CANN: Cellular Automata Neural Network

Keiland W. Cooper

Introduction:

The goal of this model is attempt to replicate some of the dynamics of a neural system with a cellular automa architecture. The implementation takes the general assumption of a neuron:

1. *A cell will fire when it gets n input from other cells (usually n > 2)*
2. *A cell can fire only for t = 1*
3. *A cell must undergo a refractory phase, for t = 1, until it can fire again*

These assumptions will be implemented in the cellular automa similar to the assumptions above, such that to fire, cell x must have not fired within one timestep and have greater than two neighboring cells also firing. Thus, each cell must retain 'memory' of its past firing. In addition, I also added the option for a null cells to exist, to emulate structure to see if the dynamics of the activity alter. Lastly, I attempted to emulate excitatory and inhibitory cells to no avail, which could be a future problem to entertain. I also tried a different approach, trying to achieve this with one matrix:

1. *If a cell is on (1), then at t+1 it will be refractory (-1)*
2. *If a cell is refractory, then the next step it will be Off-and-ready (0)*
3. *If this cell has two neighbors that are on, then it will fire (1)*

This produced interesting results, such as wave like patterns and oscillations.

(I also included a version of Game Of Life that I made)

Methods:

To implement the grid and the information on each cell I used three matrices:

1. Fire/Off - Keeps track of the if the cell is firing (0,1)
2. Cell Type - Type of cells (-1,0, 1)
3. Fire History (0,1,2...n) depending on your specified delay required

These three matrices are highly interconnected and rely on each to be updated for the program to function properly. Getting the memory aspect to work correctly is tricky, and there are still bugs that need to be worked out for a full simulation to work properly. fire/off or grid as it's named in the code is the last step to be altered, and is what you are visually seeing as you look at the animation. Each indice of grid also corresponds to the information in the other matrices.

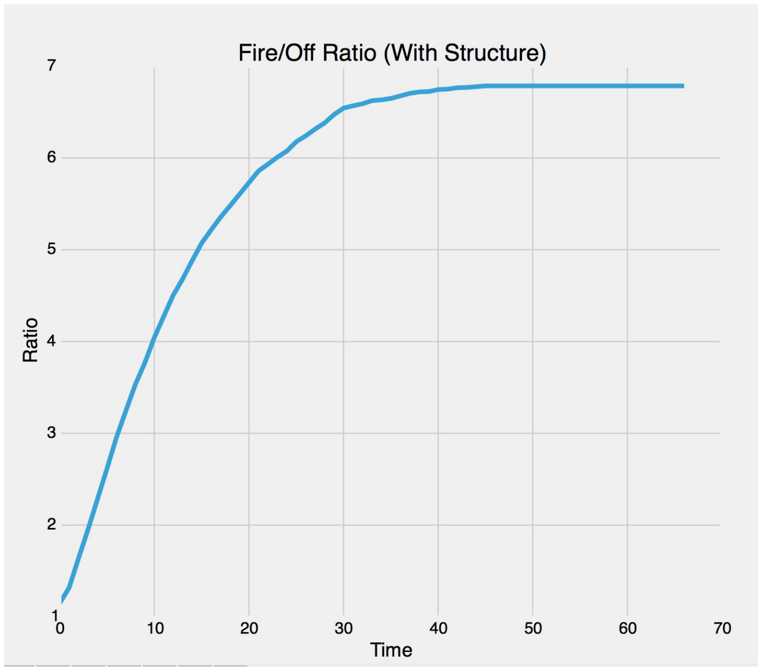
The cells around each selected cell are acknowledged with a modified kin detection function. This value is then passed to step() for further evaluation by the rest of the algorithm.

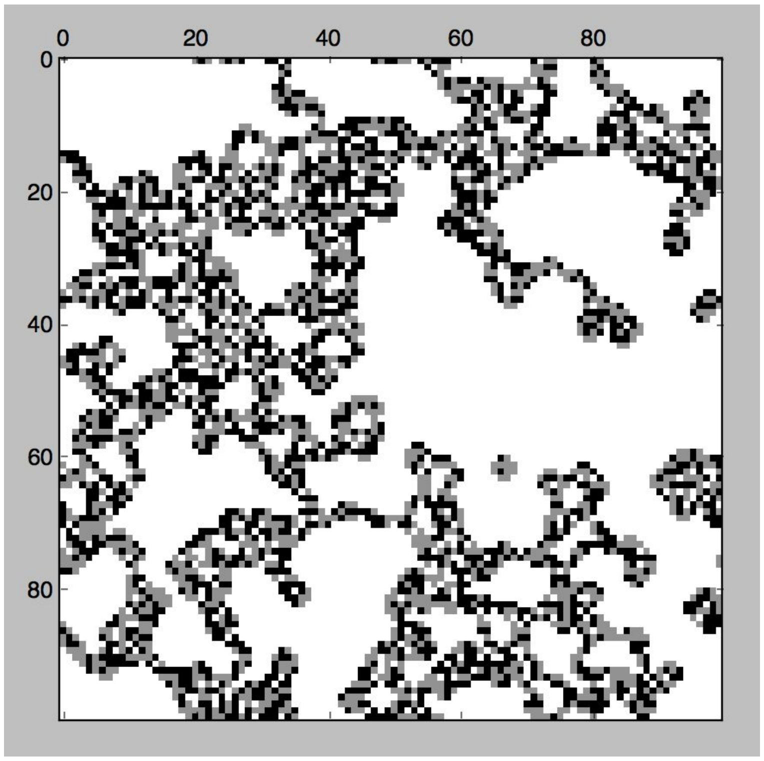
The graphics were also an interesting challenge, as they required a bit of debugging for the live version of the program to function properly. I used matplotlibs animate feature, and it allowed me to flash the updated grid for viewing the changes.

Other considerations are the ratios of each of the types of cells, and the differences between these proportions. In addition, probabilistic firing could also be an interesting idea to look at for the implementation of the model.

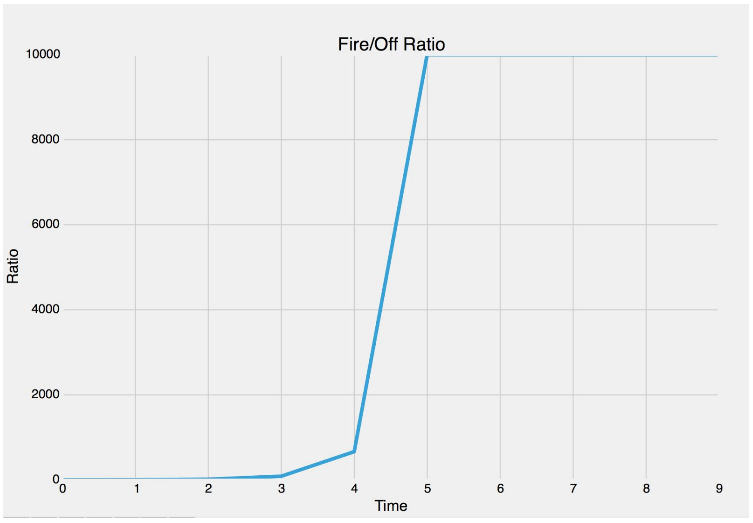
Results:
I successfully modeled the effects of an excitatory network lacking inhibition.
It is evident from the model that the activity will spread unchecked if there are no other cell types in place.

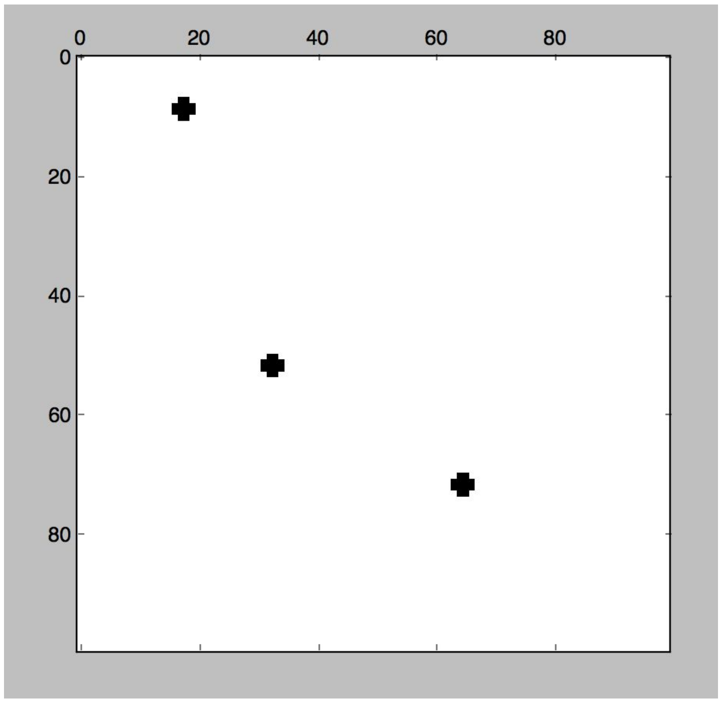
I plotted the ratio between a cell being on vs it being off over time. A curve is evident as the activity spreads.



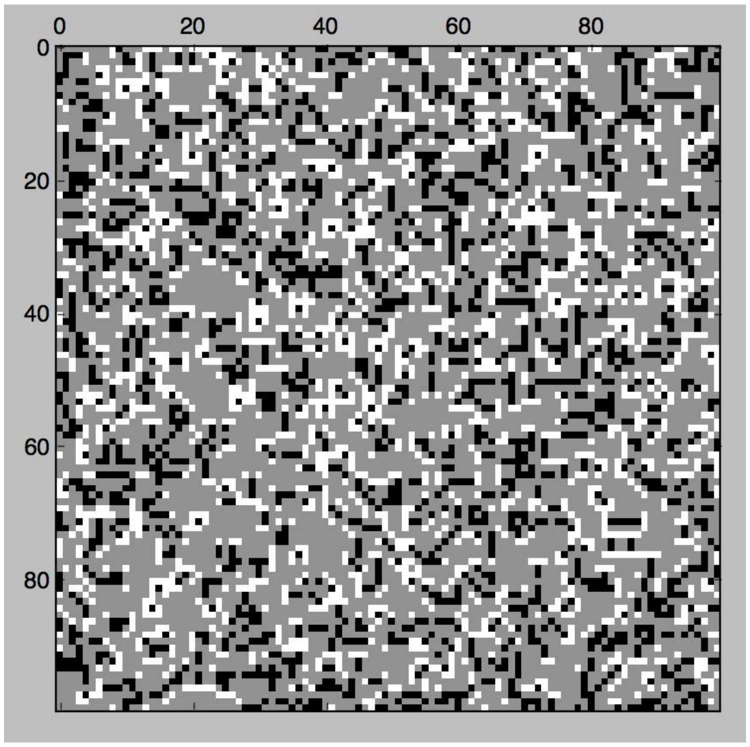


Above is the final state of the activation spread (white), with structure (grey). Black represents 'off' cells

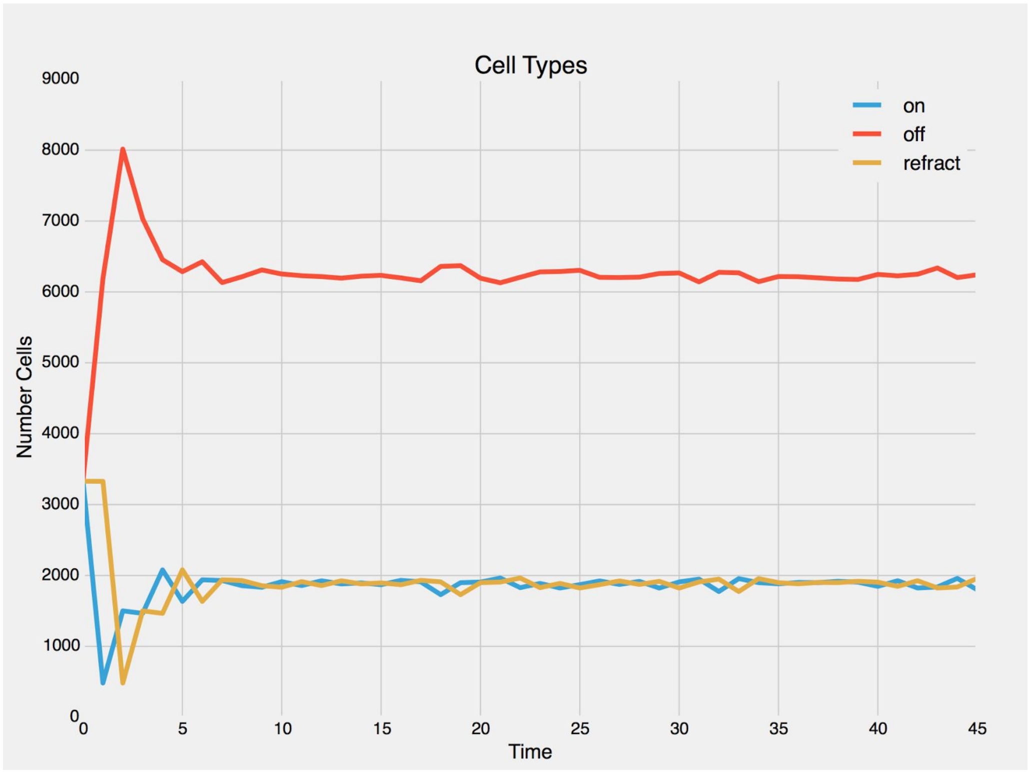




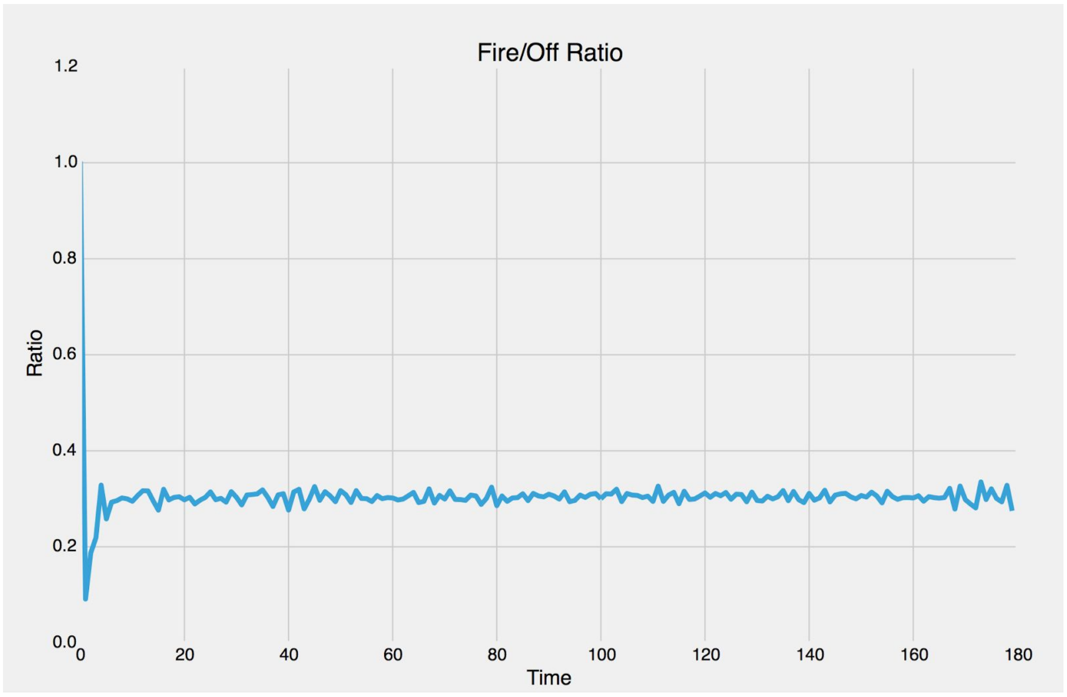
Pockets of off activity can be recognized occasionally, however there is a stark appearance of the graphs sharp accent.



This graph depicts the steady state of the neuronal simulation.



It can be see the ratios between the cell types. On and refractory are closely related.


Fire/Off Ratio
Ratio
Time