VON NEUMANN NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

In the mid-twentieth century, mathematician and polymath John von Neumann created a computational system on an array of cells as a simple model of the human brain, where each cell had one of a finite set of roles or states that he predicted would be modelled by a diffusion process. In this work, we show that such a system, when developed in a modern deep learning setting, enables the construction of an artificial neuron having specialized roles that can be learnt. We refer to this neuron as the Von Neumann neuron, and the resulting neural network from such neurons result in a self-engineered design whose architecture is only dependent on the structure and locations of its inputs and outputs on this cellular array. The mathematical framework for these Von Neumann Networks (VNNs) is also constructed and shows that they are based on the extension of neural operators and the learning of Green's functions with convolutions on a cellular topology having a diffusion signature. We also prove that these VNNs are part of a more general computational system called cellular machines that are computationally universal. Initial experiments show that VNN based multi-layered perceptrons outperform their equivalent deep learning variant on basic tasks, while being more parameter efficient and are capable of learning new types of tasks. This includes the ability to solve for and construct an extension of the Von Neumann (hardware) architecture common to all modern computers to cells and suggests new opportunities that could be explored.

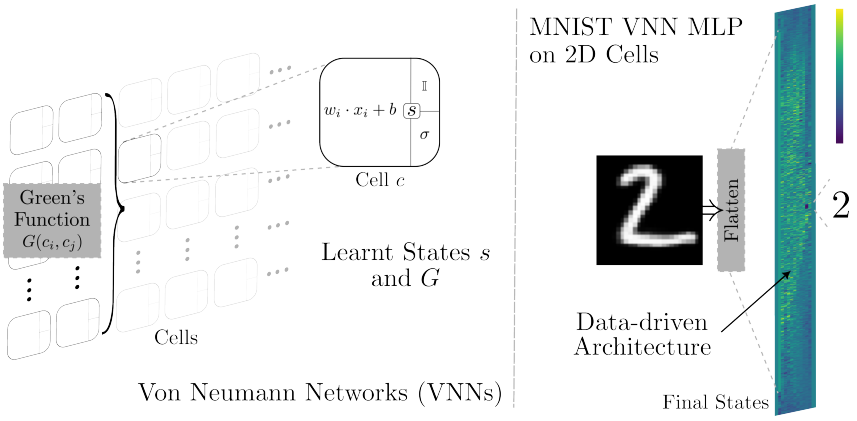


Figure 1: Inspired by the works of John von Neumann, we show that with a suitable state-based artificial neuron as a cell and neural operators implemented as convolutional Green's functions on cellular arrays, result in neural networks we call Von Neumann Networks capable of self-engineering its own connectivity and architecture entirely embedded within its array. The MNIST example above (right) shows the flattened input as a column into a VNN model results in the array of states representing cells responsible for making decisions, i.e. traditional neurons, forms a data-driven encoder-like structure (yellow points) as network propagates from left to right.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

In the mid-twentieth century, John von Neumann (or Johnny[1] as he insisted (Bhattacharya, 2021)) envisioned a computational system on an infinite array of square cells to mimic the human brain. Johnny was motivated in understanding how the human brain functions and could be modelled as a digital machine (von Neumann, 1958, pg. 43) and these simplified cellular structures were designed to approximate this model of a digital brain (von Neumann, 1966, pg. 93). His key insight in this model was that these cells could make decisions like neurons in the brain but would also maintain states depending on their purpose and then used it to construct a machine capable of reproducing itself (von Neumann, 1966, published posthumously). This theory of self-replicating machines is one of the founding works of cellular automata (CA), in which simulations evolve the state of each cell in this array via a set of simple rules based on current states and their neighbours (Sipper, 1998). He would eventually propose a continuous version of the model that would be "a system of non-linear partial differential equations (PDEs), essentially of the diffusion type" (von Neumann, 1966, pg. 95) but he would not be able to write up this continuous model before his death in 1957. In recent years, deep learning has resolved many problems associated with artificial neural networks and has been used to solve a large number of important problems (LeCun et al., 2015) including more recently in language (Vaswani et al., 2017), chemistry (Jumper et al., 2021) and computer vision (Krizhevsky et al., 2012; He et al., 2016; Liu et al., 2022; Rao et al., 2023).

In what follows, we present a novel computational model on arrays of square cells that we prove to be computationally universal and result in a new neural network framework capable of learning their own architectures and connectivity (see Figure 1). This cellular model is based on a novel artificial neuron based on Johnny's cell design that maintains its own role or state modernized for the deep learning era. The proposed Neurocell framework simulates the interaction of neurons with extensions to neural operators and learning neuron signal propagators known as Green's functions (equivalently point spread functions (PSFs)) to cellular topologies with convolutional operators in order to propagate their impulses. The framework then enables each neuron's (continuous) state be optimized through stochastic gradient descent, thereby allowing cells to learn their own (continuous CA-like) rules of evolution. The result is a discrete system that integrates a partial differential equation of diffusion type analogous to what Johnny predicted and models neural networks as structures embedded within these cells in what we will call Von Neumann Networks (VNNs). In the following sections, we review similar works to the proposed framework, its mathematical framework using Green's functions, results of experiments that show flexible networks capable of learning models of different shapes and tasks, and three different proofs of their computational univesality.

2 RELATED WORKS

Johnny's primary motivation for a cellular system was to mimic the human brain with a simple computational model via cells representing neurons that had states enabling different roles. Johnny would eventually develop a cellular model that implements a machine with the ability to replicate itself by designing each cell in his system to have 29 different states for this replication process (von Neumann, 1966, pg. 132). Codd (1968) would simplify the number of states from Johnie's 29 to just 8 including the important 'signal' state that passes on signals within the array corresponding to the computation. Conway would show that a cellular system with significant complexity and capability of universal computation could be achieved with just two states, although self-replication has not been shown yet (Gardner, 1970). Since Johnny's initial proposal of a 29 state self-replicating machine, majority of advancements have been largely focused on simplifying the number of states required, while creating various rule sets of CA that produce Turing complete systems; see a brief review of this area provided in Appendix A. After the development of traditional CA approaches such as the Game of Life (GoL) (Gardner, 1970; Wolfram, 1983; Langton, 1984, see Appendix I for a brief introduction) that have hand-crafted rules, various attempts have been made to learn CA rules. Chua and Yang (1988) introduced Cellular Neural Networks (CeNNs) that use hand-crafted templates and solutions to ordinary differential equations to create their networks prior to the deep learning era (see Chua et al. (1998) for an introduction). They would be capable of learning global structures for pattern creation as long as a suitable template for local connectivity was found. The templates usually have 19 parameters for a 3x3 neighborhood and would eventually be learnt through

[1]We follow Freeman Dyson's convention (von Neumann, 2018, see Forward)

genetic algorithms and backpropagation algorithms from recurrent networks (Chua et al., 1998). These CeNNs would later be extended to provide Turing completeness of their models (Roska and Chua, 1993). Neural operators were recently specifically introduced to learn the functions responsible for differentiating and integrating within networks (Kovachki et al., 2023). Additional work has shown that these operators can be computed efficiently using convolutions and Fourier space (Li et al., 2020) and recently showing that Green's functions themselves can be learnt (Yoo et al., 2025). Other works have attempted to improve neural networks by extending its use of dynamic information routing (Sabour et al., 2017) and learning of activation functions on network edges (Liu et al., 2024).

The recent work of Mordvintsev et al. (2020) has involved applying modern deep learning models to learn the CA rules as a multi-layered perceptron (MLP) to generate specific patterns across its cells. Since it is impossible to know *a priori* if a pattern produces complex behaviour without first simulating it in CAs like GoL, generating patterns for applications is highly desirable. In their formulation however, each cell contains an MLP that processes the neighbourhood and learns to produce a particular target pattern or image. These neural CA together with morphogensis originally proposed by Turing (1990) are very well suited for generating textures on images (Mordvintsev and Niklasson, 2021) and meshes (Pajouheshgar et al., 2024). Recently, Miotti et al. (2025) have even proposed using these neural CA to represent logic systems. Recent work on differential logic gates has made advances in learning kernels using logic gates (Petersen et al., 2022). These logic gate networks have recently been combined with neural CA to learn rules for CA including the rules GoL (Miotti et al., 2025). Learning of architectures is also not new with the advent of neural architecture search (NAS) in deep learning over the last decade especially when using reinforcement learning (Zoph and Le, 2017). In general, these works optimize for the architecture either by using a more general pre-existing architecture such as trusses (Liu et al., 2019), supernets (Lu et al., 2021) or fabrics (Saxena and Verbeek, 2016)) of inter-connected layers to select the optimal one or picking layers as needed via tree search (Wang et al., 2021) (see (Wang and Zhu, 2024) for a detailed review). Various attempts have been made to add memory and states to deep learning, but these have had largely unstable training, although they were able to learn to solve simple problems (Graves et al., 2014; 2016). A few attempts have been made to produce learnable CA including CoDI (Gers et al., 1998) that relies on spiking networks and genetic encoding to simulate its network.

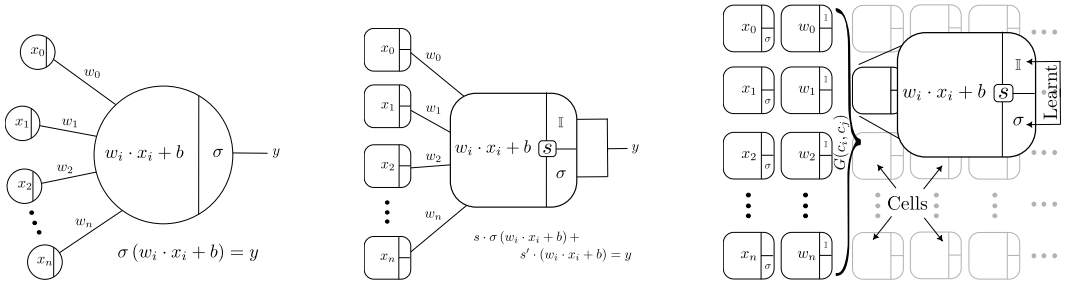
In this work, we show that our proposed Neurocell framework is a modern deep learning interpretation of CeNNs that is an extension of neural operators to a cellular array topology, in which learning Green's functions is simplified to regular convolution operations that known to be easier to optimize than differentiable logic gates (see Figure 1). The resulting VNNs will be shown to generalize deep learning as cellular machines capable of learning its own architecture in the following sections, or equivalently its own CA rules and of universal computation with a direct mapping to a tape based self-engineered Turing machine (TM). To better understand and precisely highlight the differences between our proposed VNNs and related works, we provide a comparison table in Appendix B. The VNNs are the only system that can leverage differentiable, continuous learning capabilities similar to Neural Operators and SSMs, yet apply them to the spatial, rule-based philosophy of Cellular Automata and CeNNs for self-organizing using efficient convolutions. Unlike NCAs that require an entire MLP network per cell to learn neighbourhood rules for interactions, VNNs show that its self-organization and healing only requires a single state parameter per cell and propagate signals via standard convolutions that are easier to learn than logic gates when compared to Differential Logic Gates. This allows it to elegantly self-engineer macroscopic architectures (like MLPs or ALUs) directly onto a cellular grid giving network weights spatial meaning.

3 METHODS

In this section, the goal is to create a modern neural network framework, but on arrays of cells as John von Neuman originally intended. This involves creating a deep learning like neuron, but one that maintains its own state or role within the network called a Von Neumann Neuron. We extend this approach so that cells have learnable states and signal propagation letting them decide their role in the network (e.g. whether to activate or not etc.) and create a network with its own learnt architecture called VNNs. We then extend how signals propagate within the array allowing different neural operators to be learnt dictating how layers of cells can mix and connect in arbitrary dimensions. We will end with a description of the experiments used to show the properties of this framework.

3.1 VON NEUMANN NEURON

The conventional deep learning artificial neuron has evolved from McCulloch and Pitts (1943) style design shown in Figure 2a. In this approach, a neuron activates on a biased weighted sum of inputs to



(a) Deep learning neuron. (b) Proposed neuron. (c) Proposed network on cells.

Figure 2: The artificial neuron (a) represents the typical deep learning definition, where inputs x_i are weighted by w_i and biased with b via (fixed) connections to this neuron then activated through the non-linear function σ when firing, noting our use of the Einstein summation convention. (b) shows the proposed state-based Von Neumann neuron, where s is the Codd state of the neuron and s' is the complement (usually $1 - s$) enabling them to individually learn whether to passing on the signal or activate. (c) shows the proposed VNN using these neurons embedded into a cellular topology, seamlessly allowing it to learn its own architecture suited to the objective function(s).

make a decision analogous to a non-linear switch. The network then involves a cascade of these non-linear decisions usually across fixed structures (such as multiple layers (Amari, 1967), trusses (Liu et al., 2019) or fabrics (Saxena and Verbeek, 2016)) of inter-connected neurons that are functionally equivalent to a perceptron (Rosenblatt, 1958). The universal approximation theorem then ensures that any non-linear function can be approximated by the network to arbitrary accuracy as long as the structure has at least a single layer of infinite number of neurons (Hornik et al., 1989; Cybenko, 1989) or many layers of finite number of neurons, i.e. it is deep enough. This approach assumes that the neurons are correctly connected into the optimal architecture beforehand and that every neuron is required to partake decisions for each input of the network at every layer. Although some neurons can be designated to 'dropout' of the decision making in a process defined beforehand, the former does not take into account an evolving architecture based on the objective function and the latter makes sub-architectures, where neurons can specialize by choosing to pass on the decision to other neurons to make, very difficult to obtain unless they are pre-defined into the architecture. Both require a hand crafted or pre-defined fixed architectures to be in place during training. Even neural architecture search models, such as fabrics or trusses (Saxena and Verbeek, 2016; Liu et al., 2019), have a weighted form of fixed, pre-defined (albeit a very large) number of weighted sub-architectures.

We propose extending the usual definition of the deep learning neuron in Figure 2a to a Von Neumann neuron that also contains a state s and use this state to enable neurons to toggle their role between passing on signals and traditional activation based decision making as shown in Figure 2b. Depending on the value of this Codd state s, the neuron will pass on the biased weighted sum only (i.e. applying an Identity $\mathbb{I}$ operation instead of the activation σ) acting as a connection between neurons only (letting them defer decision making to other connected neurons) or make a non-linear decision with the designated activation function σ as shown in Figures 2b and 2c. We designate this Codd state s as a learnable parameter and $s' = 1 - s$ taking inspiration from works utilizing contrastive loss and in the RealMLP (Holzmüller et al., 2024). The Von Neumann neuron will have two main advantages: Firstly, it enables the network weights w_n to be directly embedded with the array of cells and act as connections or signal neurons etc. instead of only being capable of decision making. As we'll show in the subsequent section, we can either then hard-code layers ($s = 1$) and connections ($s = 0$) directly within the array of cells to mimic traditional deep learning architectures or learn the states s directly through stochastic gradient descent. This will end up learning the network architecture with s and s' rather than hard-coding them. The network then cascades not only non-linear decisions but linear superimposed signals across dynamic structures adjusted on the fly depending on the objective function. The weights of the network can also be observed directly distributed spatially across the array during its use. The result is the system's ability to learn its own network architecture and its

many propagation functions or forms with Codd states s via stochastic gradient descent. Although our initial construction will be in view of neurons in arrays of cells in the subsequent section, this Von Neumann neuron can theoretically be used in any type of structured or unstructured (graphs etc.) topologies. In the next section, we describe how one can use this von Neumann neuron on array of cells to create Von Neumann Networks (VNNs) suited specifically for fast implementation on digital hardware as Johnny intended via current tensor based deep learning frameworks.

3.2 Von Neumann Networks

Johnny, after working with his close friend Stanisław M. Ulam (creator of the Monte Carlo method), believed that the cellular model of a neural network based brain analogue would be "more amendable to logical and mathematical treatment" (von Neumann, 1966, pg. 94). In this section, we propose such a construction through an extension of neural operators (Kovachki et al., 2023) on a cellular topology and the recently introduced Neural Green's Functions (Yoo et al., 2025) with our Von Neumann neurons of section 3.1 in creating a neural network on cellular arrays we denote as VNNs.

Approach. With our Von Neumann neuron as a cell (see section 3.1), the cell's influence on its surroundings is initially equivalent to a unit response or Dirac δ-function because it is isolated and unconnected. To build a network out of these cells, we need to expand the influence of each cell across the array keeping track of the cascading signals on a scalar field we will refer to as the Chua field C. This gradual smearing out of the neuron δ-functions can be modelled by using the correct propagation function or PSF similar to Huygen's principle, where many point sources interact to create a wavefront, so that the PSF then models the propagation of the cell signals. Since we eventually wish to utilize stochastic gradient descent to solve for the network connections via back propagation, the backward operator is a linear differential operator $\mathcal{D}$, this propagator will be the Green's function G and the forward pass is the integral form of the differential operator involving this Green's function. We will show that this formulation is an extension of neural operators (Kovachki et al., 2023) to a cellular topology, its implementation via convolutions (Li et al., 2020) and a cellular form of the recently introduced Neural Green's Functions (Yoo et al., 2025).

Formulation. Let each cell c in the cellular array we call the Chua field C be defined by the Von Neumann neuron of section 3.1 and Figure 2b. Given the universal approximation theorem, each cellular neuron c contributes a small non-linear superposition into the final approximation by 'divvying up' these contributions (Hornik et al., 1989; Cybenko, 1989). Following (Kovachki et al., 2023), let these contributions be represented the application of a linear differential operator $\mathcal{D}$ to our Chua field C driven the by source $f(c)$ that represents the inputs as

$$\mathcal{D}C(c) = f(c). \tag{1}$$

Then the contribution or response of each cell with input cell c_i at some other cell c is given by $\delta(c - c_i)$ and depends on the response of the system called its Green's function G (Arfken and Weber, 2001) as

$$\mathcal{D}G(c, c_i) = \delta(c - c_i). \tag{2}$$

The linearity of operator $\mathcal{D}$ allows the solution for any source $f(c)$ to be constructed via the total superposition of these contributions via the integral

$$C(c) = \int_{\Omega} G(c, c_i) f(c_i) dc, \tag{3}$$

over the sub-field with boundary Ω, thereby constructing the inverse operator $\mathcal{D}^{-1}$. Numerically, equation (3) is essentially the forward pass of the network given a suitable function G and equation (1) is applied during back-propagation via automatic differentiation on the computational graph. It is important to note that in general the Green's function G is independent of the sub-field with boundary Ω and the source term (and thus inputs) $f(c)$ (Yoo et al., 2025). Thus, the final form of the field C will contain the various input and output sources at the desired locations when optimizing the objective function and the regions in-between will form the "layers" of the VNN according to the final Codd states s. The Green's function the VNN typically acts as a propagator or PSF of the system that enhances or reduces the effects of the source $f(c)$ with respect to the different points in the field (Arfken and Weber, 2001, pg. 550), and in our case it propagates the signals across the field C to form connections and/or activations of the Von Neumann neurons comprising the cells c according to the objective function and learnt Codd states s (see Figure 2c).

Implementation. To implement the VNN, we utilize a tensor form of equations (1)and (3), the Chua field as a tensor $\boldsymbol{C}$ and the discrete forms of the operators $\boldsymbol{\mathcal{D}}$ & $\boldsymbol{G}$. Since the VNN is embedded within the cells, the Chua field $\boldsymbol{C}$ contains the input and outputs and are not restricted to any particular shape or number. As neural operators can be represented as convolution kernels (Li et al., 2020), we are able to use convolutions in representing the Green's functions G as large kernels and also learn multiple discrete Green's functions $\boldsymbol{G}_j$ for the system by adapting well known general convolution operators in various deep learning frameworks such as JAX (Bradbury et al., 2018). The weights w, biases b and Codd states s are present in each of the cells, so that we may use the discrete fields for weights $\boldsymbol{W}$, biases $\boldsymbol{B}$ and states $\boldsymbol{S}$ respectively to represent the main constituents of the VNN. Finally, the integration of the entire sub-field with boundary Ω will need to be broken down into multiple discrete steps that depend on the distance between the various inputs & outputs, as well as the size of the kernel representing the Green's functions $\boldsymbol{G}_j$. To compute a step of the forward pass of the randomly initialized VNN with inputs represented as the discrete spatial distribution $f[\boldsymbol{c}]$ within $\boldsymbol{C}$, we convolve the field $\boldsymbol{C}^{(n)}$ at timestep t_n with $\boldsymbol{G}_j$ as $\boldsymbol{C}_j^{(n)} = \boldsymbol{G}_j * \boldsymbol{C}^{(n)}$. After which there are two possibilities depending on the Codd state field $\boldsymbol{S}$ assuming it is gated to produce $s \in [0, 1]$ per cell. Firstly, there are cells which choose to 'pass-through' the signal having been weighted by $\boldsymbol{W}$ and will not activate. Secondly, there are cells that decide to activate by σ given its weighted neighbours. In the latter, the signals passed through by neighbouring cells already have the weights applied, so that the next timestep $t^{(n+1)}$ can be given by

$$\boldsymbol{C}^{(n+1)} = \frac{1}{M} \sum_{j}^{M-1} \left[(\boldsymbol{I} - \boldsymbol{S}) \cdot \left(\boldsymbol{C}_j^{(n)} \cdot \boldsymbol{W} \right) + \boldsymbol{S} \cdot \sigma \left(\boldsymbol{C}_j^{(n)} + \boldsymbol{B} \right) \right], \tag{4}$$

where M is the total number of Green's functions with $0 \leqslant j < M$, $\cdot$ and $*$ represent the element-wise (Hadamard) product and convolution operations respectively, and $\boldsymbol{I}$ is the identity field (i.e. all ones). The total number of steps depends on the desired output locations and size of the Green's kernel with the maximal distance that such a kernel can cover is $(k-1)/2$, where k is the kernel size (assuming k is odd and $\boldsymbol{C}$ size to be even) in the direction of propagation to ensure information is interconnected between each step. Note that the averaging across j does not need to be done per step and can be done at the final step for maximal feature representation at the cost of computational complexity and memory. At first it may seem that need for these steps seem unusual, but they effectively represent the computation of each "layer" of the network analogous to the matrix-vector multiplication in a deep learning MLPs (see Appendix F), the structure of which can be controlled depending on how the Green's functions are applied to form different VNN architectures.

Architectural Forms. One of the chief advantages of VNNs is the simplicity of representing network architectures, especially for MLPs in arbitrary dimensions as their connectivity in this space have the same forward (and thus backward) operators regardless of its dimension. In this work, we present two forms of VNNs based on either assuming that the inputs and outputs have an MLP-like structure as hyperplanes (i.e. for a 2D field C, these are row or column vectors) or not making any assumptions about these structures at all. In each case, to form the actual network architectures themselves, we need to either hand-craft the Codd states s into the desired form or learn the architecture itself given a particular kernel shape and application of the Green's functions that dictate the propagation of the neural signals and therefore define the local connectivity of the Von Neumann neurons. A discussion of the former is made in Appendix F to show the correspondence of the proposed theory to the well established deep learning theory. The latter is discussed in the remainder of this section.

Let the $h^{(n)}(c)$ be a function of cells c at layer n in a hyperplane structure of dimension $d-1$ for a Chua field $\boldsymbol{C}$ of dimension d or simply $h^{(n)}$ for brevity. We assume that the forward pass is towards the ascending indices in the channel dimension and adopt the channel last convention. We also limit the Green's function $\boldsymbol{G}$ to this structure, but with a width ω in the channel dimension to obtain a hyperplane layer specific Green's functions $\bar{\boldsymbol{G}}_j$ with a step distance of ω for each layer. The same structure can be applied to the weights $\boldsymbol{W}$, biases $\boldsymbol{B}$ and states $\boldsymbol{S}$ to give $\overline{\boldsymbol{W}}$, $\bar{\boldsymbol{B}}$ and $\bar{\boldsymbol{S}}$. Then the Von Neumann update is a forward pass step, which is also limited to ω convolutions of hyperplanes of dimension $d-1$, gives the next layer $h^{(n+1)}$ as

$$h^{(n+1)} = \frac{1}{M} \sum_{j}^{M-1} \left[(\bar{\boldsymbol{I}} - \bar{\boldsymbol{S}}) \cdot \left(\bar{\boldsymbol{G}}_j * h^{(n)} \cdot \overline{\boldsymbol{W}} \right) + \bar{\boldsymbol{S}} \cdot \sigma \left(\bar{\boldsymbol{G}}_j * h^{(n)} + \bar{\boldsymbol{B}} \right) \right], \tag{5}$$

with convolution having the highest operator precedence. For the case that no structure can be or is assumed for the network and its convolution, the full Green's function is learnt and the convolution is always full d dimensional, therefore the layers of the network are also d dimensional in shape and the forward step is given by (4). Additional details for the forward step and model training procedure can be found in Appendix C. We may also incorporate information bottlenecks and other constraints on cells and the architecture through setting those states to $-\infty$ similar to its use in triangular matrices for computing attention in transformers. In the next section, we present the experiments conducted and theorem proved in this work to demonstrate the VNN in its various forms.

Experiments. To show that VNNs can support not only regular MLP vector like data, we construct circular layered perceptrons (CLPs) and tape-like MLPs models show the various different configurations of the inputs and outputs, and thus the networks, are possible in solving binary arithmetic including exclusive OR (XOR) operations. Models were trained to do 8-bit arithmetic between two inputs that result in an output of the same bit depth. To show that multi-dimensional VNN MLPs are simple and scalable, we present image classification results for the simple MNIST and CIFAR10 image datasets compared to regular deep learning MLPs. A simple three layer (256, 128, 32) and five layer (512, 256, 128, 32, 16) traditional deep learning MLPs models were compared with Hyperplanes ($d-1$ dimensional) and full (d dimensional) VNN models for MNIST and CIFAR10 respectively. To show that the VNN is capable of learning a unified model for all arithmetic and logic, we also create an arithmetic logic unit (ALU) model to demonstrate its ability to use the full d dimensional space of the Chua field C at once rather than the hyperplane structure. We compare a traditional MLP with both hyperplane and full VNNs for 8-bit arithmetic and logic given by a 4-bit control code to produce a single 8-bit output with carry bits ignored in this study. For the full model, we compare two configurations: one with a box like shape with the control code orthogonal to the inputs/outputs, so that most of the model is contained compactly within the space between them; and a tape-like model where all controls, inputs and outputs are configured in a line together. In the last experiment, we push the advantage VNNs of learning architectures into a network that has not discernible layers because the entire space is a 1D model on a 1D Chua field C with 1D Green's functions to emulate a TM like architecture with a tape like model, but with the Green's function acting like a learnable transition table or program showing its equivalence to a TM for later proofs.

Setup. Implementation of the Neurocell framework and VNNs were performed on the JAX numerical and auto-differentiation framework (Bradbury et al., 2018) (the code is provided as supplementary material). The experiments were carried out on NVIDIA RTX 3090 & 2080 Ti GeForce graphics cards (CUDA 12.8) and an AMD Radeon 7900 XTX (ROCm 6.1) graphics card. All results were verified across these cards, and CIFAR experiments were also run on an NVIDIA A100. The JAX grad transform was used to perform the backpropagation and the AdamW optimizer (Loshchilov and Hutter, 2018) with a cosine decay learning rate scheduler starting from $1e^{-3}$ wherever possible.

4 RESULTS

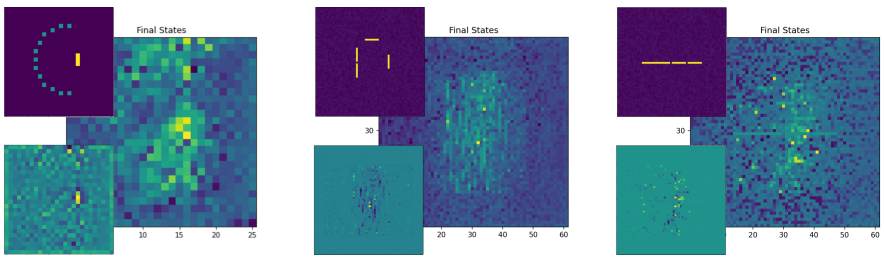


Figure 3: The resulting VNNs final states (thus the architecture) for CLP (left) and two different ALU models. The inputs/outputs and controls in the case of the ALU models are shown in the top left insets with the bottom right showing the final weights of the models.

Experiments. The various types of initial network shapes explored in this work and the resulting VNN final model states for the CLP and ALU models using the full Green's functions approach are shown in Figure 3 (see Appendix G for the resulting Green's functions and ablations on learning architecture in Appendix D). The CLP model shows VNNs using a concave shape for the input in a

simple classification task (Aeberhard et al., 1994). Figure 1 (right) shows the resulting MNIST model and its data-driven encoder-like structure determined automatically during training. Indeed, the VNN supports arbitrary shapes for its inputs/outputs as long as there are sufficient steps taken between them to ensure that signals are fully connected. Table 1 shows how the VNN MLP compares against the deep learning MLP on various datasets and tasks. Applications of VNNs to well suited problems is presented in Appendix E, including hardware-like repairable networks and simulating the GoL.

Table 1: Comparison of traditional MLPs and VNNs performance across various datasets and tasks

Dataset	Task	Model	Acc. (%)	# Params	Inf. Time (s)	Train Time (s)	Throughput (samples/s)
MNIST	Classify	MLP	96.5	244.5K	2.8921	151.70	3,457.66
		Logistic Regression	92.6	7.9K	0.0469	156.68	213K
		Random Forest	96.8	707.6K	0.1251	3.91	80K
		VNN	96.7	41.2K	4.4076	562.57	2,268.79
CIFAR10	Classify	MLP	62.4	11.1M	11.2762	1614.03	886.82
		VNN 3D	71.9	841.7K	15.4264	8789.62	648.24
		VNN 3D Full	77.7	1.2M	25.7458	18750.47	388.41
ALU	Arithmetic	MLP	66.8	2.0K	2.9845	295.85	67.01
		VNN	69.9	7.9K	2.3770	661.08	84.14
		VNN 2D Full	99.8	16.3K	6.1487	13154.22	32.53
XOR	Arithmetic	MLP	100.0	36.2K	0.7814	21.19	8,190.13
		Linear Regression	1.1	136	0.0003	0.01	20.2M
		Polynomial LR	66.3	7.8K	0.0136	0.80	471K
		Random Forest	36.3	6.1M	0.2179	12.44	30K
		VNN	100.0	3.3K	0.5438	113.28	91.94

The ALU model shown in Figure 3 (middle) is shown in Table 1 as VNN Full and is able to learn the task that neither the deep learning MLP nor the VNN MLP could. This is likely due to the extra 2D connectivity that the Full VNN supports for a 2D Green's function rather than a hyperplane on the Chua field C, which was actually predicted by Johnny (von Neumann, 1958). The ALU model in Figure 3 (right) shows that this cannot be caused by the 2D layout of the input/outputs and control alone, since all these elements are entirely in 1D in this model, but rather the 2D instructions that appear to be necessary. More importantly, this ALU model shows that together with the input/outputs and memory, the Full VNN is capable of simulating the entire Von Neumann (hardware) architecture (von Neumann, 1993, first published in 1945) as a Everything on Chip (EoC) design as opposed to a traditional System on Chip (SoC) system. The primary advantage of such a EoC design would be the ability to 'repair' the chip by fine-tuning the model after designating the damaged cells as $-\infty$ similar to its use in computing attention. Finally, the vertical and horizontal structures imprinted on the final states of both ALU models are purely data-driven and requires further exploration. Figure 4 shows that fact all one needs is a tape (1D) structure, where the resulting

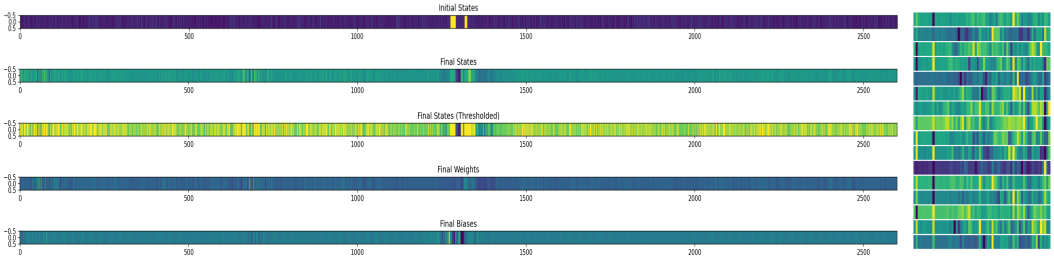


Figure 4: The resulting VNN tape based model (purely 1D Chua field C with 1D Green's functions and 1D convolutions) for XOR arithmetic of two 8-bit numbers. The 16 "instructions" learnt as 1D Green's functions for the model are shown on the right.

tape based model is from only 1D operations on a 1D field C for 8-bit XOR arithmetic. The resulting

model and setup is very reminiscent of a Turing machine (TM) and the Green's function represent the instructions learnt for the task (see Figure 4 (right)). Additional work is required in studying these instructions and applying them to other unsolved problems in TMs such as the Busy Beaver (Rado, 1962). Lastly, although the CIFAR10 VNN results outperform the traditional MLP, they are still far from state-of-the-art to Convolutional Neural Networks (CNNs) such as the ResNet that can achieve 94% accuracy (He et al., 2016, see DAWNBench Challenge). Further work is required in constructing CNN-like VNNs that are translational/rotational equivariant to reach that level of performance.

Theorems. We construct three proofs that show VNNs are Turing complete (see a brief basic introduction to TMs in Appendix H). The first of this type can be found in Appendix F that shows VNNs have a direct correspondence to deep learning and hence are also universal. The second proof is based on showing GoL equivalence to VNNs, see Proposition 2 in Appendix E.2. Finally, our VNN is a cellular machine (CM) that features stochastic gradient descent to learn its own program.

Definition 1 (Cellular Machine)**.** *A CM is an infinite array of cells in which each cell maintains its own state and connectivity defined by neighborhood rules via a set of Green's functions that characterizes it, and in conjunction with the individual cell states, create the transition rules. The inverse Green's functions can be used to backpropagate signals and learn its own program.*

Proposition 1 (Machine-based Universality)**.** *The cellular machine (CM) is equivalent to a TM that is Turing complete by being able to simulate the Von Neumann (hardware) architecture.*

Proof. We observe that a CM is equivalent to a TM with the cells themselves being the (multi-dimensional) tape of the TM, the learnt states are the states of the TM, the Green's functions are the program and the convolutions are analogous to operations of the read/write head moving across the tape (see AI-assisted LEAN verification in the supplementary material). To show that a CM is universal and capable of being a universal Turing machine (UTM), consider the ALU model presented in Figure 3 (middle) that shows a CM can possess input/outputs, the ALU and hold memory within its cells, making it equivalent to a Von Neumann (hardware) architecture that is Turing complete. ◂

Discussion. The unique properties of VNNs with respect to related works are that they are the only fully differentiable, continuous learning capabilities of Neural Operators, while reducing the Neural CA requirements of an MLP per cell to just a single state parameter and use convolutions rather than more challenging logic gates to grow data-driven network architectures. The architecture ablations in Appendix D show that VNNs only need a minimum grid and kernel size for the problem at hand, and the convergence rate is dictated by the number of Green's functions. A minimum grid and kernel sizes are required to hold the inputs and connect the neurons, upon which any size greater can be used at the cost of efficiency and computation time. In fact, Figure 4 that shows a VNN tape based model, where the dimensionality of the cellular array need not be a hyper-parameter either, which fits in accordance to computational theory that only needs 1D tape to accomplish all computation.

Limitations and Future Work. The current Neurocell framework relies on learning Green's functions using convolutions, but the kernel sizes tend to be large (> 13), hence support for large kernel sizes need to be explored, such as scaling methods (Ding et al., 2022) or dilated approximations (Guo et al., 2023). The states being optimized are not strictly discrete like CAs, perhaps quantized states (Hwang and Sung, 2014) could improve efficiency or a study similar to Li et al. (2017). Further suggested studies include adapting other deep learning models such as transformers and CNNs, extensions of the Von Neumann neuron to other topologies, where convolution operators exist such as graphs (Kipf and Welling, 2017), as well as learning to construct self-replicating systems and artificial life.

CONCLUSION

This work presented VNNs that uses state-based neurons as cells on an infinite array of cells in any dimension that learn their role in the network allowing for them to self-engineer their own architecture. The result is a computationally universal system capable of networks with unstructured layers that can embed a TM with a tape structure and a 'healable' Von Neumann (hardware) architecture through model training. Future work will involve creating intricate multi-agent systems, automating CAs and eventually constructing models that can create self-replicating machines on demand.

AI USE STATEMENT

In this work, we used generative AI tools for checking grammar, formatting LaTeXtables and generic wordsmithing, as well as assistance in the writing the LEAN verification of a mathematical proof. We have not used generative AI tools for developing theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, implementing methods, supporting qualitative and thematic data analysis or interpreting results. Finally generating synthetic data sets, assistance with translation and cleaning, and reformat dataset are not applicable to this work. Additionally, we used generative AI tools for some of the summarization or analysis of existing literature, as well as suggest experimental parameters and edit some of the hand-crafted code for ablations and comparisons. We have reviewed all AI-assisted work. For example, we have proof read the manuscript, code documentation and checked the generated LEAN verification code. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not include any studies that involve human subjects. The authors have no potential conflicts of interest and sponsorship to declare.

REPRODUCIBILITY STATEMENT

We have provided all of the source code of the Neurocell framework, its various models, ablations and experiments, which has been submitted as supplementary materials. The appendix has been crafted to make the work as self-contained as possible with additional theoretical materials and results with detailed explanations of assumptions, as well as architectural and experimental details have been included in the appendix. Datasets used in the experiments are mostly benchmark datasets and any synthetic data are binary sequences with the relevant code provided (for XOR etc.). A complete description of the data processing steps are also provided in the supplementary materials via the code and some are also in the appendix.

REFERENCES

Aeberhard, S., Coomans, D., de Vel, O., 1994. Comparative analysis of statistical pattern recognition methods in high dimensional settings. Pattern Recognition 27, 1065–1077. doi:10.1016/0031-3203(94)90145-7.

Amari, S., 1967. A Theory of Adaptive Pattern Classifiers. IEEE Transactions on Electronic Computers EC-16, 299–307. doi:10.1109/PGEC.1967.264666.

Arfken, G.B., Weber, H., 2001. Mathematical Methods for Physicists. 5th ed., Academic Press Inc, San Diego.

Banks, E.R., 1971. Information Processing and Transmission in Cellular Automata Accepted: 2023-03-29T14:55:25Z.

Bhattacharya, A., 2021. The Man from the Future: The Visionary Life of John Von Neumann. W. W. Norton & Company, New York.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M.J., Katariya, Y., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., Zhang, Q., 2018. JAX: composable transformations of Python+NumPy programs. URL: http://github.com/jax-ml/jax.

Buckley, W.R., 2008. Signal crossing solutions in von Neumann self-replicating cellular automata., in: Automata, pp. 453–503.

Chan, B.W.C., 2020. Lenia and Expanded Universe, in: The 2020 Conference on Artificial Life, pp. 221–229. doi:10.1162/isal_a_00297. arXiv:2005.03742 [nlin].

Chua, L., Yang, L., 1988. Cellular neural networks: applications. IEEE Transactions on Circuits and Systems 35, 1273–1290. doi:10.1109/31.7601.

Chua, L.O., Gulak, G., Pierzchala, E., Rodríguez-Vázquez, A. (Eds.), 1998. Cellular Neural Networks and Analog VLSI. Springer US, Boston, MA. doi:10.1007/978-1-4757-4730-0.

Codd, E.F., 1968. Cellular Automata. Academic Press. Google-Books-ID: W_tfAwAAQBAJ.

Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems 2, 303–314. doi:10.1007/BF02551274.

Ding, X., Zhang, X., Han, J., Ding, G., 2022. Scaling Up Your Kernels to 31x31: Revisiting Large Kernel Design in CNNs, pp. 11963–11975.

Gardner, M., 1970. Mathematical Games.

Gers, F., de Garis, H., Korkin, M., 1998. CoDi-1Bit : A simplified cellular automata based neuron model, in: Hao, J.K., Lutton, E., Ronald, E., Schoenauer, M., Snyers, D. (Eds.), Artificial Evolution, Springer, Berlin, Heidelberg. pp. 315–333. doi:10.1007/BFb0026610.

Gilpin, W., 2019. Cellular automata as convolutional neural networks. Physical Review E 100. doi:10.1103/PhysRevE.100.032402.

Graves, A., Wayne, G., Danihelka, I., 2014. Neural Turing Machines. doi:10.48550/arXiv.1410.5401. arXiv:1410.5401 [cs].

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S.G., Grefenstette, E., Ramalho, T., Agapiou, J., Badia, A.P., Hermann, K.M., Zwols, Y., Ostrovski, G., Cain, A., King, H., Summerfield, C., Blunsom, P., Kavukcuoglu, K., Hassabis, D., 2016. Hybrid computing using a neural network with dynamic external memory. Nature 538, 471–476. doi:10.1038/nature20101. publisher: Nature Publishing Group.

Guo, M.H., Lu, C.Z., Liu, Z.N., Cheng, M.M., Hu, S.M., 2023. Visual attention network. Computational Visual Media 9, 733–752. doi:10.1007/s41095-023-0364-2.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep Residual Learning for Image Recognition, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778. doi:10.1109/CVPR.2016.90.

Holzmüller, D., Grinsztajn, L., Steinwart, I., 2024. Better by default: Strong pre-tuned MLPs and boosted trees on tabular data.

Hong Kong, Chan, B.W.C., 2019. Lenia: Biology of Artificial Life. Complex Systems 28, 251–286. doi:10.25088/ComplexSystems.28.3.251.

Hornik, K., Stinchcombe, M., White, H., 1989. Multilayer feedforward networks are universal approximators. Neural Networks 2, 359–366. doi:10.1016/0893-6080(89)90020-8.

Hutton, T.J., 2010. Codd's Self-Replicating Computer. Artificial Life 16, 99–117. doi:10.1162/artl.2010.16.2.16200.

Hwang, K., Sung, W., 2014. Fixed-point feedforward deep neural network design using weights +1, 0, and -1, in: IEEE Workshop on Signal Processing Systems (SiPS), pp. 1–6. doi:10.1109/SiPS.2014.6986082.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S.A.A., Ballard, A.J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A.W., Kavukcuoglu, K., Kohli, P., Hassabis, D., 2021. Highly accurate protein structure prediction with AlphaFold. Nature 596, 583–589. doi:10.1038/s41586-021-03819-2. publisher: Nature Publishing Group.

Kipf, T.N., Welling, M., 2017. Semi-Supervised Classification with Graph Convolutional Networks. URL: https://openreview.net/forum?id=SJU4ayYgl.

Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., Anandkumar, A., 2023. Neural operator: learning maps between function spaces with applications to PDEs. J. Mach. Learn. Res. 24, 89:4061–89:4157.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. ImageNet Classification with Deep Convolutional Neural Networks, in: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (Eds.), Advances in Neural Information Processing Systems 25. Curran Associates, Inc., pp. 1097–1105.

Langton, C.G., 1984. Self-reproduction in cellular automata. Physica D: Nonlinear Phenomena 10, 135–144. doi:10.1016/0167-2789(84)90256-2.

LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521, 436–444. doi:10.1038/nature14539. number: 7553 Publisher: Nature Publishing Group.

Li, H., De, S., Xu, Z., Studer, C., Samet, H., Goldstein, T., 2017. Training Quantized Nets: A Deeper Understanding, in: Advances in Neural Information Processing Systems, Curran Associates, Inc.

Li, Z., Kovachki, N.B., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., Anandkumar, A., 2020. Fourier Neural Operator for Parametric Partial Differential Equations, in: International Conference on Learning Representations.

Liu, C., Chen, L., Schroff, F., Adam, H., Hua, W., Yuille, A.L., Fei-Fei, L., 2019. Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation, in: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 82–92. doi:10.1109/CVPR.2019.00017. iSSN: 2575-7075.

Liu, Z., Mao, H., Wu, C.Y., Feichtenhofer, C., Darrell, T., Xie, S., 2022. A ConvNet for the 2020s, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 11976–11986. doi:10.1109/CVPR52688.2022.01167.

Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljacic, M., Hou, T.Y., Tegmark, M., 2024. KAN: Kolmogorov–Arnold Networks, in: Neural Information Processing Systems. URL: https://openreview.net/forum?id=Ozo7qJ5vZi.

Loshchilov, I., Hutter, F., 2018. Decoupled Weight Decay Regularization. URL: https://openreview.net/forum?id=Bkg6RiCqY7.

Lu, Z., Sreekumar, G., Goodman, E., Banzhaf, W., Deb, K., Boddeti, V.N., 2021. Neural Architecture Transfer. IEEE Transactions on Pattern Analysis and Machine Intelligence 43, 2971–2989. doi:10.1109/TPAMI.2021.3052758. conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

Mange, D., Stauffer, A., Peparaolo, L., Tempesti, G., 2004. A macroscopic view of self-replication. Proceedings of the IEEE 92, 1929–1945. doi:10.1109/JPROC.2004.837631. conference Name: Proceedings of the IEEE.

McCulloch, W.S., Pitts, W., 1943. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics 5, 115–133. doi:10.1007/BF02478259.

Miotti, P., Niklasson, E., Randazzo, E., Mordvintsev, A., 2025. Differentiable Logic Cellular Automata: From Game of Life to Pattern Generation. doi:10.48550/arXiv.2506.04912. arXiv:2506.04912 [cs].

Moore, C., Mertens, S., 2011. The Nature of Computation. 1 edition ed., Oxford University Press, Oxford New York Auckland Caoe Town Dar es Salaam.

Mordvintsev, A., Niklasson, E., 2021. μNCA: Texture Generation with Ultra-Compact Neural Cellular Automata. doi:10.48550/arXiv.2111.13545. arXiv:2111.13545 [cs].

Mordvintsev, A., Randazzo, E., Niklasson, E., Levin, M., 2020. Growing Neural Cellular Automata. Distill 5, e23. doi:10.23915/distill.00023.

von Neumann, J., 1958. The Computer and the Brain. Yale University Press. Google-Books-ID: hvPAUX0glAYC.

von Neumann, J., 1966. Theory of self-reproducing automata / Edited and completed by Arthur W. Burks. University of Illinois Press, Urbana Ill.}.

von Neumann, J., 1993. First draft of a report on the EDVAC. IEEE Annals of the History of Computing 15, 27–75. doi:10.1109/85.238389.

von Neumann, J., 2018. Mathematical Foundations of Quantum Mechanics: New Edition: 58. Princeton University Press, Princeton.

Pajouheshgar, E., Xu, Y., Mordvintsev, A., Niklasson, E., Zhang, T., Süsstrunk, S., 2024. Mesh Neural Cellular Automata. doi:10.48550/arXiv.2311.02820. arXiv:2311.02820 [cs].

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830.

Petersen, F., Borgelt, C., Kuehne, H., Deussen, O., 2022. Deep differentiable logic gate networks, in: Proceedings of the 36th International Conference on Neural Information Processing Systems, Curran Associates Inc., Red Hook, NY, USA. pp. 2006–2018.

Pérez, J., Marinković, J., Barceló, P., 2018. On the Turing Completeness of Modern Neural Network Architectures. URL: https://openreview.net/forum?id=HyGBdo0qFm.

Rado, T., 1962. On Non-Computable Functions. Bell System Technical Journal 41, 877–884. doi:10.1002/j.1538-7305.1962.tb00480.x. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/j.1538-7305.1962.tb00480.x.

Rao, Y., Zhao, W., Zhu, Z., Zhou, J., Lu, J., 2023. GFNet: Global Filter Networks for Visual Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 45, 10960–10973. doi:10.1109/TPAMI.2023.3263824.

Rendell, P., 2013. A Fully Universal Turing Machine in Conway's Game of Life. Journal of Cellular Automata 8, 19–38.

Rosenblatt, F., 1958. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review 65, 386–408. doi:10.1037/h0042519. place: US Publisher: American Psychological Association.

Roska, T., Chua, L., 1993. The CNN universal machine: an analogic array computer. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 40, 163–173. doi:10.1109/82.222815.

Sabour, S., Frosst, N., Hinton, G.E., 2017. Dynamic routing between capsules, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, Curran Associates Inc., Red Hook, NY, USA. pp. 3859–3869.

Saxena, S., Verbeek, J., 2016. Convolutional Neural Fabrics, in: Advances in Neural Information Processing Systems, Curran Associates, Inc.

Sipper, M., 1998. Fifty Years of Research on Self-Replication: An Overview. Artificial Life 4, 237–257. doi:10.1162/106454698568576. conference Name: Artificial Life.

Turing, A.M., 1937. On Computable Numbers, with an Application to the Entscheidungsproblem. Proceedings of the London Mathematical Society s2-42, 230–265. doi:10.1112/plms/s2-42.1.230.

Turing, A.M., 1990. The chemical basis of morphogenesis. Bulletin of Mathematical Biology 52, 153–197. doi:10.1007/BF02459572.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I., 2017. Attention is all you need, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, Curran Associates Inc., Red Hook, NY, USA. pp. 6000–6010.

Wang, L., Xie, S., Li, T., Fonseca, R., Tian, Y., 2021. Sample-Efficient Neural Architecture Search by Learning Actions for Monte Carlo Tree Search. IEEE Transactions on Pattern Analysis and Machine Intelligence , 1–1doi:10.1109/TPAMI.2021.3071343. conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

Wang, X., Zhu, W., 2024. Advances in neural architecture search. National Science Review 11, nwae282. doi:10.1093/nsr/nwae282.

Wolfram, S., 1983. Statistical mechanics of cellular automata. Reviews of Modern Physics 55, 601–644. doi:10.1103/RevModPhys.55.601. publisher: American Physical Society.

Yoo, S., Yeo, K., Hwang, J., Sung, M., 2025. Neural Green's Functions, in: Neural Information Processing Systems. URL: https://openreview.net/forum?id=t2wMLGmcXd.

Zoph, B., Le, Q., 2017. Neural Architecture Search with Reinforcement Learning. URL: https://openreview.net/forum?id=r1Ue8Hcxg.

A Brief Review of Self-replication

John von Neumann used 29 different states in his original formulation to imbue his cellular neurons with special functions during the self-replication process of his machine, where the machine encodes instructions of a universal Turing machine that eventually computes the replication (von Neumann, 1966). Significant progress has been made in implementing (Mange et al., 2004; Buckley, 2008) and extending self-replicating machines of von Neumann in the decades following his work. Codd (1968), with subsequent recent amendments from Hutton (2010), reduced the requirement of self-replication requiring 29 states to a total of only 8 states. Others have further simplified self-replication to fewer states, reduced the dimensionality, and removed universal requirements (Banks, 1971; Wolfram, 1983; Langton, 1984). A more detailed review of this area can be found in Sipper (1998).

More recent work has focused on simpler CA with as few rules as possible, while maintaining complex behaviour. The most famous of these is John Conway's Game of Life (Gardner, 1970), where the cells have only 2 states and rules that change the state of cells based on its neighbours. Surprisingly, even this simple 2-state cellular system is Turing complete (Rendell, 2013). More complex rules have been developed to simulate artificial life (Hong Kong and Chan, 2019) and are the subject of ongoing research (Chan, 2020).

B Related Works Comparison

The proposed VNNs are unique in that we show that a neural network is possible where a simple mechanism, namely a singular learnt parameter called the Codd state s within a neuron allows that neuron to adapt its role in a network, enabling a data-driven architecture to be trained. In the table below we precisely highlight these differences between VNNs and related works.

Model / Approach	Continuous vs Discrete	Differentiability	Hardware Efficiency	Interpretability	Architecture Learning	Core Mechanism
VNNs	Continuous (can approximate discrete)	Fully Differentiable	High (maps well to parallel & spatial hardware)	Moderate (visualizable spatial weights/states)	Yes (cells learn and heal)	Learnable Green's functions & cell Codd states
Neural Operators	Continuous	Fully Differentiable	Moderate to High	Low	No (Fixed Topology)	Mapping between infinite-dimensional function spaces
Cellular Automata (e.g., GoL)	Discrete	Non-Differentiable	High (simple bitwise operations)	High (rules are explicit)	No (Emergent patterns, but rules are fixed)	Hand-crafted local transition rules
CeNNs	Continuous (Analog)	Differentiable (ODE based)	High (Analog VLSI)	Moderate	No (Fixed templates)	Analog ODEs with local cloning templates
Neural CA	Continuous	Fully Differentiable	Low (MLP per cell is heavy)	Low	Yes (Morphogenesis/Healing)	Per-cell MLPs predicting local state changes
Diff. Logic Gates	Continuous (train) / Discrete (infer)	Differentiable (via relaxations)	Very High (pure logic inference)	High (resolves to logic circuits)	No (Usually searches within fixed supergraph)	Continuous relaxation of Boolean operations
Random Forests	Discrete	Non-Differentiable	Moderate (Branching paths)	Very High	Yes (Tree structure is grown)	Information Gain / Gini impurity splits

Model / Approach	Continuous vs Discrete	Differentiability	Hardware Efficiency	Interpretability	Architecture Learning	Core Mechanism
State Space Models	Continuous	Fully Differentiable	High (Linear recurrence)	Low	No (Fixed state sizes and sequence flow)	Continuous-time ODEs for temporal sequences

As far as the author is aware, none of these systems is able to utilize a single 1D model for all layers and weights/biases like the VNNs as shown in 1D tape model in Figure 4. The 1D CA of Wolfram is similar in nature but is non-differentiable. In Figure 4 of the paper, the VNN tape model embeds the entire network topology: inputs, outputs, intermediate representations, weights/biases into a single, unified 1D spatial array (the 1D Chua field). The computation then temporally unfolds through 1D convolutions, where the learnt 1D Green's functions act as the transition rules, blurring the line between spatial layers and time like a Turing Machine tape. In otherwords, a VNN can dynamically learn to route signals back and forth across a single 1D space. All other models require at best multiple distinct layers of additional 1D or 2D blocks stacked together to form their respective models.

Significance. VNNs have the potential to be one of the few ways to achieve neuron-level yet hardware-native form of artificial intelligence capable of self-engineering and healing (see Appendix E for evidence of some of these applications).

C Algorithms and Training Procedure

We provide the training procedure with algorithm tables for the forward pass operator in Algorithm 1 and the training procedure in Algorithm 2 as shown below, as well as tables for the architectural choices.

Algorithm 1 VNN Forward Pass

Require: Cellular field $C^{(0)}$, Weights W, Biases B, States S, Green's functions $\{G_j\}_{j=1}^{M}$, Depth d
Require: Activation function σ
for $n = 0$ **to** $d - 1$ **do**
  Initialize $C^{(n+1)} \leftarrow 0$
  for $j = 1$ **to** M **do**
    Compute propagated field: $C_j^{(n)} = G_j * C^{(n)}$
    Apply standardization/layer normalization (optional) on $C_j^{(n)}$
    Compute passive signal: $P_j = (I - S) \cdot (C_j^{(n)} \cdot W)$
    Compute active decision: $A_j = S \cdot \sigma(C_j^{(n)} + B)$
    Blend states: $F_j = P_j + A_j$
    Accumulate: $C^{(n+1)} \leftarrow C^{(n+1)} + F_j$
  end for
  Average across filters: $C^{(n+1)} \leftarrow \frac{1}{M} C^{(n+1)}$
end for
return $C^{(d)}$

Algorithm 1 shows that VNNs are trained using standard Stochastic Gradient Descent (SGD) and backpropagation, treating the cellular machine as a differentiable computation graph. During training, the backward pass simulates the inverse linear differential operator $\mathcal{D}^{-1}$, distributing gradients back through the continuous cellular field.

Algorithm 2 VNN Training Procedure

Require: Training dataset $\mathfrak{D}$, Learning rate schedule, Optimizer (e.g., AdamW)

Initialization:
Initialize cellular universe field $C \leftarrow 0$
Initialize Weights $W \sim \mathcal{N}(\mu, \sigma^2)$ and Biases $B \leftarrow 0$
Initialize States S randomly, then bound via sigmoid $S \leftarrow \sigma_{sig}(S_{raw})$
Fix $S \leftarrow 1.0$ at input and output coordinates
Initialize Green's functions G_j
while not converged **do**
  Sample a minibatch of inputs X and targets Y from $\mathfrak{D}$
  Place X at designated input coordinates in $C^{(0)}$
  Compute forward pass $\hat{Y} = \text{VNN_Forward}(C^{(0)}, W, B, S, \{G_j\})$ sampled at output coordinates
  Compute loss $\mathcal{L}(\hat{Y}, Y)$
  Compute gradients $\nabla_{W,B,S_{raw},G_j}\mathcal{L}$ via reverse-mode auto-differentiation
  Update parameters W, B, S_{raw}, G_j using Optimizer
end while
return $W, B, S, \{G_j\}$

Architectural Design

Parameter / Decision	MNIST VNN MLP	CIFAR10 VNN 3D Hyperplane	VNN ALU Chip
Dataset	MNIST	CIFAR-10 (via Grain Dataloader)	Synthetic ALU Arithmetic (128k samples)
Epochs	8	35	200
Batch Size	64	100	128
Layers	4	20	N/A
Network Depth	9 (`layers * 2 + 1`)	40 (`layers * 2`)	4
Kernel Size	`(71, 3)`	`(9, 9, 41)` (channel first, depth last)	`(23, 13)`
Directions (Green's function)	2	32	16
Universe Grid Size (N)	1024×17 ($N_x \times N_y$)	$44 \times 44 \times 89$ ($N_x \times N_y \times N_z$)	62×62 (Square Chip Layout)
Optimizer	Adam	AdamW	AdamW
Learning Rate	`0.01` (Constant)	Warmup Cosine Decay (`peak=0.03`, `init=0.001`, `end=0.0005`)	Warmup Cosine Decay (`peak=0.03`, `init=0.001`, `end=0.0005`)
Activation (Hidden)	`jax.nn.tanh`	`jax.nn.leaky_relu`	`jax.nn.tanh`
Activation (Final)	`jax.nn.sigmoid`	`jax.nn.leaky_relu`	None
Loss Function	Softmax Cross Entropy (`optax`)	Softmax Cross Entropy (`optax`)	Mean Squared Error (`optax`)
Input Shape Mapping	1D Vector (`28*28, 1`)	2D/3D Tensor mapped into field	Sequential 1D Inputs (X_1, X_2, C) mapped via offsets

Architectural design choices for the XOR model is provided in the ablation appendix.

D Ablation Studies

From the feature comparison table, one can observe that VNNs and Neural CAs are the only fully differentiable systems capable of learning their own architecture. To show clarify what parts of the architecture are free variables and which are hyper-parameters, we present the ablations below on a XOR 2D VNN MLP model with the details below.

XOR VNN Ablation Model Details

- **Base Grid (Universe) Size**: $N \times N$, where baseline $N = 50$.
- **Depth**: $layers = 5$, resulting in an internal logical depth of 11 ($2 \times layers + 1$) representing alternating decision and signal layers, plus an output layer.
- **Activations**: Internal states utilize `jax.nn.tanh` for non-linear activations.
- **Layout**: Inputs and outputs are spatially separated and offset from the center of the grid.
- **Samples**: $32,000$ total generated samples.
- **Inputs**: The inputs are unpacked into 8-bit arrays and continuous noise ($\delta = 0.2$).
- **Outputs**: Raw binary XOR results.
- **Data Split**: Standard 80/20 train-test split.
- **Epochs**: 100
- **Batch Size**: 128
- **Optimizer**: `optax.adamw`
- **Learning Rate Schedule**: Warmup cosine decay
  - Initial LR: 0.001
  - Peak LR: 0.03
  - Warmup Steps: 2.5% of total steps
  - End LR: 0.0005
- **Loss Function**: Mean Squared Error (MSE)

Ablation: Directions

Value	Accuracy	Total Time (s)	Time per Epoch (s)	Params Count	Model Size (MB)
2	0.9995	104.99	1.0499	7602	0.0385
4	0.9997	100.06	1.0006	7704	0.0389
Baseline (8)	1.0000	141.61	1.4161	7908	0.0397
12	0.9844	140.87	1.4087	8112	0.0405
16	0.9998	153.96	1.5396	8316	0.0413

Ablation: Grid Size

Value	Accuracy	Total Time (s)	Time per Epoch (s)	Params Count	Model Size (MB)
40	0.9998	113.88	1.1388	5208	0.0260
Baseline (50)	1.0000	141.61	1.4161	7908	0.0397
60	0.9998	157.90	1.5790	11208	0.0565
70	0.9998	189.32	1.8932	15108	0.0763
80	1.0000	205.79	2.0579	19608	0.0992

Ablation: Kernel Size

Value	Accuracy	Total Time (s)	Time per Epoch (s)	Params Count	Model Size (MB)
(5, 3)	0.0600	93.29	0.9329	7620	0.0386
(9, 3)	0.9995	113.23	1.1323	7716	0.0390
Baseline (17,3)	1.0000	141.61	1.4161	7908	0.0397
(13, 3)	1.0000	131.87	1.3187	7812	0.0393
(21, 3)	0.9998	185.26	1.8526	10596	0.0533

E APPLICATIONS

In this work, we show two possible applications of VNNs to demonstrate the types of problems suitable to solve that include repairable networks and emulating the GoL.

E.1 REPAIRABLE NETWORKS

We believe the VNN framework is uniquely well-suited for hardware-native implementations of models that can adapt their architectures by embedding themselves directly in the topology itself. Our experiments on repairable networks below demonstrate that a VNN architecture can be repaired when damaged in applications, such as corrupted computational elements of a CPU in damaged spacecraft or an autonomous vehicle, through simple fine-tuning despite losing a small percentage of its neurons.

Data Pipeline The model is trained on a synthesized bitwise arithmetic and logic dataset on the ALU task. Each sample consists of two 8-bit input operands (X_1, X_2) and a control sequence (C).

Simulating Hardware Damage To simulate physical hardware failure or radiation-induced bit-flips, we introduce ablations to a fully pre-trained VNN acting as an ALU. For a specified damage level (e.g., 1% to 5% of the total grid), we randomly select a subset of the VNN's internal cells. The Codd states (s) of these cells are forcefully overwritten to $-\infty$ (similarly to masked attention in transformers), while their corresponding weights (w) and biases (b) are zeroed out. We apply a protective mask over the input, control, and output interface cells to simulate damage to the internal routing and mirror a scenario where hardware is partially corrupted but inputs remain intact. In practice, inputs and outputs can be moved too, although we did not conduct these experiments due to time constraints of the review process.

Repair and Training Procedure Following the ablation, the VNN typically experiences a degradation in arithmetic accuracy. We then initiate a repair phase:

1. **Fine-Tuning:** The damaged VNN is fine-tuned on a new training set for a rapid recovery period of 50 epochs.
2. **Evaluation:** The network is evaluated on the test set immediately after damage, and once again after the 50-epoch fine-tuning period to measure the recovery of accuracy.

Results An example of the type of damage inflicted on the cellular system is shown in Figure 5. The VNNs were able to be repaired in all instances and the accuracy degradation was proportional to the percentage of damaged cells. The ablations were run across fixed number of epochs during the fine tuning and resulted in the recovery of accuracy, see table below. The repair levels obtained seem to imply that small amounts of damage can be fully restored through fixed fine tuning intervals.

Table: The accuracies of the VNN MLP on the arithmetic task before and after 50-epoch repair at varying damage levels.

Damage Level	Accuracy Before Repair	Accuracy After Repair	Repair Time (s)	Params Count	Model Size (MB)
0.00 (Baseline)	0.9987	0.9987	0.00	16316	0.0769
0.01	0.6812	0.9984	475.48	16316	0.0769
0.02	0.8619	0.9987	476.35	16316	0.0769
0.03	0.3973	0.9711	472.98	16316	0.0769
0.04	0.1775	0.9981	476.69	16316	0.0769
0.05	0.7674	0.9983	473.40	16316	0.0769

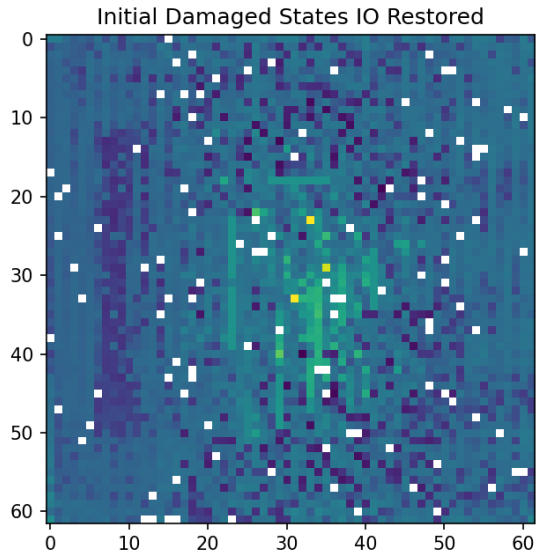


Figure 5: The simulated damage used in experiments showing 3% of cells whose states designate dead neurons, but with input and output (IO) cells maintained with usual functionality.

E.2 GoL Simulation

To demonstrate the capability of the VNNs in strictly learning an approximation of the discrete transition rules of Game of Life (GoL) (see Appendix I for a brief introduction), we constructed a dedicated 64x64 grid experiment.

Data Pipeline To approximate the 3x3 neighborhood transition rules of GoL, a dynamic dataset is generated prior to training following similar findings of Gilpin (2019):

1. **Patterns:** For each sample trajectory, the script randomly selects between five classical GoL patterns: `Blinker` (3x2), `Glider` (3x3), `Toad` (4x4), `Pulsar` (13x13), and `GliderGun` (10x37).
2. **Initial Placement:** The selected pattern is inserted into a blank 64x64 grid with randomized initial coordinates, buffered by their evolution steps (`l-steps = 12`) on all boundaries to prevent cropping.
3. **Evolution:** The grid is evolved using the ground-truth GoL simulator.
4. **Dataset Size:** This process generated 51,200 strictly controlled transition pairs with batch size of 128.

Model Architecture A 64x64 grid with kernel size at 3x3 to mimic the GoL rules with a depth of 2 having 13440 parameters. A total of 128 independent Green's functions (directions) are learned during training. The `sigmoid` activation is used for signal propagation, and no activations for the final step. The VNN was trained via AdamW with a warmup cosine decay schedule, using an Mean Squared Error (MSE) loss. The training script and the framework will be released open-source with these reviews.

Results The model converged by epoch 40 of training, reaching and maintaining 1.0 (100%) Accuracy on predicting the precise subsequent GoL generation. We show (uneditted) images of the learned 3x3 Green's function routing parameters discovered by the network that seem to encode the GoL rules in Figure 11. Some of the 3x3 filters of Figure 11 show a form of the GoL rules by counting neighbours, e.g. a particular pattern of 3 alive neighbours. Others seem to be combinations

of other neighbor operations that require further study. This leads us to make the connection to universality based on Proposition 2.

Proposition 2 (Rules-based Universality)**.** *The cellular machine (CM) is equivalent to a GoL system, able to construct a TM and is therefore capable of universal computation.*

Proof. Consider that each of the Von Neumann neurons are simple finite state automata because of its two-state Codd state s, i.e. the neighbourhood is its input and the output to change in its current state. The neuron can then change its internal state s if the neurons are linked together, i.e. if there is some way to link the inputs and changes in states of each neuron. In the GoL, this is achieved via the glider construct that is able to propagate throughout the space and the Gosper glider gun is able to produce an arbitrary stream of such gliders acting as a signal generator. Johnny is able to achieve this through explicitly constructing a signal generator construct and Wolfram is able to do this in Rule 110 because of propoagating signals with his 1D CA. The VNN has propagation built into it by design through the use of Green's functions, which also known as propagators in physics. The Green's functions provides the link between neurons, causing them to change their states in accordance to the weights/biases of the network that is optimized through stochastic gradient descent.

We begin by noting that Johnny's original formulation of a cellular system was for self-replication that was implemented as a TM with a program for replication and is therefore Turing complete (von Neumann, 1966). We showed that (1) a VNN is possible that learns a (continuous) two-state system as a contrastive learning process between the state 0 and 1 (see Figure 2b) similar to the GoL, (2) it is possible to learn all of the rules of GoL as a neighborhood function, since it only involves the 8-connected neighborhood and (3) it is then possible to build the UTM directly on the cells using these two-state rules as shown similarly for the GoL (Rendell, 2013) (see Appendix I). Evidence for learning the GoL rules are provided in Appendix E.2 showing that the Green's functions can approximate them. ◂

The exact mapping of the 3x3 filters to the GoL rules and compression of these rules with respect to the number of Green's functions remains an avenue of future work.

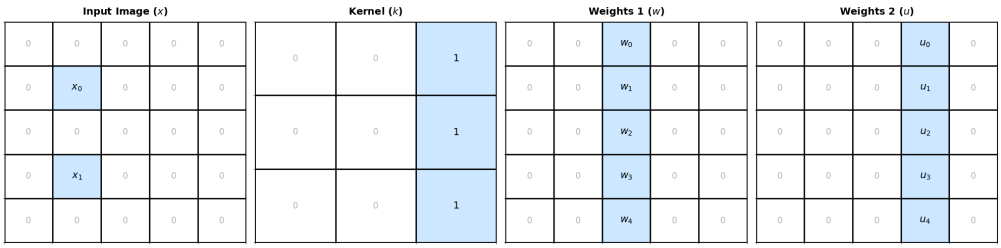
F Theory Correspondence

The correspondence principle in physics refers to when a new theory approaches the established theory at an expected limit. In this section, we discuss the correspondence of the VNNs with respect to deep learning theory. However, before looking at the correspondence in detail, we begin with a simple example of a two neuron VNN in a 5×5 universe of cells forward propagated by a 3×3 forward operator or Green's function. Figure 6 shows the forward pass of the VNN as an algebraic representation to derive the terms that result at each location in the universe during each step of the two convolution and weighting computations. In this case, the output neuron can be extracted at the coordinate $(2, 3)$ in Figure 6a to perform simple tasks such as binary classification etc. It is important to note that not only are there terms from what we would expect from deep learning, but also the existence of second order terms that appear to enhance the VNNs ability, particularly when using larger multi-dimensional kernels. The interaction of these second, third etc. order terms is determined by the size of the kernel(s) and the arrangement of the inputs within the universe.

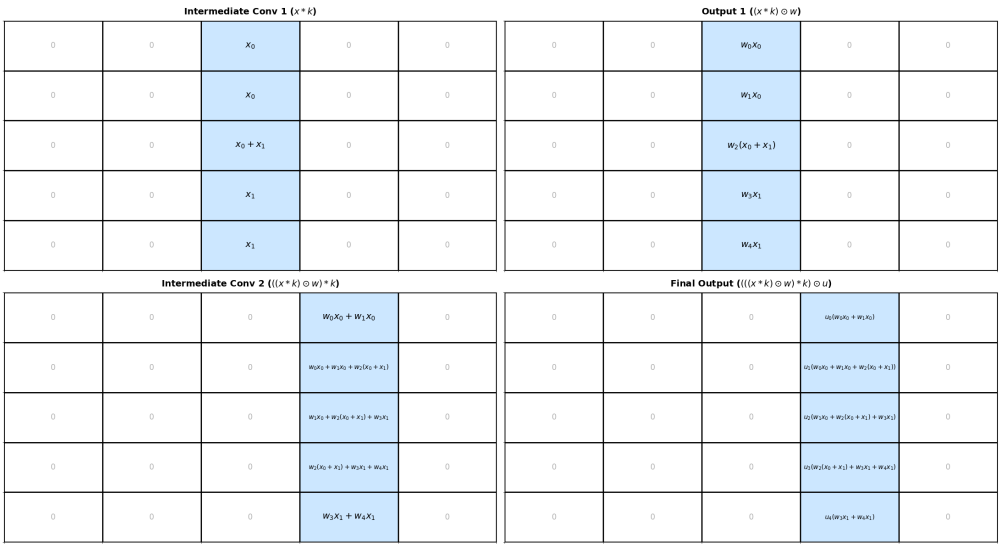
To explore this correspondence in detail, let us assume that the forward operator is given by (3) and the back propagation is conducted according to the backward operator given by (1). If we set the Green's function is to be fixed like it is for deep learning because the neurons are given as Figure 2a without Codd states s, then the linear differential operator $\mathcal{D}$ is also fixed and is given by the Laplacian ∇^2, so that the backward operator becomes the heat equation

$$\nabla^2 C(c) = f(c). \tag{6}$$

The Green's function to the above equation is a Gaussian function (Arfken and Weber, 2001) that is commonly approximated by a smoothing kernel in practice and its propagation leads to a diffusion process, the most crude of which is the averaging filter consisting of all ones. This is what Johnny must have referred to when he predicted that such a system must be of diffusion type in the 1950s. Thus, the standard MLP from deep learning literature can be implemented in VNNs as a system with Green's functions with all ones and hand-crafted Codd states s (that are fixed and represent decision



(a) The initial arrays



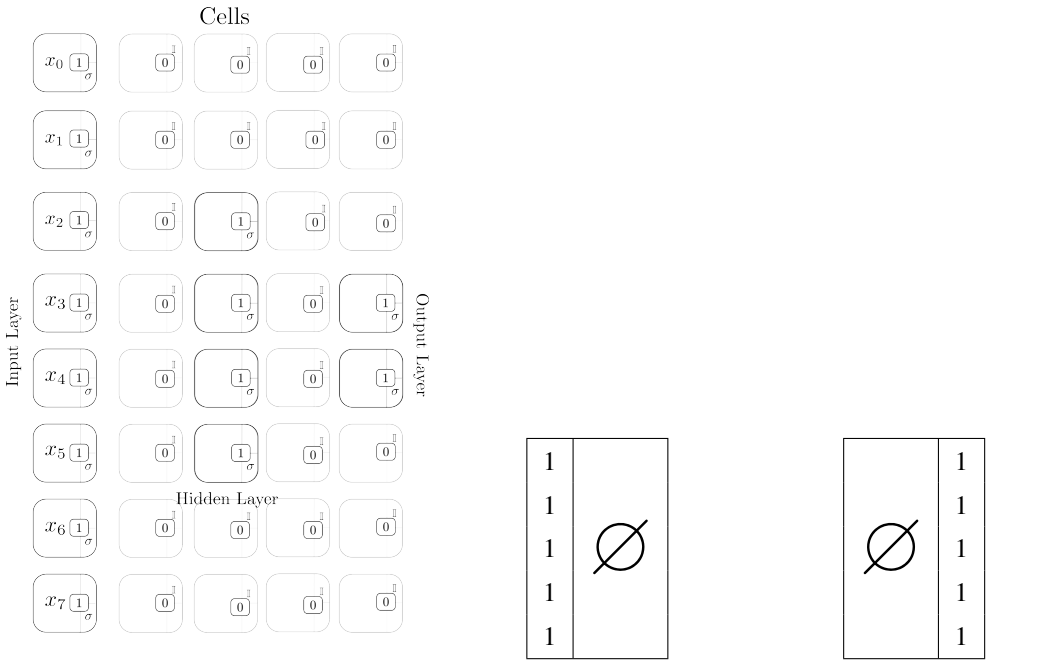
(b) Intermediate and final results

Figure 6: An example of a simple two neuron VNN forward pass algebraically. Here we have broken up the weights into separate arrays for illustrative purposes. The biases can be done similarly except through addition rather than multiplication and activations can be applied per layer as needed.

and signal neurons) on the Chua field C as fixed parameters with the appropriate hidden layers etc. to fit the desired architectural approach (see section 3.2).

Proposition 3 (Correspondence-based Universality)**.** *The Neurocell framework is equivalent to deep learning as a special case, where the Green's function and Codd states are fixed, and that this case is equivalently Turing complete.*

Proof. We can view also this from the traditional deep learning formulation point of view using the matrix-vector form of an MLP. Consider the VNN MLP with the depth along the row dimension (left to right) within the Chua field C that consists inputs as a vector of length 8, 1 hidden layer of length 4 and 1 output layer of length 2 (see Figure 7). Thus, we have a hyperplane $d = 2$ form of VNNs, where each of the layers take the form of vectors of dimension $d - 1$. We could place the input vector in 8 adjacent cells along one column as a column vector with all these 8 cells having decision Codd states $s = 1$. Then a column for signal neurons to hold the connections between inputs as the connection weights with $s = 0$ in-between the input layer and the subsequent hidden layer of 4 cells of decision Codd states $s = 1$. Likewise, this is followed by a column for signal neurons creating another signal layer and then the output layer cell. The forward pass is then given by the element-wise multiplication of the corresponding column from the tensor of weights $\boldsymbol{W}$ with the resulting convolution of the Green's function consisting of all ones with the input and the signal layers (see Figure 7b and 7c). Because of this correspondence with deep learning, whose most general architectures such as the transformer are known to be Turing complete (Pérez et al., 2018), the Neurocell framework is also Turing complete because the multiplications and convolutions can be represented equivalently as a matrix that results in a toeplitz matrix (which is a circulant matrix



(a) Deep learning like VNN MLP (b) Backward pass operator (c) Forward pass operator

Figure 7: (a) represents the typical deep learning MLP equivalent as a VNN, where inputs are given by x_i, the Codd states s are hard-coded to ones for hidden layers, then activated through the non-linear function σ when firing, and zeros for signals. (b) shows the forward pass Green's function that approximates the diffusion process. (c) shows the backward pass Green's function that approximates the diffusion process. Both (a) and (b) can be used to approximate the true forward pass and backpropagation numerically with convolutions only, the latter would be weighted by the learning rate. Note that (a) and (b) are flipped horizontally in deep learning frameworks, where convolutions are commonly implemented as correlations rather than strict convolutions for efficiency.

if using circular convolutions and block toeplitz matrices for minibatches), and thus corresponds to the usual deep learning formulation using matrix-vector forms. In other words, the static hyperplane VNNs are a special case of traditional deep learning MLPs, where the matrix multiplications take the form of toeplitz matrices. ◂

G Data-driven Green's Functions

In this section, we show the various Green's functions learned from the models covered in this work. Figures 8 and 9 shows the 16 Green's functions found for the ALU models presented in Figure 3 (middle and right). Figure 10 shows the resulting 3 Green's functions from the CLP model for simple classification of the Wine dataset (Aeberhard et al., 1994) provided within Scikit-Learn (Pedregosa et al., 2011) from Figure 3 (left). The functions show a similar symmetry to the original half-circular shape of the input. Figure 11 shows the learnt Green's functions of the GoL (see Appendix I for a brief introduction) experiments in simulating the GoL with a VNN (see Appendix E.2 for details). The learnt 3x3 neighborhoods seem to encapsulate the GoL rules despite being compact in number (i.e. less than the number required by 2^9 possible combinations expected in the rules).

H Computation

Imagine that you could reduce all known computations, anything you know a computer can do and even arithmetic, down to just a few fundamental operations of reading, writing and moving symbols around on pieces of paper and nothing more. It would be simple enough to even solve any computation by hand given enough patience. With such a simple machine, we could not only

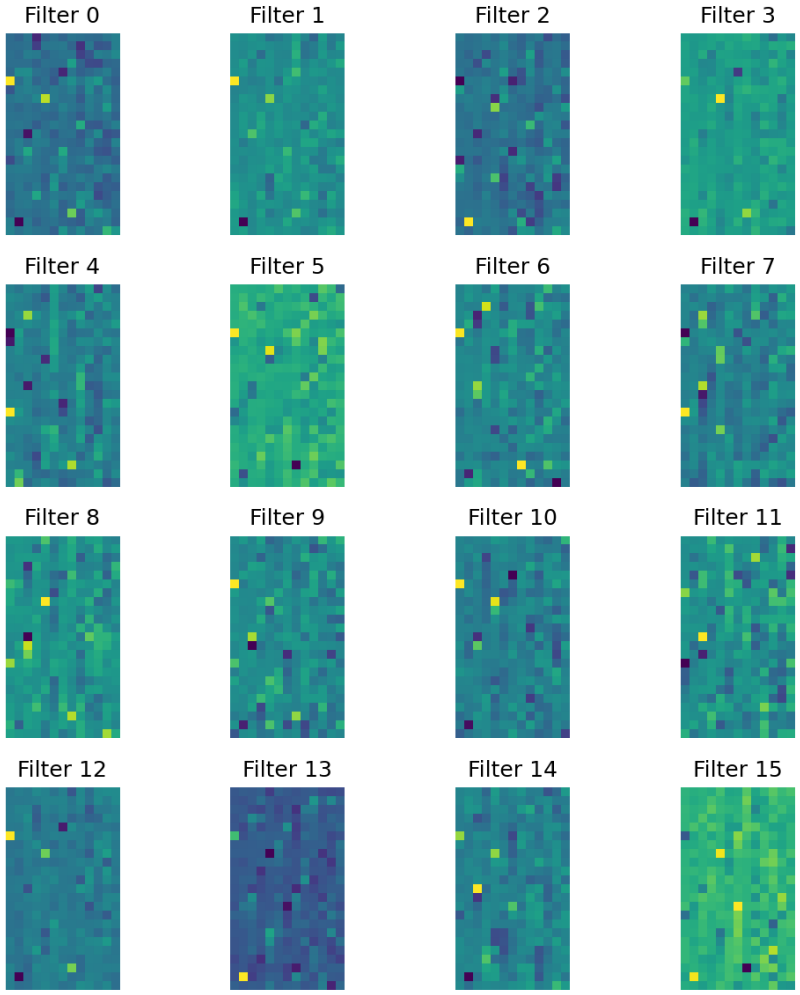


Figure 8: The resulting Green's functions learned for the box ALU model from Figure 3 (middle).

study computation and prove its properties, but also help design a physical realisation of such a computation machine. Alan Turing (Turing, 1937) introduced just such a machine in order to study the "Entscheidungsproblem" (or the decision problem) and in the process developed a theory of computation. These machines are known as Turing machines (TMs) in his honour and eventually led to the design the first digital computers by John von Neumann and others.

In the remaining sections of this chapter, we will construct these machines and design programs for them. In order to do this, we first investigate simpler machines that are possible, which do not compute everything, but are still useful for very simple specialised devices. These machines are devices that do not have any memory.

H.1 Finite State Machines

Consider having to design a device or system that only needs to decide whether to accept an input or reject it. Equivalently, you would like a system that produces a yes/on or no/off decision. In such a scenario, a machine that has a finite number of states and one that processes the input one element at a time is appropriate. This type of device that has no memory (in the sense of a tape or array) and is called a finite state machine (FSM). First, we need to define the symbols that we will use for the

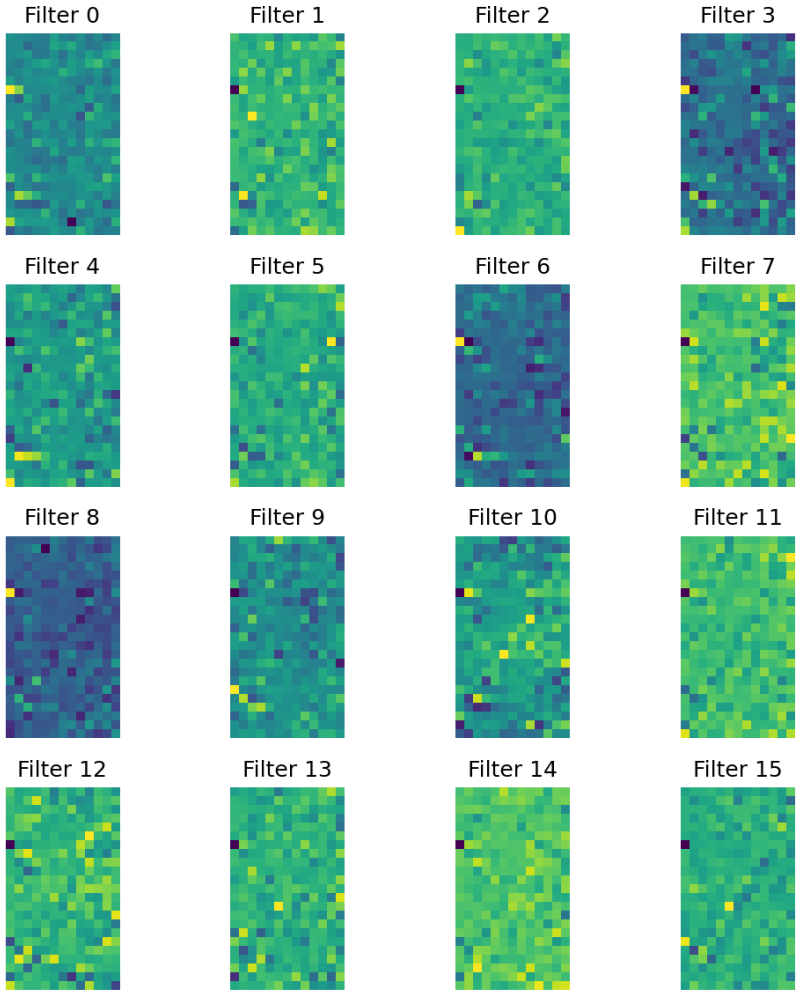


Figure 9: The resulting Green's functions learned for the tape-like ALU model from Figure 3 (right).

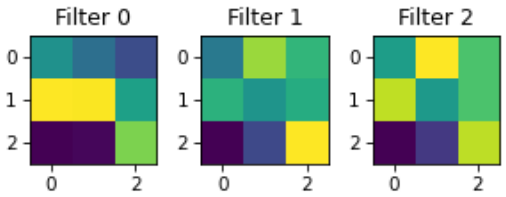


Figure 10: The resulting Green's functions learned for the CLP model from Figure 3 (left).

input and the computation to apply on. The set of all possible unique symbols or characters is called the alphabet $\mathcal{A}$. The input is then a string of symbols concatenated together from the alphabet.

Formally, each FSM firstly consists of pre-defined set of states $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ for number of states n, which is used during the computation. This includes three important states that all FSMs have: an initial state s_0, an accept state s_{yes} and a reject state s_{no}. Secondly, we need the set of transitions or instructions δ for the FSM that defines what state the machine should go into when

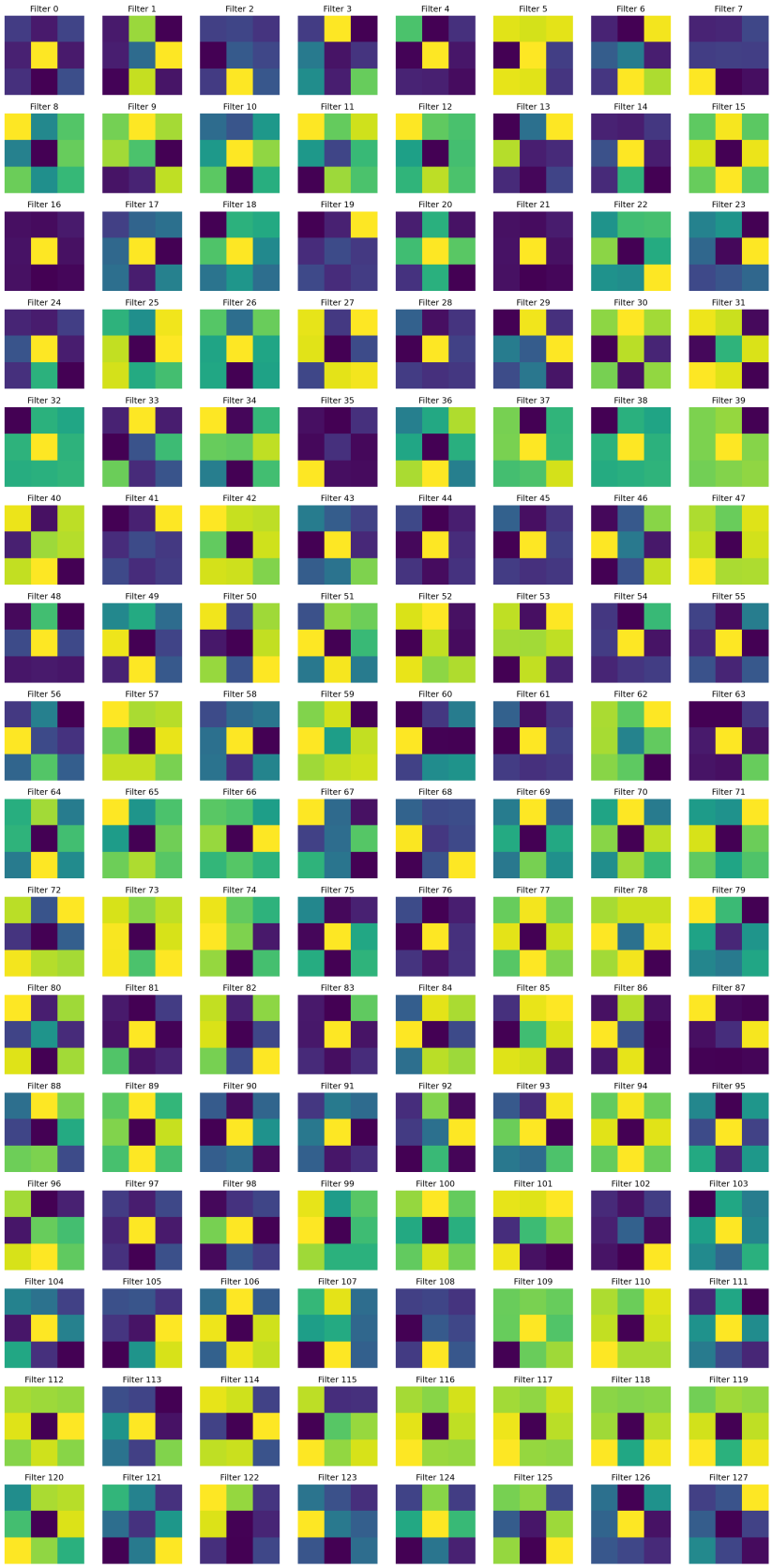


Figure 11: The resulting 128 data-driven Green's functions learnt for the GoL VNN simulation.

in a particular state and it reads a certain character from the input string. In notation, we mean that $\delta : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, i.e. state paired with character to another state.

H.2 TURING MACHINES

We saw that FSMs have an alphabet $\mathcal{A}$, associated states $\mathcal{S}$ and transition rules $\delta : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. It does not however, have memory for working as a human would need to do arbitrary computations. This would also require keeping track of the "awareness" within that memory. Thus, we define a more powerful machine that has a tape for memory and a read/write head for keeping track of awareness, which is known as a TM. The tape is a one dimensional (1D) array that can be thought of as sequential access memory. The read/write head moves left or right along the array and points to a cell in the tape that corresponds to the current computation taking place. The tape may hold the input and output strings, any intermediate results or working and even the program as we shall see later in this section. Figure 12 shows an illustration of the tape and head.

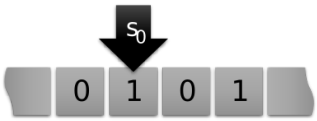


Figure 12: An illustration of a Turing machine with its tape and read/write head. In theory, the tape extends to infinity on both sides.

For theoretical reasons, we may consider the tape to be infinite in length. Unlike FSMs, the head also permits us to back along the input and mark on the tape. In fact, we are permitted to write any symbol we like as long as we define a tape alphabet $\mathcal{A}_T$ that is a superset of symbols permitted on the tape with the input alphabet $\mathcal{A}_i$.

With memory available for working and ability to read/write and move the head, we have more degrees of freedom with our transitions. Indeed, now our transitions have the form $\delta : \mathcal{S} \times \mathcal{A}_T \to \mathcal{S} \times \mathcal{A}_T \times \{L, R\}$, where L and R represent the left and right movement of the head. Finally, the Turing machine also has an initial state s_0 like an FSM, but only has single accept and reject states, s_a and s_r respectively, because we have the ability to complete the computation instantly without having to process the entire input string since we are free to move back and forth along the input. Thus, we can formally define the concept of a Turing machine.

Definition 2 (Turing Machine)**.** *A Turing machine is a seven tuple $T = (\mathcal{S}, \mathcal{A}_i, \mathcal{A}_T, \delta, s_0, s_a, s_r)$, where the sets $\mathcal{S}, \mathcal{A}_i$ and $\mathcal{A}_T$ are all finite and*

1. *$\mathcal{S}$ is the set of all possible states,*
2. *$\mathcal{A}_i$ is the input alphabet without the blank symbol,*
3. *$\mathcal{A}_T$ is the tape alphabet which includes the blank symbol,*
4. *$\delta : \mathcal{S} \times \mathcal{A}_T \to \mathcal{S} \times \mathcal{A}_T \times \{L, R\}$ is the transition function,*
5. *$s_0 \in \mathcal{S}$ is the initial state,*
6. *$s_a \in \mathcal{S}$ is the accept state,*
7. *$s_r \in \mathcal{S}$ is the reject state with $s_r \neq s_a$.*

The Turing machine would function as the following:

1. We assume that the tape is infinite in both directions.
2. Begin with the input placed in the middle of the tape with the head in the left most cell of the input. The rest of the tape is considered empty or blank.
3. The input alphabet $\mathcal{A}_i$ does not have the blank symbol, so it is safe to assume that the end of the input is marked with a blank symbol.

4. The computation begins by using the tape alphabet $\mathcal{A}_T$ and following the transition function until either the accept or reject state is reached.
5. When either of s_a or s_r is reached, the machine halts. Otherwise, the machine continues forever without halting.

A detailed exposition of computation theory, including descriptions of both classical and quantum computing can be found in Moore and Mertens (2011).

H.3 Equivalences

Firstly, if we add higher dimensional tape, such as two dimensional (2D) or three dimensional (3D) tape, to increase the versatility of our memory. It is easy to show that 2D or 3D arrays are equivalent to 1D arrays. and in fact, computers have been using 1D arrays to represent multi-dimensional arrays very early on in the history of computers. This is because we can always use memory addressing tricks to simulate multi-dimensional arrays using 1D coordinates.

Secondly, we could use multiple heads to improve the speed and support more instructions across the tape and transition function. However, this simply means that we can process instructions faster, but no more improvement of the computational power of the machine. In a similar argument to simulating multi-dimensional arrays for memory with a 1D array, we can do the same with a single read/write head. The single head can complete each of the operations required for other heads one after the other sequential until all the operations of all heads are complete.

H.4 Universality

In general, there are three ways one can show completeness: construct a universal program/machine that can simulate any program; show that it can perform all basic operations that can lead to such a program; construct an actual Turing machine (or an equivalent Turing complete system) within the system itself. Alan Turing constructed the original UTM as the first type (Turing, 1937) (see Appendix H for a basic introduction), Johnny introduced the Von Neumann (hardware) architecture as second type (von Neumann, 1993) and the GoL proof (Rendell, 2013) is example of the third type.

I Game of Life

The Game of Life (GoL) was introduced by John Conway (Gardner, 1970) as the culmination of cellular automata research in which the rules have been reduced to a few simple rules. The game involves a set of cells within an $N \times N$ grid whose state is either alive or dead (i.e. 1 or 0 respectively). The grid effectively represents the 'universe' that will be simulated and the alive cells the life within it. The game is governed by a set of simple rules that dictate the next state of each cell in this universe depending on its current state and the states of its neighbours. The rules of GoL depend on the 8-connected neighbours of a cell as follows:

1. Underpopulation: A live cell that has ¡ 2 live neighbouring cells will die.
2. Survival: A live cell that has 2-3 live neighbouring cells will remain alive.
3. Overpopulation: A live cell with more than 3 live neighbours will die.
4. Reproduction: A dead cell with exactly 3 live neighbours will become alive.

The game begins with an initial state of the universe with a pattern of live cells. The universe is evolved by applying the above rules to each cell of the universe to determine the next iteration of the simulation. The evolution of the universe is observed by continual computing of the next iteration of the universe.

J Abbreviations

1D one dimensional 26
2D two dimensional 27
3D three dimensional 27

PDE partial differential equation . . . 2
PSF point spread function . . . 2
CeNN Cellular Neural Network . . . 2
CNN Convolutional Neural Network . . . 9
VNN Von Neumann Network . . . 2
FSM finite state machine . . . 23
TM Turing machine . . . 3
UTM universal Turing machine . . . 9
CM cellular machine . . . 9
ALU arithmetic logic unit . . . 7
EoC Everything on Chip . . . 8
SoC System on Chip . . . 8
CA cellular automata . . . 2
GoL Game of Life . . . 2
MLP multi-layered perceptron . . . 3
CLP circular layered perceptron . . . 7
NAS neural architecture search . . . 3