Convolutions

A convolution is a special type of linear map.

We are going to consider so-called list convolutions which are linear maps between lists of numbers, which can be considered as vector spaces.

Here is a 1D version where the elements are vectors.

$u_j = \sum \text{kernel}_i \, v_{j-i+\text{i0}}$

In 2D, the components are matrices.

$u_{ij} = \sum \text{kernel}_{kl} \, v_{i-k+\text{k0}\; j-l+\text{l0}}$

Example in 1D:

Back to the 1D case, let's say are data is the vector of length 5:
1 3 2 5 2

Let's say the kernel is the length 3 vector:
-2 3 -1

Then the convolution is
-1*1 + 3*3 + -2*2
-1*3 + 3*2 + -2*5
-1*2 + 3*5 + -2*2

which is the length 3 vector: 4 -5 9

The kernel is reversed and slides across the dataset.

Example in 1D:

Rewriting the above example in a grid, where the rows are added together

Out[ ]=

1	3	2	5	2
−1·1	3·3	−2·2	0	0
0	−1·3	3·2	−2·5	0
0	0	−1·2	3·5	−2·2

This example uses the same kernel on a larger list.

Out[ ]=

1	3	2	5	2	0	6	1
−1·1	3·3	−2·2	0	0	0	0	0
0	−1·3	3·2	−2·5	0	0	0	0
0	0	−1·2	3·5	−2·2	0	0	0
0	0	0	−1·5	3·2	−2·0	0	0
0	0	0	0	−1·2	3·0	−2·6	0
0	0	0	0	0	−1·0	3·6	−2·1

It is possible to have different boundary conditions, e.g., it can wrap around.

Out[71]=

The result of wrapping around gives a list of the same size,

Out[ ]=

	1	3	2	5	2	6	
6	1	3	2	5	2	6	1
−1.6	3.1	−2.3	0	0	0	0	0
0	−1.1	3.3	−2.2	0	0	0	0
0	0	−1.3	3.2	−2.5	0	0	0
0	0	0	−1.2	3.5	−2.2	0	0
0	0	0	0	−1.5	3.2	−2.6	0
0	0	0	0	0	−1.2	3.6	−2.1
	−9	4	−7	9	−11	14	

but there are other ways to end up with a list of the same size like padding with zeros.

Out[ ]=

	1	3	2	5	2	6	
0	1	3	2	5	2	6	0
−1.0	3.1	−2.3	0	0	0	0	0
0	−1.1	3.3	−2.2	0	0	0	0
0	0	−1.3	3.2	−2.5	0	0	0
0	0	0	−1.2	3.5	−2.2	0	0
0	0	0	0	−1.5	3.2	−2.6	0
0	0	0	0	0	−1.2	3.6	−2.0
	−3	4	−7	9	−11	16	

Idea of Convolution

One idea for using inner products comes from the standard basis. For example, when one takes v dot a standard basis vector e_2 then the result is how much e_2 is in v. Generalize that to an inner product with v of a test vector ψ to say how much of ψ is in v. For carefully chosen ψ the inner product takes on different interpretations.

By shifting the test vector ψ one gets a sequence of interpretations centered on each possible position.

We are going to introduce four applications of convolutions.

Moving Averages

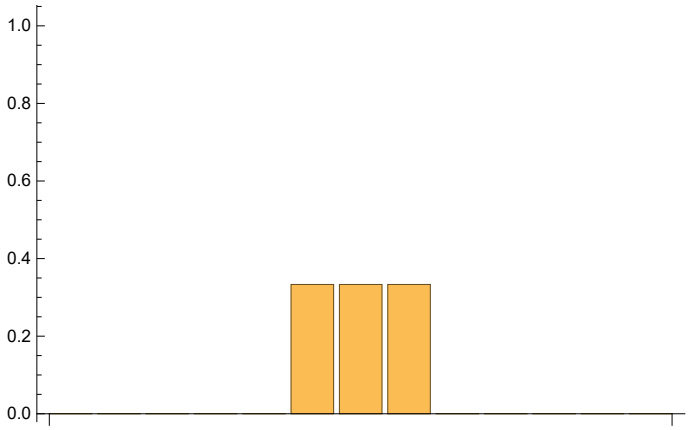
The test vector for the average is just (1 1 1 ... 1)/*n* and shifting that vector does not change its value.

More interesting is to take the average near different positions. For instance, consider a time series of rainfalls. (1.1 1.2 .1 0 ...) where on day 1 it rained 1.1 inches, day 2 it rained 1.2 inches, day 3 it

rained .1 inches, day 4 it rained 0 inches. If we want to average the rainfall to get another timeseries, we can choose to only consider the rain the day before, the day of, and the day after.

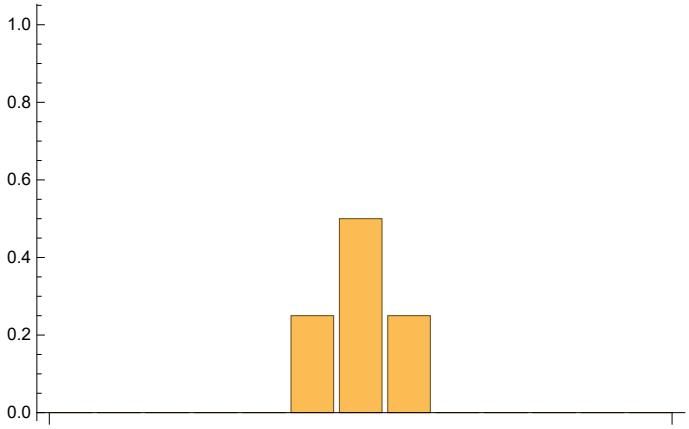
The choice of test vector would be (1 1 0 0 ... 0 1)/3 We divide by three because the total of the weights is three. The positions are treated cyclically. The last element is the element to the left of the first element.

Out[ ]=



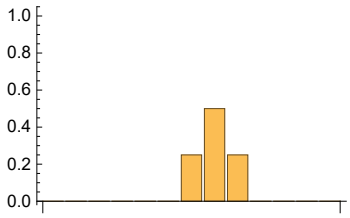
There are many choices to make. How many days to include in the average, and also whether to weight them differently. For instance, one might choose to weight the day in question twice as much with (2 1 0 0 ... 0 1)/4 Note the total of the weights is 4 so we divide by 4.

Out[ ]=

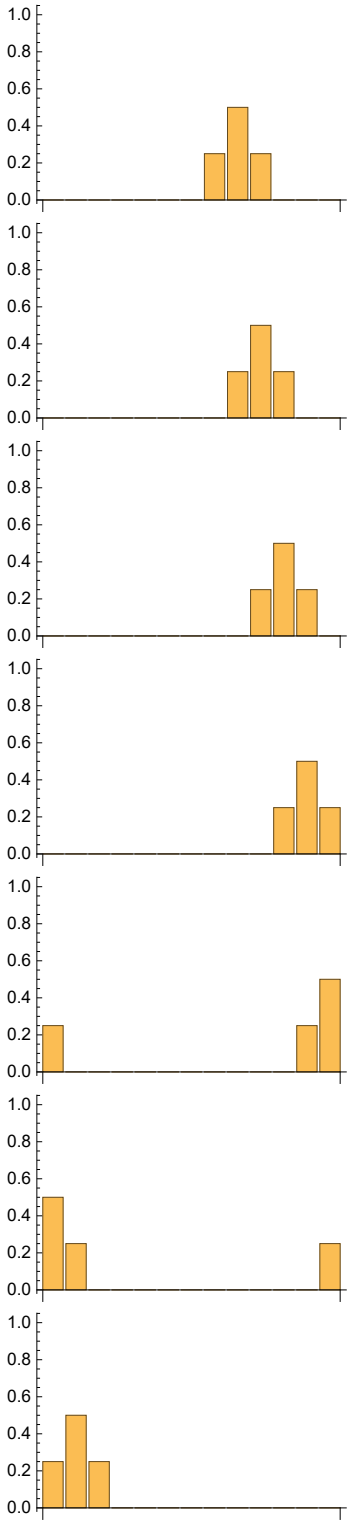


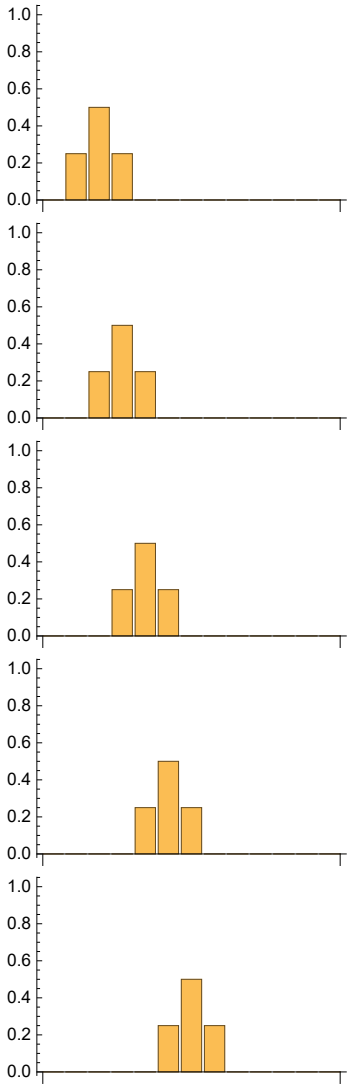
A typical approach to smooth out data is to use a convolution, and there are many possible kernels to choose from.

Here are the translated test vectors of the kernel.



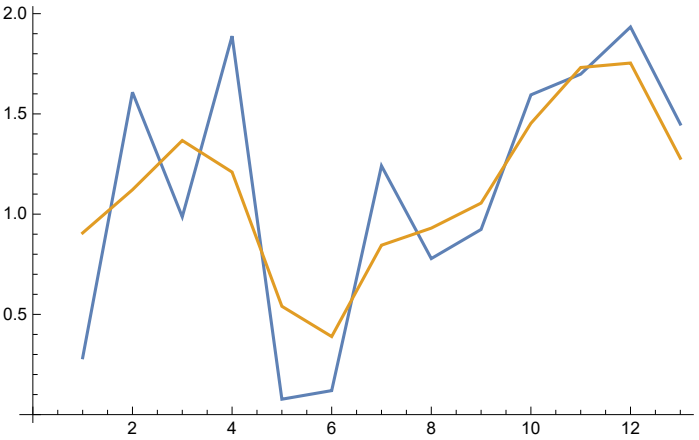
Out[ ]=





Here is an example of such a moving average.

Out[ ]=



Convolution Layers

Recall that a layer f: X → Y in a neural network transforms tensors. The shapes of X and Y are part of the definition of the layer.

One basic type is the dot layer defined by a matrix M which transforms vectors x by y = M dot x.

This generalizes to tensors of any rank.

Continuous Cellular Automaton

What happens when you convolve over and over again? It is still a linear map, but if you want to get something non-linear, one approach is to add a on-linear element after the convolution.

In this case, if the result is positive we make a 1 and if it is negative we make a 0.

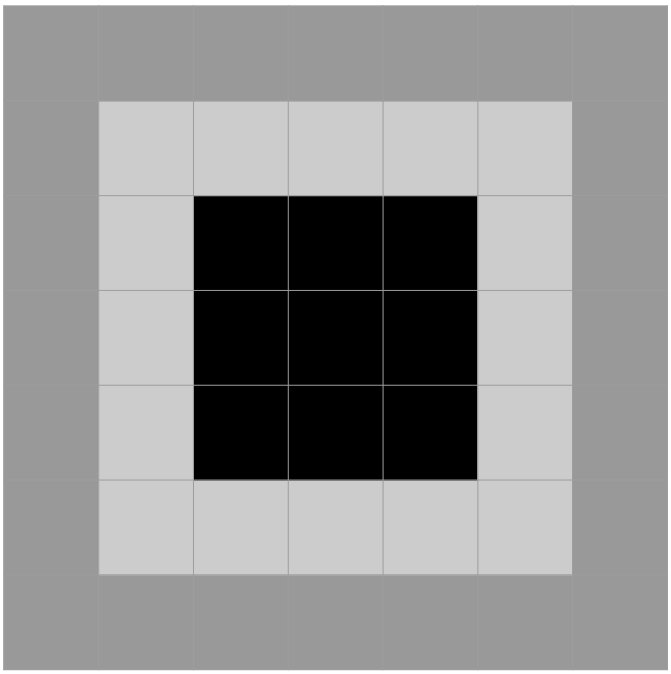
In[ ]:=
```
WMatrix[{w0_, w1_, w2_, w3_}] :=
 w3 (BoxMatrix[3, 7] - BoxMatrix[2, 7]) + w2 (BoxMatrix[2, 7] - BoxMatrix[1, 7]) +
  w1 (BoxMatrix[1, 7] - BoxMatrix[0, 7]) + w0 (BoxMatrix[0, 7])
```

In[ ]:=
```
WMatrix[{w2_, w3_}] := WMatrix[{1, 1, w2, w3}]
```

Here is a plot of kernel in a few formats.

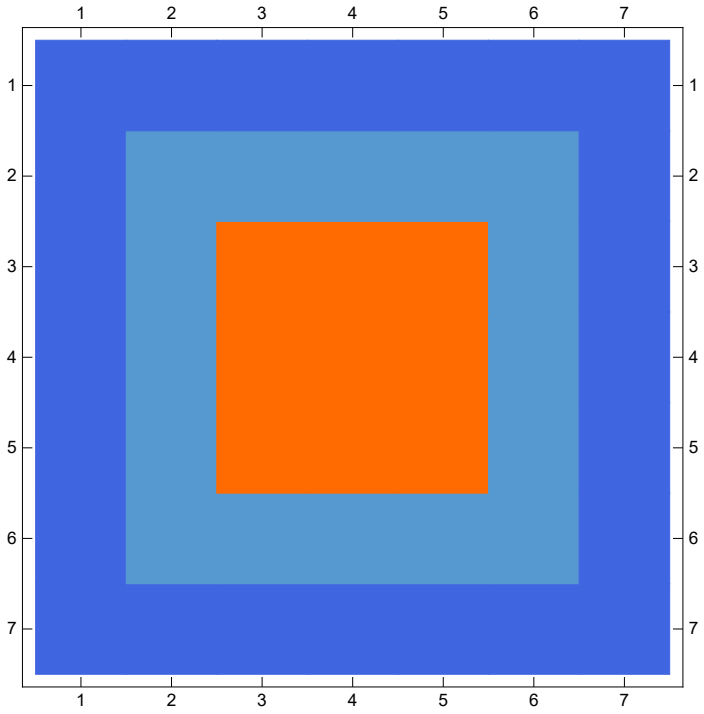
In[ ]:=
```
ArrayPlot[WMatrix[{-.2, -.4}], Mesh → True]
```

Out[ ]=



In[ ]:=
```
MatrixPlot[WMatrix[{-.2, -.4}]]
```

Out[ ]=



In[ ]:=
```
MatrixForm[WMatrix[{-.2, -.4}]]
```

Out[ ]//MatrixForm=

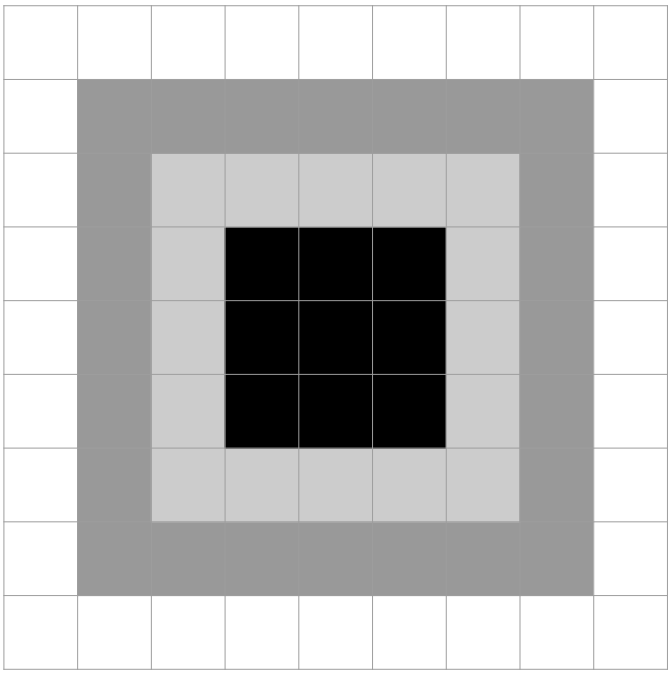
$$\begin{pmatrix} -0.4 & -0.4 & -0.4 & -0.4 & -0.4 & -0.4 & -0.4 \\ -0.4 & -0.2 & -0.2 & -0.2 & -0.2 & -0.2 & -0.4 \\ -0.4 & -0.2 & 1. & 1. & 1. & -0.2 & -0.4 \\ -0.4 & -0.2 & 1. & 1. & 1. & -0.2 & -0.4 \\ -0.4 & -0.2 & 1. & 1. & 1. & -0.2 & -0.4 \\ -0.4 & -0.2 & -0.2 & -0.2 & -0.2 & -0.2 & -0.4 \\ -0.4 & -0.4 & -0.4 & -0.4 & -0.4 & -0.4 & -0.4 \end{pmatrix}$$

The correct overhangs for a matrix of this size is {4,-4}

This shows those overhangs tested by list convolving with a single 1 in a 9x9 grid.

In[ ]:=
```
ArrayPlot[
 ListConvolve[WMatrix[{-.2, -.4}], ArrayPad[{{1}}, {4, 4}], {1, -1} 4], Mesh → True]
```

Out[ ]=

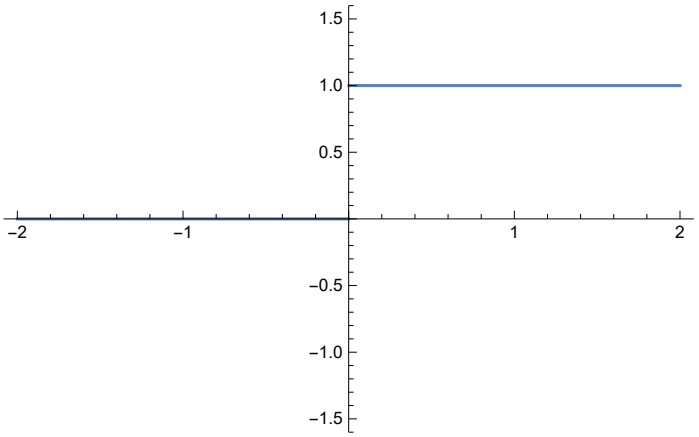


By composing with UnitStep, we implement the nonlinear portion which converts a negative value to 0 and a positive value to 1.

Here is a plot of UnitStep.

In[ ]:=
```
Plot[UnitStep[x], {x, -2, 2}, PlotRange → 1.6]
```

Out[ ]=



In Matlab you can use heaviside(x) and in Python (Numpy) you can use heaviside(x,0) The Heaviside function is the same as UnitStep except it is 1/2 at 0.

Here is a Mathematica implementation of one step of the cellular automaton. It depends on the parameters w2 and w3.

In[ ]:=
```
LCStep[{w2_, w3_}][array_] := UnitStep[ListConvolve[WMatrix[{w2, w3}], array, {1, -1} 4]]
```

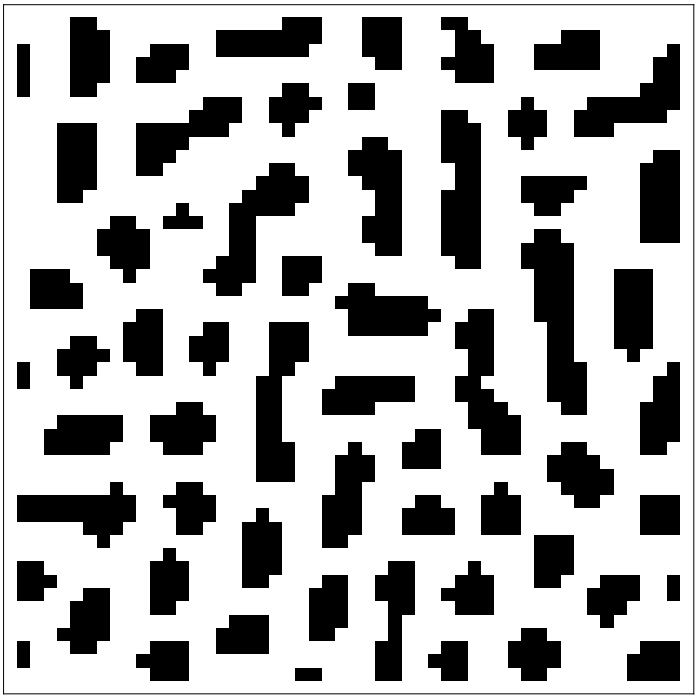
This implements t steps of the evolution.

In[ ]:=
```
LCEvolve[{w2_, w3_}, array_, t_] := Nest[LCStep[{w2, w3}], array, t]
```

This shows the result after 5 steps with random initial conditions on a 50 by 50 grid.

In[64]:=
```
ArrayPlot[LCEvolve[{-.2, -.4}, RandomInteger[1, {50, 50}], 5]]
```

Out[64]=

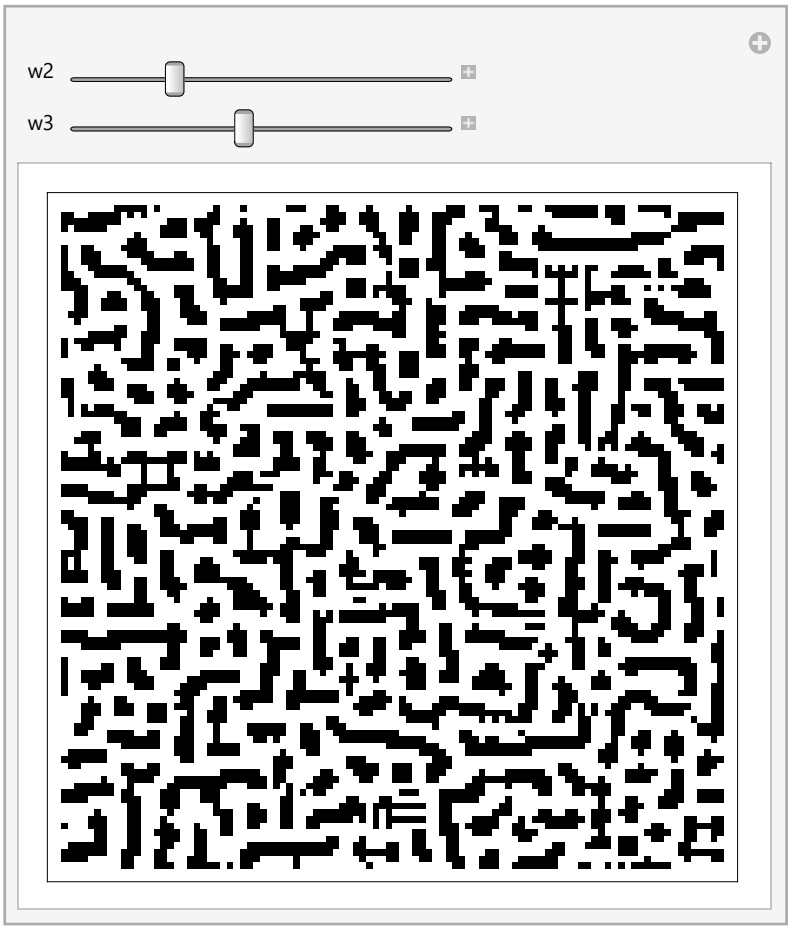


One can manipulate the parameters.

In[ ]:=
```
init2 = BlockRandom[SeedRandom[30]; RandomInteger[1, {100, 100}]];
```

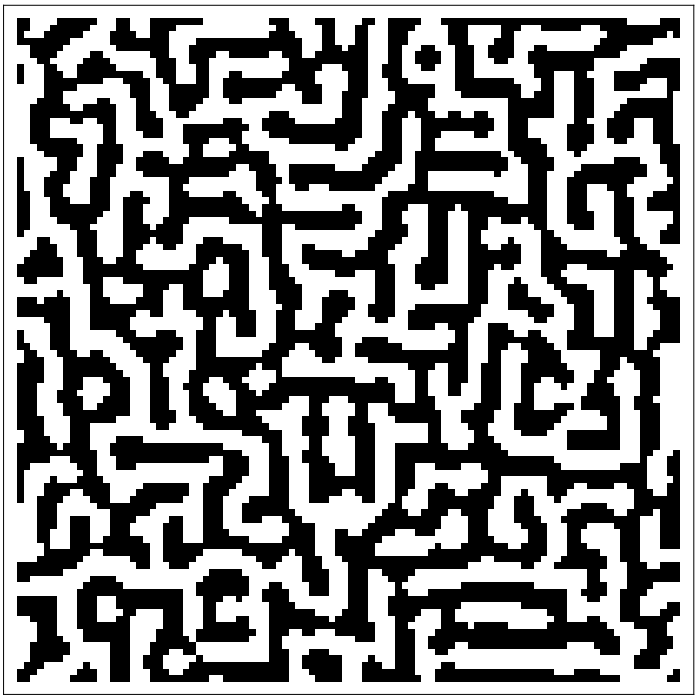
In[ ]:=
```
Manipulate[ArrayPlot[LCEvolve[{w2, w3}, init2, 5]], {{w2, -.5}, -1, 1}, {{w3, -.1}, -1, 1}]
```

Out[ ]=



In[63]:=
```
ArrayPlot[LCEvolve[{.043, -.4}, init2, 5]]
```

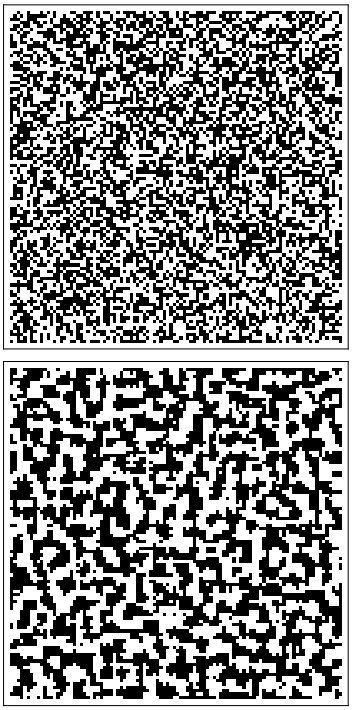
Out[63]=



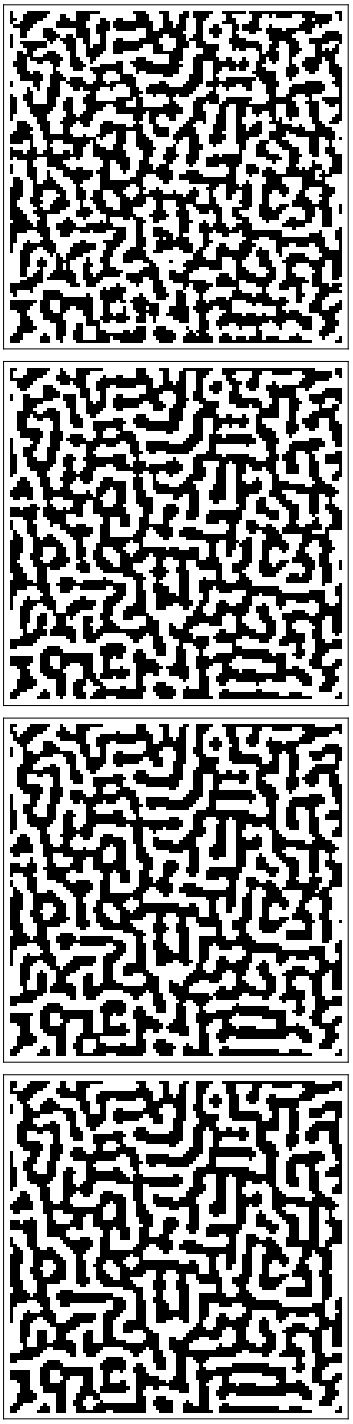
We can see how the pattern evolves by looking at intermediate time steps.

In[ ]:=
```
LCEvolveList[{w2_, w3_}, array_, t_] := NestList[LCStep[{w2, w3}], array, t]
```

In[ ]:=
```
Column[Map[ArrayPlot, LCEvolveList[{.043, -.4}, init2, 5]]]
```



Out[ ]=



Convolution for functions

In Calculus, a convolution is defined between two functions:

$f * g(x) = \int f(x-y)\, g(y)\, dy$

This corresponds to the list convolution we have been considering when the function is like a step

function.

Implementation Issues

Overhangs

When convolving a list you have to decide what to do at the boundary.

Is the convolution cyclic? It was for the continuous cellular automata. In that case the right neighbor of the left boundary is the element at the right boundary.

Do you only care to convolve what is inside? So it is not cyclic and there is no padding. For instance in the moving averages it is impossible to center the moving average near the boundary.

Padding

Padding is one solution to whether you want the convolution to wrap around, or be considered to be in a sea of zeros, or to shrink in size. This can be preferable to dealing with overhangs, but requires some work to implement.

Mathematica

Use ListConvolve

For the cyclic case use

```
ListConvolve[kernel, list, k]
```

where k = Ceiling[Length[kernel]/2]

For the noncyclic case you don't want any overhangs

```
ListConvolve[kernel, list, {1, -1}]
```

Matlab

One dimensional convolution of the type we are considering is conv(list, kernel, 'valid')

Two dimensional convolution of the type we are considering is conv2(list, kernel, 'valid')

It appears that you should adjust the boundaries of the arrays to get the kind of overhang you want.

For the unit step function you can use instead heaviside(x)

Python

One dimensional convolution of the type we are considering is numpy.convolve(list, kernel, 'valid')

It appears that you should adjust the boundaries of the arrays to get the kind of overhang you want.

The 2D convolution does not seem to be standard, but there are codelets on the internet.

For the unit step function you can use instead heaviside(x,0)