Convolutional Automata

Intro

TBD

Background

TBD

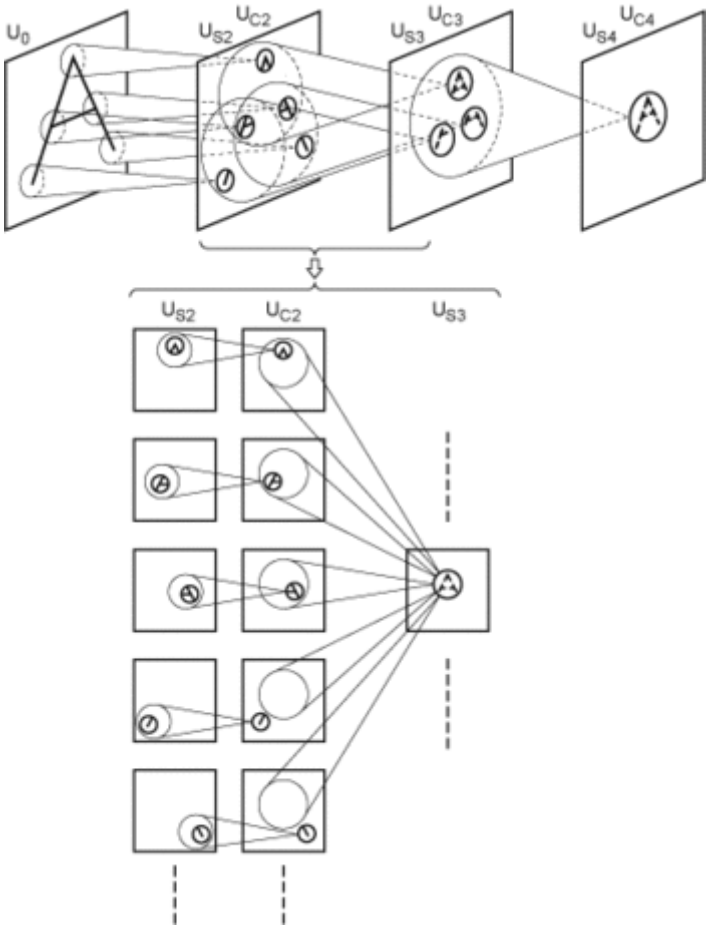
Premises

1. Gilpin has demonstrated that CAs can be represented as convolutional neural networks (CNNs) and their respective transition functions learned using a single 3x3 convlutional layer.
2. CAs can be evolved using specific aesthetic objective functions, as Heaton demonstrated with the *MergeLife* algorithm, and simple genotype representations can be used to encode all the hyper parameters of a paritcular CA.
3. According to Rafler Conway's Game of Life can be generalized into the continous domain and the notion of "neighborhood" further generalized into a specific class of radial functions.
4. Wolfram's Elementary Automata taxonomies, definitions, etc. describe the primitives of cellular automata.

Generalizing 1-D Automata Using Convolutions

Assuming all of the above premises, we can begin to generalize 1-D automata as a two part construction of a convolutional *kernel function* and *activation function*. Using these two classes of functions the *elementary automata* described by Wolfram can be expressed as *elemenatary convolutional automata*, which will be referred to as *ECA* or *ECAs* from this point forward. But first, let's describe what these classes of function are.

2-D Kernel Function for Images



Fukushima's model of pattern recognition using the neurocognitron

The *kernel* function of our generalized ECA can be thought of as the operation of analyzing the local neighborhood of a cell in a single operation.

The history of this technique comes from Fukushima's Neurocognitron and made its way into image processing. Using the kernel function technique that a single, computationally inexpensive operation can be applied over each pixel to create a transformation of the image. This includes transformations such as *blurring* or *edge detection*.

In the 2-D case, it has be generally expressed by Jaimie Ludwig (or 2) as:

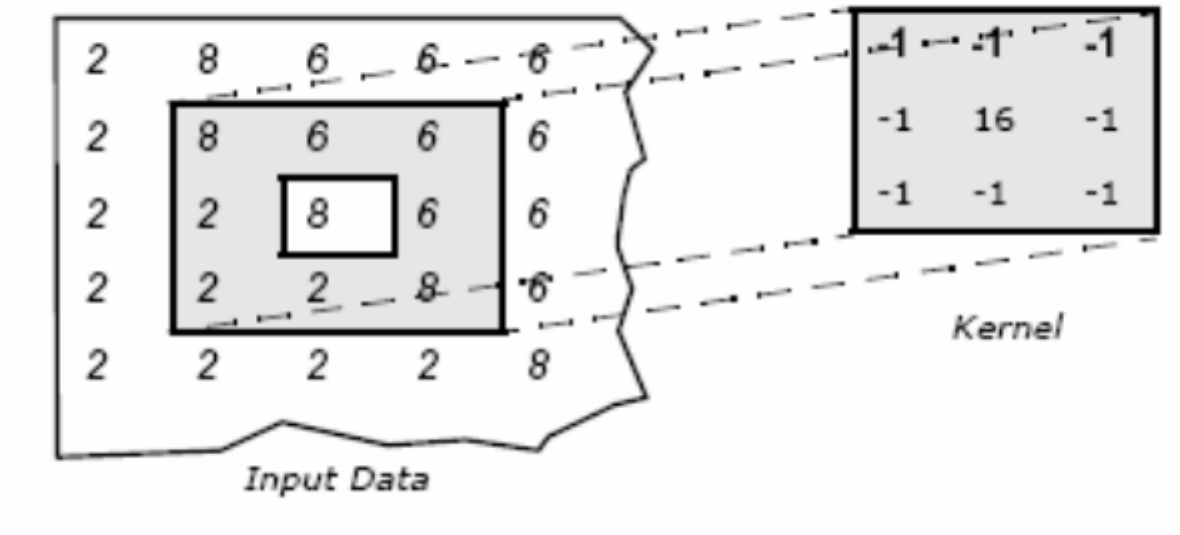
$$g(x,y) = \omega * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \omega(s,t) f(x-s, y-t)$$

Where $g(x,y)$ is is the filtered image $f(x,y)$ is the original image and every element is considered $-a \leq s \leq a$ and $-b \leq t \leq b$.

These take the form of matrix-like objects such as this one representing a 3x3 Gaussian blur effect on an image, where c is some constant coefficient:

$$\omega = c \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

This type of function could then be applied over an image as follows:



1-D Kernel Functions for Sequences

These image kernel functions can just as easily be expressed in 1-D as follows:

$$g(x) = \omega' * f(x) = \sum_{s=-a}^{a} \omega(s) f(x-s)$$

Using our Gaussian filter example, this creates the following ω'

$$\omega' = \sqrt{c} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

This could be applied over a list of pixel-like values or cells consisting of a single dimension just as easily as 2-D. For example, given the following convolution:

$$\phi = \begin{bmatrix} 2 & 1 & 2 \end{bmatrix}$$

And the following sequence s_0:

$$s_0 = \begin{bmatrix} 0 & 4 & 1 & 0 & 0 & 3 \end{bmatrix}$$

We can apply our convolution ϕ to generate s_1 as follows:

$$s_1 = \begin{bmatrix} 8 & 6 & 9 & 2 & 6 & 3 \end{bmatrix}$$

NOTE: this assumes 0 *at boundary conditions.*

This process can also be repeated over and over to produce a never ending series of transformations over the original sequence s_0, visualized here as matrix S

$$S = s_{0...n} = \begin{bmatrix} 0 & 4 & 1 & 0 & 0 & 3 \\ 8 & 6 & 9 & 2 & 6 & 3 \\ 20 & 40 & 25 & 32 & 16 & 15 \\ 100 & 130 & 169 & 114 & 110 & 47 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ s_{n,i} & s_{n,i+1} & s_{n,i+2} & s_{n,i+3} & s_{n,i+4} & s_{n,i+5} \end{bmatrix} = \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ \dots \\ s_n \end{bmatrix}$$

More generally, this can be expressed as the following recurrence relation where s_n is the a row in S:

$$s_n = \phi(s_{n-1})$$

However, given this is an additive convolution, you can see the side-effect of an ever-increasing values of each element in the sequence. To fully express the more dynamic behaviors of CAs, we need to add the *activation function*

Activation Function

For our purposes, an *activation function* is simply a secondary function that is applied element-wise over a single cell in our state space *independent* of neighborhood and *after* the *kernel* is applied. Or more generally, given a *kernel* function g and an activation function λ, the resulting state h can be expressed as:

$$h(x) = \lambda \circ g$$

or:

$$h(x) = \lambda(g(x)) = \lambda(\omega' * f(x)) = \lambda(\sum_{s=-a}^{a} \omega(s) f(x-s))$$

This has the effect of adding nonlinearity to the *kernel* function.

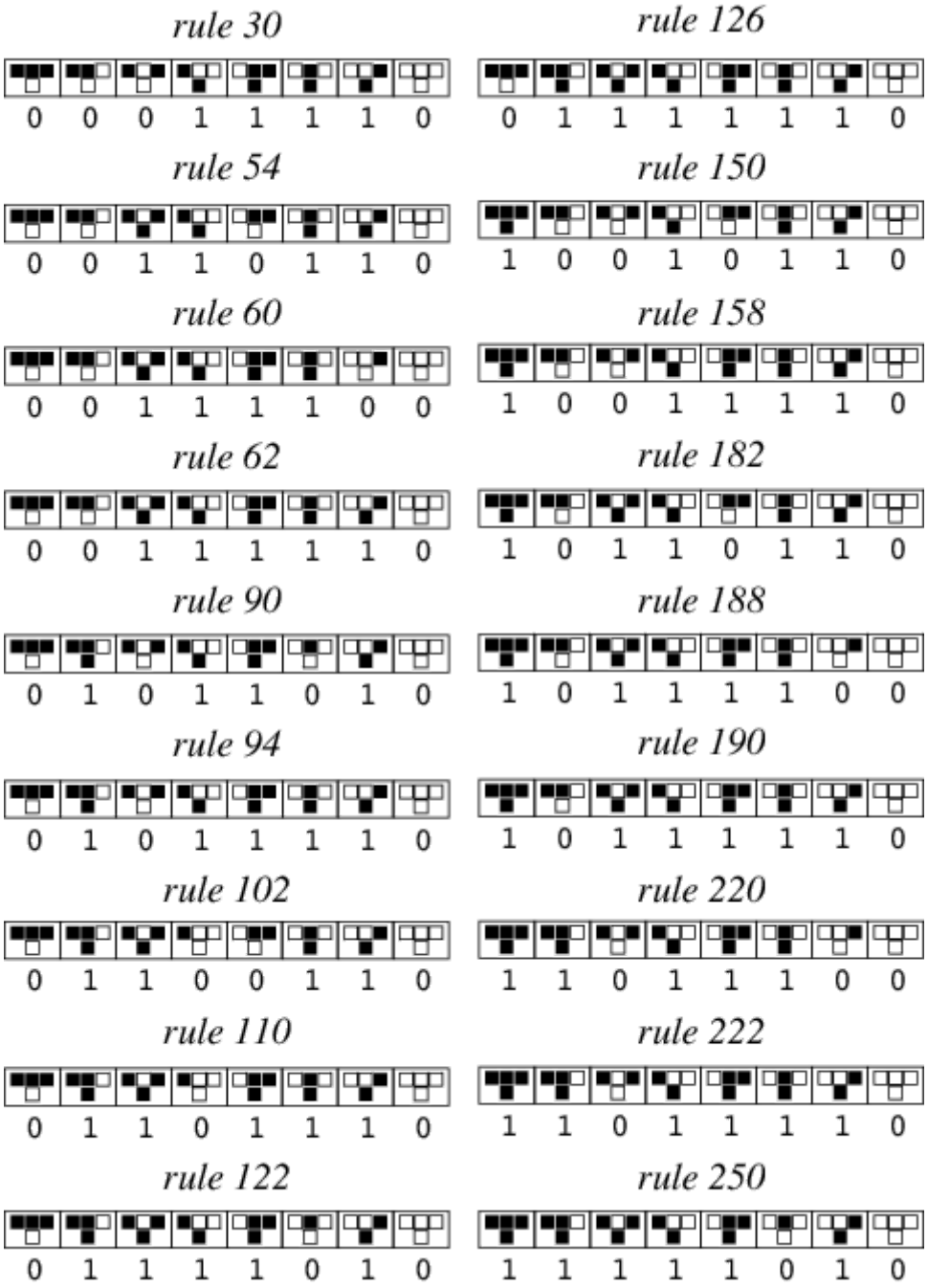
Why is nonlinearity necessary? Cellular automata transfer functions represent nonlinear transformations over the state-space and CAs model nonlinear systems. This makes nonlineary a crucial feature in the representation of ECAs and their state transitions. CA and ECA state transitions are distinct from signal processing transformations such as blurring or edge-detection, which a single *kernel* can sufficently express in linear terms.

Although Novak [1] [2] demonstrated that nonlinearity can be described using only convolutional *kernel functions* in signal processing, representing ECAs as a composition of *kernel* and *activation* provides more flexibility in *genotype* encoding when using using evolutionary computation to search the parameter space of ECAs.

Describing an Elementary Convolution Automata

Using this two-part architecture, we can begin to describe a fully-functional convolutional automata entirely by their *kernel* and *activation* components.

Wolfram describes the numbered, elementary cellular automata as being composed of sets of pairs of bit-arrays representing all 256 possible configurations:



These can be turned into mappings between a binary, 3-tuple of on/off states such as $[1, 0, 1]$ and their respective resulting state in the next iteration of 1 or 0.

The resulting *activation* and *kernel* for Rule 30 can be expressed with the 3-tuple being a set $P = \{a, b, c\}$, such that $a, b, c \in \mathbb{P}$ (i.e. they are all distinct from primes $\mathbb{P}$).

To build the *activation function* $f(x)$ we use a second set Q consisting of all possible products of a, b, c along with 0:

$$Q = \{a, b, c, ab, bc, ac, abc, 0\}$$

The function $f(x)$ then maps inputs x, which are the result of applying *kernel* ϕ to cell x, to a set R_n where $R_n \subseteq Q$, a subset of Q representing a particular elementary automata rule n.

In the case of encoding *Rule 30* to a single *activation* function we get:

$$\lambda = f(x) = \begin{cases} 1 & x \in R_{30} \\ 0 & x \notin R_{30} \end{cases}$$

Where:

$$R_{30} = \{a, bc, b, c\}$$

This can be written in trivial code using the following values for *kernel* ϕ:

$$\phi = \begin{bmatrix} 2 & 3 & 5 \end{bmatrix}$$

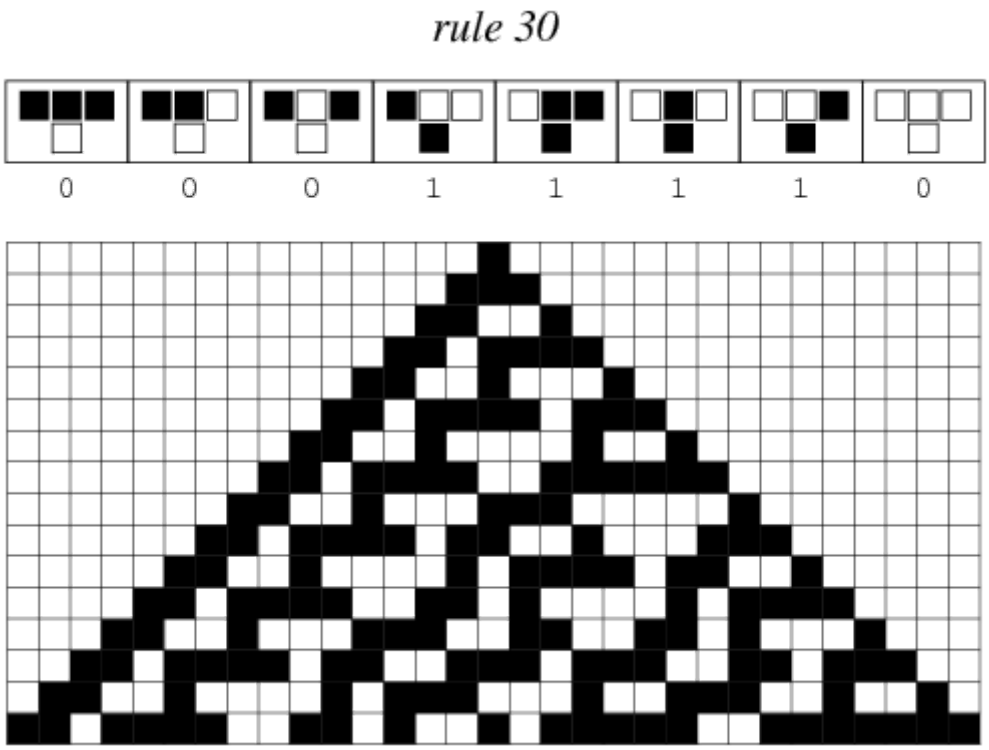
Where R_{30} is:

$$R_{30} = \{2, 15, 3, 5\}$$

Evaluating this for 10 timesteps, you can see that Wolfram's *Rule 30* thus emerges

$$S = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \\ s_9 \\ s_{10} \end{bmatrix}$$

From *Wolfram World*:



Evolving Compound Rules

By encoding the parameters and coefficients of the *activation function* and the *kernel function* as *genotypes*, the entire *rule space* can be explored using .... TBD

Objective Function

TBD (Describe objective function)

Initial States

TBD (Describe the initial states to be tested)

Pattern-Producing Convolutional Automata

Aesthetic Objective Functions

TBD (music evaluation?)

Phenotypes

TBD (piano from sequences)

BibTex

TBD