Title

Neil Babson
PhD student at Portland State University working at Teuscher Lab
new to C-BRIC

Research Direction

At Teuscher lab we are interested in exploring emerging paradigms for computation with the potential for faster, cheaper, and more energy efficient information processing.
Traditional CMOS technology faces a number of challenges in the near future that underline the need for new computing paradigms:
1. Heat – the density with which transistors can be packed on a chip is limited by the challenge of heat dissipation
2. Starvation – the growth in power consumption used for computation is unsustainable. According to SRC data computing will require more power than the world can produce by the year 2040 if current trends continue
3. Size – due to thermal noise and quantum effects it will soon be impossible to continue to shrink resistors
4. Parallelism – While industry has turned to increasing the number of parallel cores in order to keep a form of Moore's law alive, not all problems can be easily parallelized
5. Cost – According to Rock's law the cost of chip fabrication plants doubles every four years. At some point the cost for the next generation of chips may exceed their value

Ongoing Research

In order to confront these challenges we need to develop simple, efficient, low cost hardware that can solve complex problems. These can be based on simple cellular systems such as Random Boolean Networks and Cellular Automata as well as take cues from

biological systems to produce brain-inspired, or neuromrphic, systems.
Reservoir computing, using a variety of physical reservoir substrates, holds great promise for the development new of cheap and efficient hardware. One of the focuses for research at Teuscher lab is on how hierarchical and self-organizing reservoirs can improve the performance of Reservoir Computing.
Simple cellular systems and those with unstructured components have the least fabrication constraints, reducing production costs.

Simple Dynamical Systems

Reservoir hardware and software implementations can also utilize the information processing capabilities to develop faster and more energy efficient computing devices that are adaptable for the many small devices that constitute the growing Internet of Things.

Talk Outline

The remainder of this talk is in two parts. First I want to give some background on the simple cellular systems Random Boolean Networks and Cellular Automata and on the Reservoir Computing computational framework.
After that I will introduce Reservoir Computing with Cellular Automata and my own work experimenting with complex CA rules.

Random Boolean Network

Random Boolean Networks have a biological inspiration. They were first developed by Kaufman as a way to model genetic regulatory networks.
An RBN consists of a directed graph of randomly connected nodes each of which has a Boolean function assigned to it which maps inputs at at one time step according to its truth table to a 0 or 1 output; which it forwards to connected nodes at the next time step.

The number of inputs to a nodes Boolean function is determined by the in-degree, K, of connections from other nodes. The example network on the slide has a uniform in- degree of 3, but that is not necessarily the case.
The stability of the RBN depends on the average in-degree connectivity of the nodes in relation to a critical connectivity threshold. If the outputs of the Boolean functions are all randomly distributed the critical threshold is 2.
If average connectivity is below the threshold the network converges to stable behavior. If it is above the threshold chaotic dynamics are observed. At the critical connectivity threshold the network exhibits interesting behavior with aspects of both stability and chaoticity.

Cellular Automata

Two state cellular automata are a special case of RBN in which all the nodes share the same function and connectivity and they are assembled in a lattice structure.
CA were invented by John von Neumann and Stanislaw Ulam in the late 1940 and early 50s.
Von Neumann used a 29 state CA to demonstrate the possibility of a Universal Constructor, a machine that carries a blueprint of itself and is capable of self-replication. Since then Cas have been used as a modeling framework for many complex systems.

Cellular Automata

A CA is composed of a grid of cells, usually in one, two, or three dimensions.
Each cell is in one of a finite number of states.
A single transition function is applied uniformly and simultaneously to all the cells, updating their state at time t+1 depending on the states of their nearest neighbors at time t.

1D Cas

Focusing now on one dimensional CA s, two parameters define a CA, the number of states, S, and the size of the neighborhood, K.
Here we see a two state CA in the form of a line of cells.
The neighborhood of each cell is of size three, its two contiguous neighbors and itself, since by convention a cell is considered its own neighbor.
The rule space, or the number of possible transition functions for a given value of K and S is equal to the number of states raised to the power of possible neighborhood configurations S^S^K, or 256 for S=2 and K=3.

Elementary CA

These 256 CA rules are known as the elementary CA s.
Each rule is a lookup table from possible neighborhoods at time t to a new state at time t+1. The rule can be represented by reading it off the table as a base S number. Here is the transition function known as rule 90.
In order to display the CA's behavior over time, successive time steps are generally concatenated downward to create a two dimensional space-time diagram.

Complexity Classes

Stephen Wolfram, who studied the elementary CA s extensively investigating their emergent properties, proposed a classification system for rules based on the complexity of their behavior.
1. Class I rules quickly evolve to a homogeneous state from most initial configurations. All initial information is lost
2. Class II rules retain some initial information and evolve to a stable or oscillating state.
3. Class III rules evolve in a chaotic manner. Stable structures can't exist because they are quickly destroyed by noise.

4. For Class IV rules are between the stable and chaotic regimes and local structures can persist and interact with each other in complex and interesting ways.

Edge of Chaos

Another person who studied the chaoticity of CA rules was Langton, whose lambda parameter represents the fraction of transitions in a rule to an arbitrarily chosen quiescent state. If lambda=1 all of the transitions are to the quiescent state leading to homogeneous class I behavior.

At a critical lambda threshold a phase transition occurs between order and chaos. It is believed that it is at this “edge of chaos” where rules exist with enough freedom and enough structure to support the necessary requirements for performing computation: the storage, transmission, and modification of information. The edge of chaos corresponds with Wolfram’s class IV behavior, as Langton illustrates with this diagram from his paper.

Turing Complete

Many 2D CA rule, such as Conway’s famous Game of Life, and at least one elementary rule, rule 110, are known to be Turing complete, or computationally universal. Here we see an image of a functioning Turing machine built in the Game of Life universe.

Reservoir Computing

Reservoir computing is an innovation is Recurrent Neural Network design in which only the weights to the output layer are trained. The reservoir, usually a randomly connected neural network, is a dynamical system which casts temporal inputs into a high dimensional space enhancing their separability and creating features for the output layer. The input layer is randomly connected to the nodes of the reservoir, and the output layer may be randomly or fully

connected to the reservoir nodes. Only needing to train the output weights provides a large computational saving over traditional neural networks. Reservoir Computing was independently proposed under the names Echo State Networks, and Liquid State Machines.

Echo State Networks

ESNs were proposed in 2001 as a way to avoid the difficulties encountered in training RNNs by gradient descent, such as vanishing or exploding gradients, slow convergence, and convergence to local minima. The reservoir consists of randomly but sparsely connected discrete time sigmoid function nodes. An ESN must possess the "echo state property", the ability to remember – or echo – inputs, but gradually forget them over time. The reservoir weights can be scaled so that the spectral radius of the weight matrix is less than one to ensure the echo state property by preventing the runaway amplification of inputs.

Liquid State Machines

In 2002 LSMs were introduced as an approach to modeling recurrent neural circuits in biological systems. They have essentially the same design as ESNs, but using a reservoir of spiking integrate and fire neurons. Similar to the echo state property, A LSM must possess the property of "fading memory". The transient internal states of the reservoir perform computation without stable attractors, because in the absence of external perturbation the system will return to a quiescent state. The authors compare the behavior of the reservoir to the ripples that gradually subside on a pool of water disturbed by dropping pebbles into it.

Physical Reservoirs

After RC proved capable of many computational tasks it was realized that many other dynamical system besides neural networks could act as reservoirs, including physical systems. This is just a small subset of physical reservoirs that have been proposed or built.

1. Cadmium Selenide quantum dot nodes communicate by fluorescing in response to changes in their environment.
2. A tub of water was used as a reservoir whose output layer was trained on ripple patterns to perform a speech recognition task.
3. Atomic switch networks of self-assembling nanowires exhibiting neuromorphic dynamics.
4. Networks of photronics nodes which perform optical signal processing.

Liquid Brain

This slide shows my own reimplementation of the water reservoir – a very literal LSM referred to as the liquid brain. The inputs are conveyed to the reservoir by motors moving chopsticks in the water. The state of the reservoir was captured by pictures of the ripple patterns. A Gaussian filter was used to reduce noise and the images were smoothed with a Sobel edge detection mask. The weights were trained by a readout layer of parallel perceptrons.

Reservoir Computing with RBN

Another substrate that can be used to implement a reservoir as a simple cellular substrate like a RBN. This work was done by a former Teuscher lab grad student who is now a PhD student at ASU working with Jae-Sun on C-BRIC. It compares the performance of a single random reservoir, described as monolithic, to a hierarchical reservoir in which a degree of topology is introduced into the reservoir. The hierarchical reservoir was able to outperform the monolithic one by up to 60% on benchmark tasks.

Reservoir Computing with CA

Yilmaz introduced reservoir computing using the nonlinear dynamics of CA rules in 2014. Operating a ReCA system potentially offers orders of magnitude reductions in computation cost compared to an ESN due to the absence of floating point multiplications. The local nature of CA interactions are suitable for further speedup using GPU programming.

Deep ReCA

Previous ReCA work includes a hierarchical system using two connected CA reservoirs. Predictions from the first reservoir are used as inputs to the second, which was able to perform error correction, improving benchmark performance compared to a single reservoir.

Non-uniform ReCA

A non-uniform ReCA system has two subreservoirs implementing different rules joined into a single reservoir able to pass information between the two rules. For certain combinations of elementary rules the non-uniform reservoir was able to outperform monolithic reservoirs of either of the rules acting alone. It is speculated that the rules may solve different aspects of the same problem, and it remains untested whether non-uniform reservoirs using more than two rules hold promise. There is no obvious way for selecting rules that will work well together beyond trial and error.

5 Bit Memory Task

All previous ReCA work, as well as my own, which I'll describe in the remainder of this talk, used the 5-bit memory task benchmark, which is common reservoir computing research, and known to be difficult for recurrent neural networks, including ESN. The task has four binary input signals, and three binary outputs. During the first

five timesteps of one run of the task input one is one of the 32 possible five digit binary numbers and input two is its inverse. These inputs are the message that the system is supposed to remember. This is followed by a distractor period, which in accordance with common practice was 200 time steps. All this time the expected output is the same. Finally a cue signal is given on the fourth input channel after which the system must repeat the message from 205 time steps ago. The task is repeated for each of the 32 possible inputs. In order to pass the task the system must correctly predict all 210 * 32 * 3 = 20,160 bits.

ReCA System Design

The CA reservoir is made up of R subreservoirs connected end to end and wrapping around at the edges to form a single circular CA. The size of the subreservoirs is known as the diffusion length and is larger than the inputs, which allows the inputs to diffuse into the larger reservoir. A random mapping is created between the input channels and each subreservoir that is fixed and does not change throughout computation. At each timestep, after the input is applied to the reservoir, overwriting cell's states according to the mapping, the CA rule is then applied for I iterations to create one timestep of the reservoir. The output layer has three nodes because the 5-bit task requires three outputs. The reservoir is fully connected to the output layer and linear regression is used to train the weights between each reservoir cell and output node. The rule is applied again and the next input selectively overwrites the CA to form the first row of the next reservoir.

Reservoir Size

The size of the reservoir is R, the number of subreservoirs, * the diffusion length * I the number of iterations that the rule is applied. Pythons scikit-learn linear regression model is used to train the

weights. By measuring the CPU time required to train and test a reservoir for different values of I and R we can observe that the time required grows linearly with the size of the reservoir.

Complex Rules

All previous ReCA research involved the 256 elementary rules.
In contrast to the elementary rules, 1D rules with more than two states and/or neighborhood larger than three are referred to as complex. It seems possible that the richer dynamics of some complex rules could make them more effective as reservoirs.

Complex Rule Space

The large rule space for 3 state and neighborhood 5 CA rules make an exhaustive search for reservoir rules impossible, but a stochastic search was able to find consistently find rules that passed the 5-bit benchmark on their first try.

Complex Rule Space

For 4 and 5 state CA s the space of possible rules is huge and a stochastic search no longer finds rules that perform well on the benchmark. This is because the vast majority of possible rules are Category III, much too chaotic to support stable structures and support computation.

Genetic Algorithm

In order to find 4 and five state rules for ReCA a genetic algorithm was used to first reduce the chaoticity of the contending rules. A population of rules space-time diagrams was from an initial random state, and their fitness was evaluated using a “smallest largest” rule.
< Read rule>

Using the smallest area ensures that all the states contribite to the dynamic behavior of the system, and prevents the tendency for the GA toward a uniform Class I behavior.

Results

The performance of the most successful six rules of each CA type were compared.