018). Neural ordinary differential equations.

Chopard, B. and Droz, M. (1998). *Cellular Automata Modeling of Physical Systems*. Cambridge University Press.

Gardner, M. (1970). Mathematical games. *Scientific American*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

Mordvintsev, A., Randazzo, E., Niklasson, E., and Levin, M. (2020). Growing neural cellular automata. *Distill*, 5(2):e23.

Niklasson, E., Mordvintsev, A., Randazzo, E., and Levin, M. (2021). Self-organising textures. *Distill*, 6(2).

Randazzo, E., Mordvintsev, A., Niklasson, E., Levin, M., and Greydanus, S. (2020). Self-classifying mnist digits. *Distill*, 5(8).

Sandler, M., Zhmoginov, A., Luo, L., Mordvintsev, A., Randazzo, E., and Arcas, B. A. y. (2020). Image segmentation via cellular automata.

Sobel, I. and Feldman, G. (1973). A 33 isotropic gradient operator for image processing. page 271–272.

Von Neumann, J., Burks, A. W., and Goldstine, H. H. (1966). *Theory of self-reproducing automata*. University of Illinois Press.

Wolfram, S. and Gad-el Hak, M. (2003). A new kind of science. *Applied Mechanics Reviews*, 56(2):B18.