Neural Networks

Rowan Comish

Computational and Physics Background

Neural networks are the result of attempting to model the way a brain functions. They function based off of whether each neuron is firing. A neuron is a node that is connected to other neurons though an output and an input channel. It is through these connections that a wide variety of calculations can be made. In this project, I attempt to model it using an Ising model, with some minor changes in its implementation.

In order to achieve the functionality similar to the brain, we are interested in having the neurons perform as a memory storage and image identifier. The memories will be patterns stored as NxN arrays. The image identifier will use the way we set up the energy calculation in order to minimize the energy:

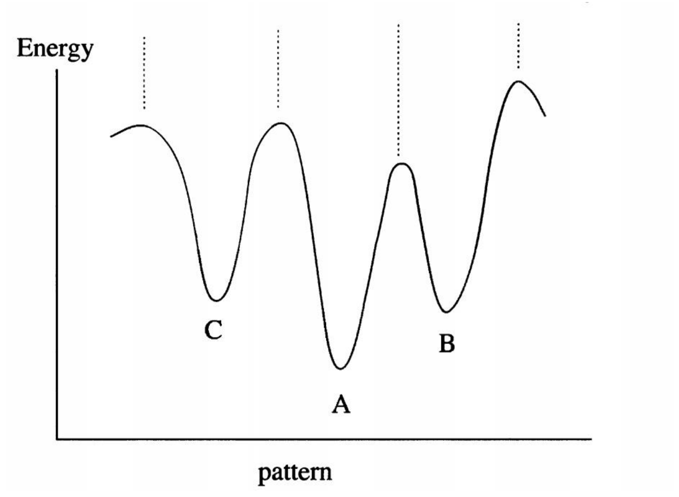
$$E = -\sum_{i,j} J_{ij}\, s_i s_j$$

Where $s_i s_j$ are the values of the ith and jth neurons. For simplicity we will have +1 as firing and -1 as not. A difference between this model and the ising model, is that all the neurons will be connected to each other. This is similar to how a brain works.

J_{ij} is the interaction strength between neurons, and this will be dependent on the patterns we have stored.

$$J_{ij} = \frac{1}{M}\sum_{m} s_i(m) s_j(m)$$

Where M is the number of patterns stored and $s_i(m)s_j(m)$ are the corresponding neuron states of pattern m. This will result in our stored patterns having the most negative energies, as the $s_i(m)s_j(m)\ \ and\ s_i s_j$ will be equal when a pattern is the same as one of our stored patterns. Having multiple patterns will result in energy system similar to:



Where A,B,C are the patterns corresponding to the letters. They are local minima which patterns will tend to if they are in the area near one of the minima. This model shows how, if two patterns are closely related, they would be very close together and therefore could result in overlapping patterns.

The standard ising model will be used to analyze patterns and the Monte-Carlo flipping rule as well. We must systematically give every neuron the chance to flip instead of randomly choosing one in the normal Monte-Carlo way. This will make the computation time needed fairly long depending on the size of the patterns. The J_{ij} itself is an n**2 x n**2 matrix, that is needed to store all the interaction energies.We will accept the flip if the change in energy is negative and reject it if it is positive. This is the most simplest model, and we will also investigate cooling, where the positive change in energy will have a probability to be accepted of $P = \exp(-\frac{dE}{kT})$, and for simplicity we will set k=1.There is a theoretical limit to the amount of patterns we can store(0.13N**2), and we will investigate the actual amount we can store without having the patterns overlap.

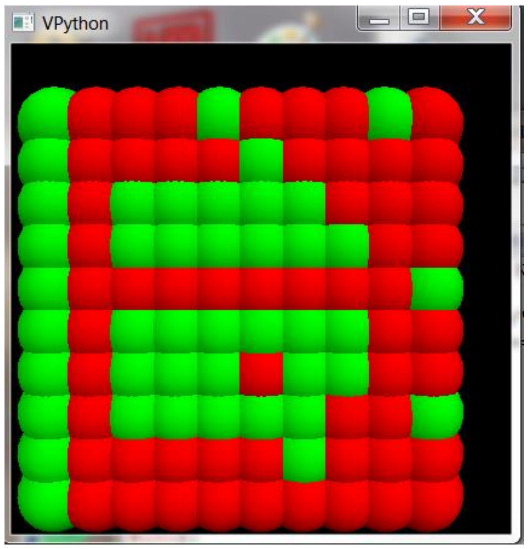
Neural_Netowrk.py has most of the code and results for this project(most of it is animation in vpython). The basic function is selecting P1, as the input for energy and selecting which way you want the computation to be done. For choice = 1, it animates a monte-carlo flip throughout the pattern, and attempts to match it to the other patterns. If P2, or P3 are chosen, we still see that the pattern tends to the pattern that was flipped, even though there are other patterns in the memory.

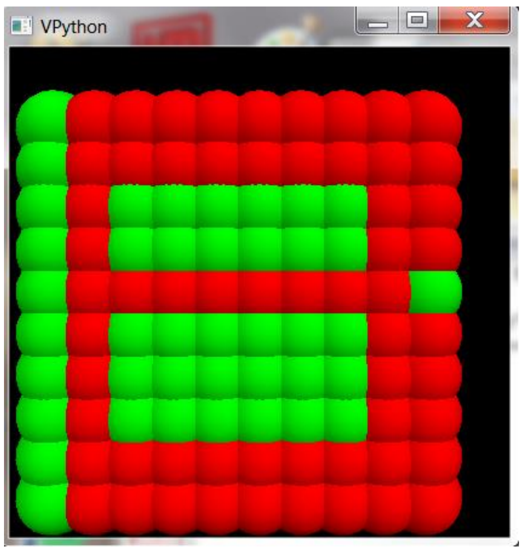
This tendency changes as we increase the number of stored patterns. When I had P4 or 4 patterns in the memory, the result of the flipping was not what was expected. Two things happened:

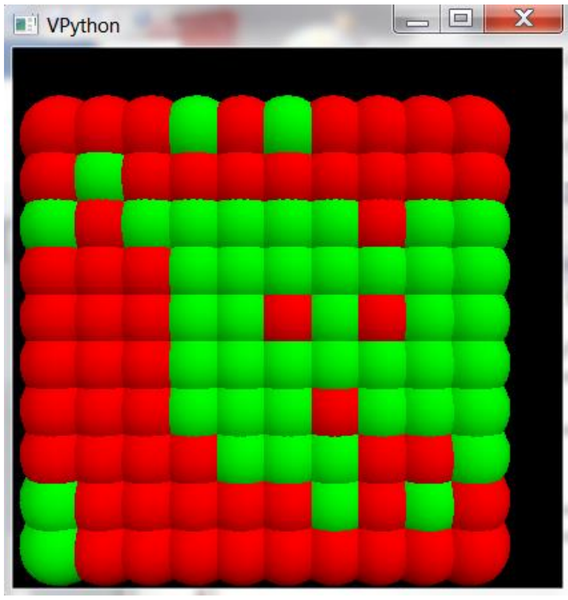
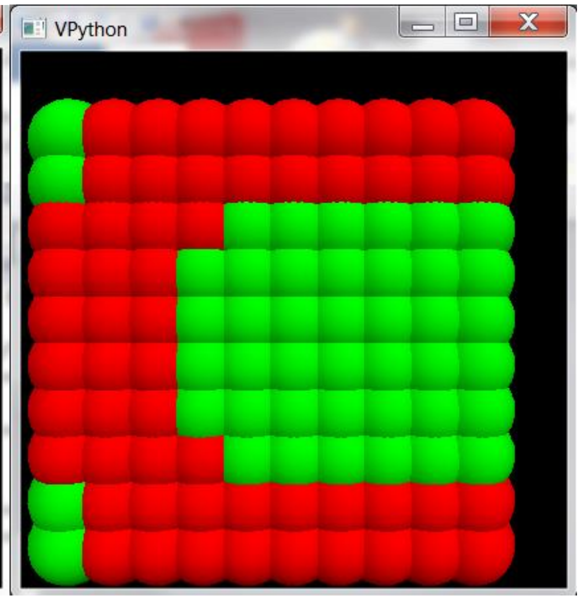
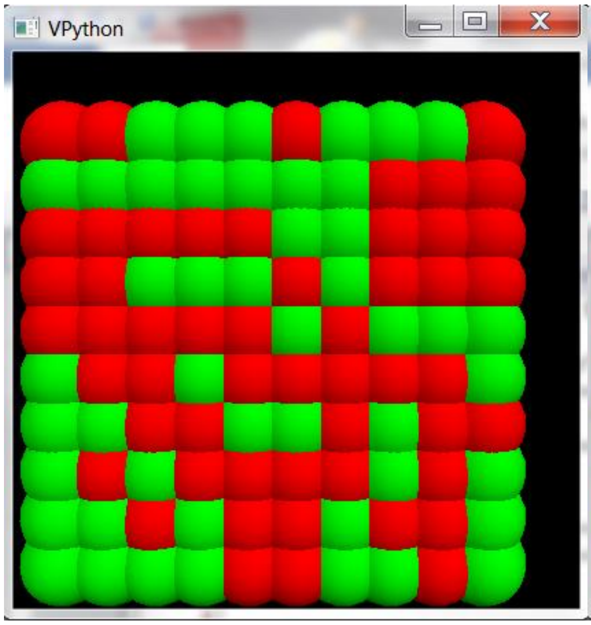
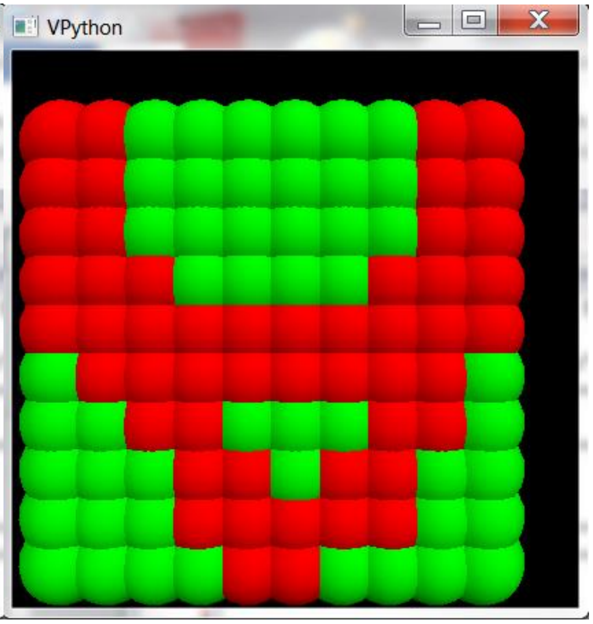
1. A new pattern was the result, which was kind of in the middle of two of the stored ones
2. We tended toward a pattern, for example B, and then ended up in pattern, C for example. This is an example of two patterns sharing a relatively close distance on the energy graph so an input could tend towards one rather than the other.

In 1Pattern.py, I investigated the ability of the network to identify patterns, when a percentage of the Jij was set to 0. The threshold I observed was around 0.5, anything greater than that, the pattern would not converge to one of the stored patterns.

The monte-carlo method where flips were allowed to have a dE > 0 using the temperature and Boltzmann probability worked more reliably than the basic method. Although the temperature method sometimes took longer because it would have to iterate again where one of the neurons was flipped. The values of Tmax, Tmin and tau were chosen because they provided a convergence in the quickest manner.
Here are some examples of before and after:






VPython

VPython

VPython

VPython

References:

Goodfellow et al*, Deep Learnin*g, Book in preparation for MIT Press, 2015. http://goodfeli.github.io/dlbook/.Web.

Giordano, Nicholas J. *Computational Physics.* Upper Saddle River NJ: Prentice Hall, 1977. Print.