Random Neural Networks : Experiments with Learning, Architecture, and Brainwashing

Patrick McGuire

27 April 1990

1 Introduction

John Clark states,"The first objective of neural-network computer simulation is to produce a net which, after an initial instantaneous stimulus to a selected set of neurons, remains active and does not display catastrophic behavior. The second objective of computer simulation is to produce interesting cycling behavior."

In the last report, I stated that we had successfully modelled a network in discrete time that fulfilled Clark's first objective, that is: we were able to keep our randomly connected net from becoming epileptic and from dying. We accomplished this by brainwashing the initially random distribution of synaptic strengths, so that the resulting distribution was much narrower and much more interesting. But we were unable to observe the fabled cycling of the network's activity, and thus fulfill Clark's second objective for the network modeller.

Since the last report, we have not only observed cycling in a random network, but we have also seen cycling of activity in architectured networks. We have also been able to change the amplitude and the frequency of the cycles to a small degree. Henrik Bohr and I used a randomly connected neural network as a hidden layer in a Perceptron-like network. We used back propagation to train the network to associate a certain input with a particular output. Unfortunately, due to the cycling behavior of the random hidden layer and the small number of different cycles of the random layer, we were unable to train the network to recognize more than one or two different input spatial patterns. Also, since the last report, I have investigated two different ways to quantify the different activity patterns of a network: the Grassberger-Procaccia method of determining the scaling exponent of the two-particle correlation function of the network's activity, and Fourier analysis to determine the power spectrum as a function of cycle frequency. We may use these two methods in real-time, so that we may easily quantify the evolution of a network for a certain choice of parameter environment, or under a changing parameter environment.

2 experiments

2.1 learning

Dr. Bohr wanted to see how a perceptron/back-propagation network behaved when random lateral connections were added in the hidden layer of a three layer network. He thought that the lateral connections would cause leakage of the hidden layer's spatial representation of the spatial input pattern to other hidden layer neurons. Hence a well-defined spatial input pattern would become less well-defined in the hidden layer, and even less well-defined in the output layer, due to the random lateral connections. We also knew that the temporal structure of a randomly-connected network would make pattern-learning difficult.

Nevertheless, we tried to test Bohr's hypothesis by connecting input and output synapses to a single random network. We specified which spatial input pattern to feed the input synapses, and neurons on the post-synaptic side of the output synapses responded to the spatial input pattern, as filtered through three sets of synapses: the input, the hidden-layer-lateral, and the output synapses.We tried several different input and output synaptic strength distributions, e.g.: all excitatory, some inhibitory, random magnitude of strengths (initially),constant magnitude of strengths(initially), a single I/O synapse per hidden-layer neuron, or many I/O synapses per hidden-layer- neuron. We tried any of the above options in the I/O synaptic distribution concurrently.

Then, we used the back-propagation (BP) technique to adjust the I/O synaptic strength (weights), so that after a long enough time the network would filter a given spatial input pattern of neural firing states into a given spatial output pattern of neural firing states. Rumelhart's BP technique says to compare the observed output pattern to the desired output pattern,using some metric, then change the weights between the output layer and the hidden layer to try to minimize the metric at the next time step. After doing this, we determine an expected hidden layer pattern as a function of the output expected and observed patterns, the change of the weights, and the derivative w.r.t. firing state of the threshold linearity function. We continued the recursive BP process over the last set of weights, from the hidden layer to the input neurons.

Unfortunately, when we applied the back-propagation technique alone to our perceptron with random hidden layer (PRHL), the PRHL could not learn to recognize any patterns whatsoever. I attributed this to the fact that BP does not allow very well for time dependent firing functions. With output thresholds that are proportional to the total activity of the network, we taught the network to recall at most three patterns for no more than 40% of the time. With signal averaging over many time steps, we might be able to teach the desired input/output pattern relationship to the PRHL, though we probably would have slow convergence. I looked at the matrix of output weights before and after I taught the PRHL slowly (bw = -1%) for one hundred time steps. I noticed that about 80to zero. We probably did not renormalize properly.Even so, at the time, I attributed the large number of zeroes in the connection matrix to the low volatility of our HL. Volatility is a measure of the number of different cycling patterns easily available to the net. I do not have a good numerical measure of volatility yet, but I can judge qualitatively by eye how many different cycling patterns result from a series of small kicks to the network. I had been able to get no more than eight different patterns, regardless of parameter choices. Hence, I put pattern recognition by PRHL on the back burner, until I could find a much more volatile random network.

2.2 volatility

In order to find a more volatile random network, I first tried to vary parameters of the unstructured random network. After a couple of weeks, we had our parameters tuned well enough, that we were able to observe cycling every time we turned the network on. But we were unable to kick the net and observe a different cycle. However, I did observe that when I had the network cycling already, and then I changed the global decay parameter from 30/that the network changed cyclic modes. I did not have a measure of this change, but I thought that the difference between the cyclic modes was substantial.

Then I worked on a way to quantitatively measure the difference between cyclic modes. I developed a fast fourier analysis technique that takes the total activity of the network for 2^n time steps, and does a real fft,which allows me to plot the spectral density versus frequency. $n \geq 8$ gives good approximation to a delta function for a sinusoidal function: the magnitude of the spike on a plot of the spectral density , $P(f) = C^2 + S^2$, vs. frequency (cycles/step) is at least 100 times the valleys around the spike,and the resolution goes like $(1/n)$. But I had trouble for $n = 12$, as when I plotted some of my power spectra for a periodic trajectory, much white noise was present, and no spikes. Also, for $n \geq 13$, the computer took an inordinately long amount of time (30 min) to calculate the power spectrum. I also did an FFT for the activity of each neuron and added all the neurons' FFTs together, to determine the spectral density, but I found that this was equivalent to the spectral density of the entire network. This is obvious in retrospect, since the FFT is a linear operator. I might have determined the spectral density of each neuron, $P(f) = C^2 + S^2$, and then added the spectral densities of all the neurons together, but I did not think about that until right now. Also there may be some way to coherently add the transforms of the neural activities together, but since the neurons are not on a spatial grid, I cannot think of an obvious way to do this.

2.3 Experiments with brainwashing and decay

I present results for the evolution of the power spectrum, as a function of brainwashing time (with and without architecture) and as a function of the decay parameter. With time slowed down by a factor of five, and the decay rate held constant ($dec = dec/5 = 0.06/step$) and the length of the action potentials multiplied by five, I did spectral analysis of the total activity of the network for 2^{11} steps. I then brainwashed the network at 0.2/time steps, after which I let the transients die out for 300 time steps. Then I again did spectral analysis. I repeated this process a total of 500 times, both for an unarchitectured random network ($N = 100, M = 5, \mid V_{max}(t = 0) \mid = 500$), and for an architectured network($Netlets = 3, N_{netlet} = 33, M_{int} = 4, M_{ext} = 1, \mid V_{maxint}(t = 0) \mid = 500, \mid V_{maxext}(t = 0) \mid = 2000$).

For the unarchitectured network, the power spectrum became more sharply peaked around f=0 as the total amount of brainwashed time increased, for $bwtime < bwtime_{max}$.After $bwtime_{max}$, the power spectrum stayed quite the same for a long while. Then after some other time, $bwtime_C$, intermittent bursts of high frequency activity appeared. Perhaps this was a period of nearly constant activity, but since I automated the procedure, and did not store the complete power spectrum on disk, I am unsure what this intermittency means. Also the total integrated power of the network increased exponentially with brainwashed time.

For the architectured network, I only brainwashed the internal connections of the netlets, and the synaptic strengths of the connections between the netlets remained constant. The spectral density distribution fluctuated rapidly as a function of brainwashed time, with quite an intermittent character. Sometimes the distribution would be sharply peaked around f=0, and sometimes the distribution would be relatively flat, or with high frequency peaks. Also, the total integrated power was relatively constant with total brainwashed time, but the intermittency is evident. I need to study this situation in more detail.

I tried to fit the power spectrum to the sum of two exponentials, $P(f) = 10^{a_1 - f/f_1} + 10^{a_2 - f/f_2}$, and then watch how f_1 changed with time. But this method of characterizing the power spectrum does not work well when the two exponents are nearly the same, when the power spectrum is not very smooth, and when the power spectrum is not the sum of two exponentials. The latter two conditions persisted for a great deal, so I chose another method to characterize the power spectrum. I decided to determine the frequency,$f(x)$, at which $x\%$ of the total integrated power was below f. I did this for several values of x for the unarchitectured net : $x \in (20, 50, 90, 99)$ and plotted these versus total brainwashed steps. Essentially, this is a contour plot of spectral density versus bwtime. On the graph, the top curve is for $x = 99\%$, the second from top is for $x = 90\%$, the third from the top is for $x = 50\%$, the fourth from the top is for $x = 20\%$. The spectral distribution narrows until bwtime = 1000, then it is essentially the same until bwtime = 2300, when high frequency components pop in intermittently. I did the same for the architectured network, but I only plotted the contour for $x = 99\%$,since the graph would be all black if I plotted the other three contours.

With slowed-down time, as I have implemented it, short cycles ($T \leq 100$) are no longer so omnipresent, so that a change in the cycle length does not affect the power spectrum as drastically as it does with normal time. Therefore, I used normal time to analyze the response of the network to a change of the decay parameter. I did the following for the same architectured network as before:

1) brainwash the internal netlet connections for bwtime = 150 @ bw = 1%/step.

2) decay = 0.

3) decay = decay + 0.05; run for 500 steps

4) FFT for 2^{11} steps.

5) Return to 3; until decay = 0.65

I looked at the power spectrum and determined the cycle length. for each of the different values of decay. I found the following cycle lengths:

Decay	Cycle Length
0.05	19
0.10	28
0.15	5
0.20	65
0.25	13
0.30	70
0.35	140
0.40	60
0.45	12
0.50	3
0.55	3
0.60	3
0.65	6

3 Conclusion

As you can see, we can change the properties of a network by brainwashing, by architecture and by changing global parameters such as decay. We do need to investigate such changes further, probably in some evolutionary scheme. But first we need to determine what we would like our network to do, so that we can maximize the network's ability to do that action. We can do this by taking an ensemble of networks, and changing each of their parameter sets a small amount per time step, and only keep those nets that tend , over a long time period to increase the network's ability to do that action. Such actions might include variability of cycle length, volatility, and learning ability.