The Hopfield Network

Brad Vandenberg-Daves

1 Introduction

The goal of this paper is to serve as an introduction to the Hopfield network as a model of associative memory, and to delve into the mathematical properties of the model which can help us understand and predict its behavior. In so doing, we will uncover the fundamental limitations of the Hopfield model and use simulations to attempt to reproduce analytically derived results about the network's storage capabilities.

The Hopfield Network as a model of associative memory was initially proposed in John Hopfield's 1982 paper on the subject. In it, he describes what he calls content addressable memory–what we will call associative memory–as a set of stored packets of information that can be retrieved of the basis of an incomplete or slightly altered version of one of these packets. For example, we might expect a person familiar with 20th century physics to be able to recall the name "Erwin Schrödinger" when prompted with the information "Erwin S", or "Ewrin Sxhördinger". The goal of the Hopfield network is to simulate this phenomenon in a trained recurrent neural network which can be fed an incomplete or corrupted input like "Erwin S" or "Ewrin Sxhördinger", and evolve to settle on a state that represents the corresponding stored memory–"Erwin Schrödinger". Hopfield's paper represented an important breakthrough in the modelling of associative memory in that it introduced a novel conceptualization of the phenomenon as an energy minimization problem. This connected associative memory with the mathematics of dynamical systems theory and eventually with that of condensed matter physics.

Hopfield's 1982 paper introduced the basic structure of the model including a prescription for training the network, a definition of the network energy, and a means of predicting the likelihood of a memory retrieval error. It also gave a rough estimate for the expected storage capacity of memories of the network. In particular, Hopfield showed that when the number of stored patterns in the network is too large in comparison to the number of neurons, the network will break down and be unable to retrieve the stored memories reliably. In the following decades, significant work was done to try to understand the mechanics of the Hopfield network, and in particular, the limitations on its storage capacity. Most notably, several papers by Daniel J. Amit et al. helped to shed light on the nature of the storage capacity problem. By applying mathematical techniques developed in condensed matter physics to the Hopfield network, Amit et al. were able to explain and predict the breakdown of memory in the network as a function of network storage capacity and another parameter that roughly translates to synaptic noise between neurons. The most powerful result of this work was the phase diagram of the Hopfield model which describes in a single figure the effectiveness of memory retrieval in every possible Hopfield network. This phase diagram will play a central role in motivating much of the results described throughout this paper, because in order to understand the phase diagram of the Hopfield network, one must first understand its deepest inner workings.

2 Network Dynamics

The first order of business is to describe the Hopfield network precisely. Just like in standard feed forward neural networks, the activity level of neurons in the Hopfield network is related by some activation function to the level of activation that neuron is receiving. We will the denote the ith neuron's activity level as s_i, its received activation as a_i, and the universal activation function as f. Then we can write

$$s_i = f(a_i - t_i)$$

where t_i is the neuron's activation threshold. In the standard Hopfield network model, we assume that $t_i = 0$ for every neuron, and we use the activation function f defined by

$$f(a_i) = \begin{cases} 1 \text{ if } a_i > 0, \\ -1 \text{ if } a_i < 0 \\ s_i \text{ if } a_i = 0 \end{cases}.$$

This means that when a neuron's activity level is updated (when the neuron "fires"), it will update to 1 if the activation it is receiving is positive, -1 if that activation is negative, and it will remain unchanged if the it is receiving zero activation. Notice that this limits the activity level of neurons to two states we can call on for $s_i = 1$ and off for $s_i = -1$. We could just as well represent these states with 0 and 1 as was done by John Hopfield in his original paper on the network, but we choose -1 and 1 called the Ising spin in order to make certain computations easier to preform and certain formulas easier to write.

The Hopfield network is an example of a recurrent neural network in that the activation received by any neuron in the network is determined by the Ising spin of every other neuron in the network. In particular, in a network of N neurons every pair of neurons (n_i, n_j) is connected by a weights w_{ij} and w_{ji} representing the affect of n_i on n_j and the affect of n_j on n_i respectively. In the standard Hopfield model we assume symmetric weights, i.e. $w_{ij} = w_{ji}$, and also no self re-enforcing weights, i.e. $w_{ii} = 0$. From these weights and the network state, we can calculate the activation received by any neuron with the expression

$$a_j = \sum_{i=1}^{N} w_{ij} s_i.$$

In words, this means that the activation of the ith neuron in the network is a linear combination of the Ising spin of every other neuron in the network with weights corresponding to the weights w_{ij} connecting n_i to n_j.

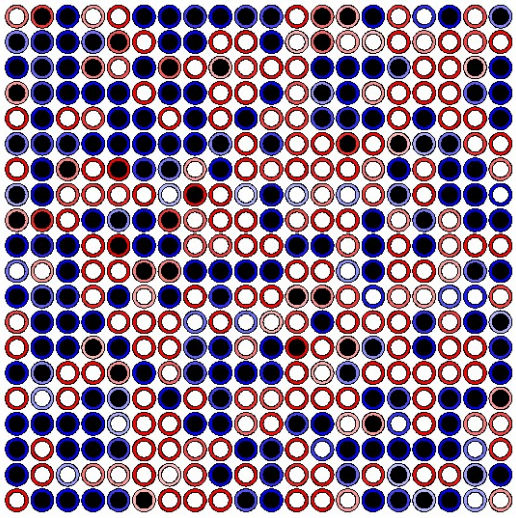
With this understanding, we can now represent the state of the network completely at a specific time. In a network of N, the state of the network is completely described the state vector $\vec{s}$, the activation vector $\vec{a}$, and the weight matrix W. The vector $\vec{s}$ is an N element vector that contains the Ising spin of every neuron in the network, $\vec{a}$ is an N element vector that contains the activation being received by every neuron in the network, and W is an N by N matrix whose ijth entry is the weight w_{ij}. Notice that our two rules $w_{ij} = w_{ji}$ and $w_{ii} = 0$ tell us immediately that W will be symmetric across the diagonal, and all of its diagonal entries will be 0. When defined in this way, we get the result

$$\vec{a} = W\vec{s},$$

making it easy to calculate the activation of every neuron with one line of code. We can visualize the state of the network as shown in figure 1 by representing the neurons as circles that are black when on and white when off. We can also add a border around the neurons representing the amount of activation that neuron is receiving. Positive activation is in blue, and negative is in red, with the intensity of the color representing the relative size of the activation. We can see from this image that the network is not fully stable is the sense that the activation of some neurons are of the opposite sign to the Ising spin of the neuron. When these neurons update, they will flip their Ising spin and the state of the network will be changed.

The last thing we need to complete our understanding of the network is a neuron update rule. There are two main categories of update rules: synchronous and asynchronous updating. A common synchronous update rule says that at each time t allow all neuron in the network to fire, however the standard Hopfield model uses an asynchronous update rule which states that at each time t we choose a neuron at random from the network and allow it to fire, i.e. update its Ising spin based in the activation it is receiving. Questions about the Hopfield network usually boil down to questions about how it evolves with time, i.e. we're interested in the behavior of $\vec{s}$ as t increases. The nature of the update rule makes the network's evolution very difficult to predict since at each successive time step it is impossible to predict which neuron will be updated, and at the same time, each update affects the activation vector and hence the result of all future updates. Obviously we're going to need a new conceptual framework to have any hope of understanding how the network might behave in general.

Figure 1: Visualization of a Hopfield network state when $N = 400$.



3 Energy of the Network

The key to understanding the evolution of the Hopfield network is a the network energy function H defined by

$$H(\vec{s}) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ji}s_i s_j.$$

The energy function takes in the network state as its input and outputs a single number called the energy of the network. This means we can assign a single (not necessarily unique) number to every possible state of the network. Most importantly, while the specific behavior of the network is unpredictable as it evolves, the behavior of the network energy is extremely predictable. This is because the formula for $H(\vec{s})$ ensures that a change Δs_j in the Ising spin of the jth neuron in the network leads to a change ΔH in the network energy of

$$\begin{aligned}\Delta H &= -\Delta s_j \sum_{i=1}^{N} w_{ji}s_i \\ &= -\Delta s_j a_j.\end{aligned}$$

Notice that Δs_j will be 0 when a neuron is updated but does not change its Ising spin, 2 when it flips from spin -1 to 1, and -2 when it flips from 1 to -1. A -1 to 1 flip only occurs when a_i is positive, and a 1 to -1 flip only occurs when a_i is negative. The network update rule thus ensures that $\Delta s_j a_j \geq 0$ and as a consequence $\Delta H \leq 0$, proving that H as a function of t is monotonically decreasing (i.e. never increasing).

Because there are finitely many possible states $\vec{s}$ that the network can occupy, there are also finitely many allowed values of $H(\vec{s})$, meaning that there is a minimum possible energy $\min H$ and a minimum possible magnitude of energy change $\min \Delta H$ associated with each update. Notice that the condition $f(a_i) = s_i$ if $a_i = 0$ tells us that $a_j = 0$ implies $\Delta s_j = 0$ i.e. there was no state change. Thus every state change in of the network corresponds to a decrease of at least $-\min \Delta H$ in the network energy. If we assume that the network is able to change its state indefinitely as it evolves then we know that the energy of the network after t updates will be some $H = -n \cdot \min \Delta H$ where we can choose n to be arbitrarily large. This contradicts the statement that a minimum energy $\min H$ exists. We have thus proven that the Hopfield network cannot evolve indefinitely, and that there must be some equilibrium states $\vec{s_e}$ corresponding to local minimums of the energy function which are stable upon network evolution.

The energy $H(\vec{s})$ of the Hopfield network can be thought of as a landscape sitting in $N + 1$ dimensional space–one for each Ising spin s_i and one for the value of H. If we start with a network of state $\vec{s}$, then we can think of its evolution as the movement of a point on this energy landscape

starting at $(\vec{s}, H(\vec{s}))$. The results we proved above tell us some important things. First, the energy landscape has at least a global minimum, and very possibly many other local minimums. We can think of these as divots in the energy landscape, because every state surrounding these minimums is of a higher energy. As the network evolves, it cannot move to a higher point on the energy landscape, so it can't climb up a hill in this energy landscape. As consequence, the network will eventually evolve to an equilibrium state corresponding to one of these divots in the energy landscape. If we already knew the energy minimum states $\vec{s_e}$ it seems likely that we would be able to predict fairly well what the long term behavior of the network would be upon evolution. We would expect that the network would usually evolve to the nearest state $\vec{s_e}$. However, we also have to take into account the size of the energy well surrounding each equilibrium state. Some might be wide and deep making them good attractors, while others might be tiny pockmarks in the energy landscape that are only able to attract networks whose state vectors lie very close by. Either way we can see that understanding the shape of this energy landscape is incredibly important in predicting how the network will evolve.

4 Hebbian Learning

The Hopfield network was originally introduced as a model of associative memory, the idea being that the tendency of the network to settle in the nearest equilibrium state upon evolution mirrors our ability to recall certain visual memories when shown an image similar to something we've seen before. In that sense, we think of the energy minimums on the energy landscape as memorized patterns or images where the size of the basin of attraction surrounding that pattern represents how well we remember it. However in order for the Hopfield network to be of any use as a means of storing and recalling memories, we must find some way of encoding the memories into the network in the first place. We can see from the definition of the energy function that it is parameterized by the weight matrix W–that is, the shape of the energy landscape and the locations of the energy minimums $\vec{s_e}$ are entirely determined by the values in W. So storing a set of memories in the Hopfield network means creating a weight matrix which ensures that each memory corresponds to some equilibrium state $\vec{s_e}$. Various learning rules have been devised o do this, the simplest of which is Hebb's learning rule.

Assume we want our network to memorize P patterns $\vec{p_1}, \vec{p_2}, \cdots, \vec{p_P}$, and denote the ith element in the μth pattern with the symbol $\epsilon_{\mu,i}$. Then Hebb's learning rule tells us to set the ijth weight in the matrix as

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^{P} \epsilon_{\mu,i} \epsilon_{\mu,j}.$$

Notice that $\epsilon_{\mu,i}\epsilon_{\mu,j}$ is 1 when $\epsilon_{\mu,i} = \epsilon_{\mu,j}$ and -1 when $\epsilon_{\mu,i} = \epsilon_{\mu,j}$. We can see from the activation equation that when the weight between two neurons is positive then they reinforce each-other and tend to want to be in the same state. If the weight between them is negative they tend to want to be in different states. The Hebb learning rule sets the weight between two neurons to be some measure of the average correlation between the state of the neurons as exhibited in the patterns. The fact that we divide by N rather than P after computing the sum is just to make a few of the upcoming computations more elegant.

Take for example the case when we are only trying to store one pattern $\vec{p}$. Then we have

$$\begin{aligned} w_{ij} &= \frac{1}{N} \epsilon_i \epsilon_j, \text{ and so} \\ a_j &= \sum_{i=1}^{N} w_{ij} s_i, \\ &= \epsilon_j \frac{1}{N} \sum_{i=1}^{N} \epsilon_i s_i. \end{aligned}$$

If we then set the network equal the pattern–$\vec{s} = \vec{p}$–we get

$$\begin{aligned} a_j &= \epsilon_j \frac{1}{N} \sum_{i=1}^{N} \epsilon_i^2 \\ &= \epsilon_j \frac{1}{N} \sum_{i=1}^{N} 1 \\ &= \epsilon_j. \end{aligned}$$

This means that the network activation vector is the same as its state vector when $\vec{s} = \vec{p}$. This derivation can be done in the general case of storing P patterns. The computation is virtually identical but with the added difficulty of juggling extra subscripts and an extra summation. The result is that when the network is storing P patterns and we set the network to be in the state of the mth pattern–$\vec{s} = \vec{p_m}$–we find that the activation a_j applied to the jth neuron is of the form

$$a_j = \epsilon_{m,j} + \sum_{\mu=1,\mu\neq m}^{P} \epsilon_{\mu,j} \frac{1}{N} \sum_{i=1}^{N} \epsilon_{\mu,i}\epsilon_{m,i}.$$

We now have a signal to noise problem because we want a_j to be as close as possible to the signal $\epsilon_{m,j}$, but the second term in the equation adds an element of noise. If the noise overpowers the signal, then there's a chance that our activation a_j will be the wrong sign, making the pattern $\vec{p_m}$ unstable. Let's examine the noise element in two parts. First, we look at the sum

$$S_1 = \frac{1}{N} \sum_{i=1}^{N} \epsilon_{\mu,i}\epsilon_{m,i}.$$

If we assume that the patterns are all high entropy (i.e. random looking) and not chosen to have high correlation to one another, then we can say that each term $\epsilon_{\mu,i}\epsilon_{m,i}$ in this sum is equally likely to be a 1 or -1. This means that we can treat each term $\epsilon_{\mu,i}\epsilon_{m,i}$ as a random variable with mean $\frac{1+(-1)}{2} = 0$ and variance $\frac{1^2+(-1)^2}{2} = 1$. In statistical terms, the random variables $\epsilon_{\mu,i}\epsilon_{m,i}$ are independent and identically distributed (i.i.d.) with mean 0 and variance 1. This property allows us to use the central limit theorem which says that the average value of a size n sample of i.i.d. variables converges to a normal distribution whose mean is that of the random variables, and whose variance is $1/n$ times that of the variables. Our sum S_1 is the average value of a size N sample of the i.i.d. variables $\epsilon_{\mu,i}\epsilon_{m,i}$, and thus for large N we can make the approximation

$$\begin{aligned} \operatorname{Var}(S_1) &= \frac{1}{N} \operatorname{Var}(\epsilon_{\mu,i}\epsilon_{m,i}) \\ &= \frac{1}{N}, \end{aligned}$$

and less surprisingly $\operatorname{Mean}(S_1) = 0$.

From here we can return to our full noise term:

$$S_2 = \sum_{\mu=1,\mu\neq m}^{P} \epsilon_{\mu,j} S_1.$$

We can treat S_1 and $\epsilon_{\mu,j}$ as independent (but very importantly not identically distributed) random variables each with mean 0 and variance $\frac{1}{N}$ and 1 respectively. The variance sum and product laws from statistics tells us that for independent random variables such as these the variance of the sum of the variables is equal to the sum of the variances of the variables, and likewise the variance of the product of the variables is the product of the variances of the variables:

$$\begin{aligned} \operatorname{Var}(X+Y) &= \operatorname{Var}(X) + \operatorname{Var}(Y), \\ \operatorname{Var}(XY) &= \operatorname{Var}(X) \operatorname{Var}(Y). \end{aligned}$$

Applying these rules to noise term gives us

$$\begin{aligned}\mathrm{Var}(S_2) &= \sum_{\mu=1,\mu\neq m}^{P} \mathrm{Var}(\epsilon_{\mu,j}S) \\ &= \sum_{\mu=1,\mu\neq m}^{P} \mathrm{Var}(\epsilon_{\mu,j})\,\mathrm{Var}(S) \\ &= \sum_{\mu=1,\mu\neq m}^{P} \frac{1}{N} \\ &= \frac{P-1}{N}.\end{aligned}$$

Thus the noise component of our activation formula is well approximated by a normal distribution with mean 0 and variance $\frac{P-1}{N}$. This means that we can think of the activation $a_j = \epsilon_{m,j} + S_2$ applied to the jth neuron in the network when $\vec{s} = \vec{p_m}$ as being randomly drawn from a normal distribution with mean $\epsilon_{m,j}$ and variance $\frac{P-1}{N} \approx \frac{P}{N}$ when P is large. When $\frac{P}{N}$ is relatively small compared to $|\epsilon_{m,j}| = 1$, then the normal distribution will be narrow enough to make it very unlikely that a_j will be chosen from it such that $f(a_j) \neq \epsilon_{m,j}$. However as $\frac{P}{N}$ increases, so to does the probability of this sort of bit error. This puts a clear limit on the storage capacity of the Hopfield network as trained with Hebb's learning rule. The value $\frac{P}{N}$ of a network is important enough that it is common to assign it its own variable $\alpha = \frac{P}{N}$. The exact value of α which marks the maximum storage capacity for the Hopfield network can change slightly depending on what bit error probability is considered to be acceptable for pattern storage, but the most commonly agreed upon value is $\alpha_c = 0.138$. A network of 1000 neurons for example would only be able to effectively store about 138 patterns in its weight matrix before significant errors in pattern recall and stability begin to appear.

5 Introducing Synaptic Noise

We have now introduced the parameter α which is one of the two most important parameters that characterize the behavior of the Hopfield network. The second of these, T, comes from extending the Hopfield model slightly to introduce an analogue of synaptic noise in the firing of biological neurons. The only change that needs to made to the model is to move to a probabilistic activation function defined by

$$f(a_i) = \begin{cases} 1 \text{ with probability} P(a_i), \\ -1 \text{ with probability } 1 - P(a_i) \end{cases}$$

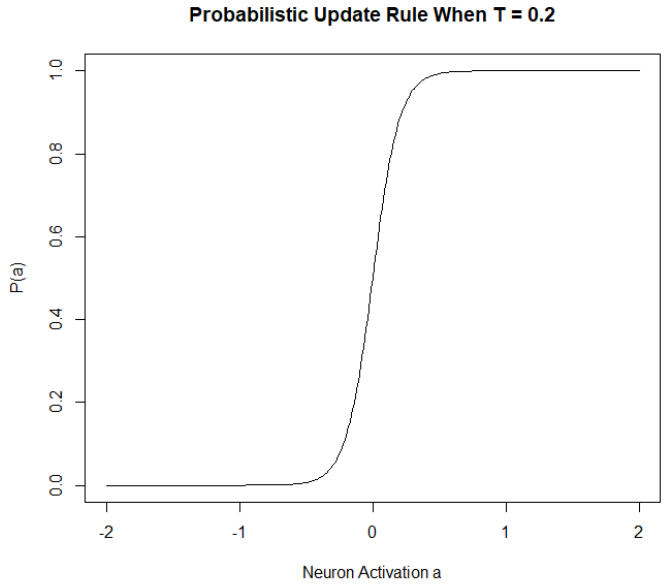
where $P(a_i)$ is the sigmoid shaped function

$$P(a_i) = \frac{1}{1 + e^{-2a_i/T}}.$$

Notice that when T is very small, the term $e^{-2a_i/T} \approx 0$ and as a consequence this activation function operates almost identically to the deterministic one. As the temperature parameter T of the network increases, the sigmoid shaped probability function begins to flatten out, and the network evolution becomes less predictable as the activation a neuron receives becomes less determinative of its update behavior. Figure 2 shows an the shape of $P(a_i)$ when $T = 0.2$. We can see that at this temperature, a neuron must be receiving an activation of with a magnitude close to 1 in order to ensure that it updates as expected.

The introduction of probability to the activation function changes the network behavior in a few important ways, all of which stem from the fact that the energy function $H(\vec{s})$ is no longer guaranteed to be monotonically monotonically decreasing upon network evolution. As T increases, the network will move around the energy landscape along more random paths, and will be increasingly likely to climb an energy gradient. This means that a network in state $\vec{s}$ might be virtually guaranteed to evolve towards some equilibrium state $\vec{s_e}$ when T is very small, but at larger values of T it might by chance climb the energy gradient surrounding $\vec{s_e}$ and fall into the energy basin of a different equilibrium state. Because the energy function is no longer guaranteed to monotonically decrease upon network

Figure 2: $P(a_i)$



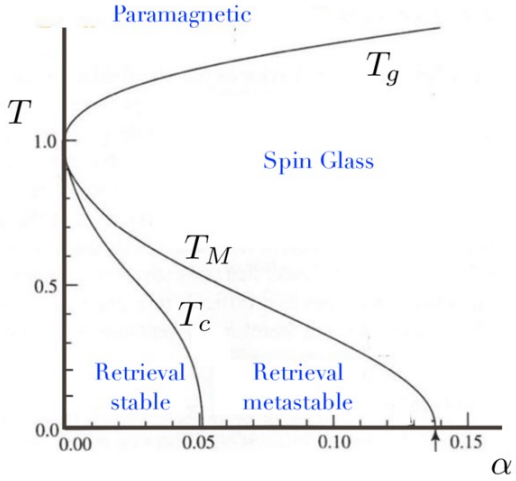
evolution, there is nothing stopping a network from moving around the energy landscape forever, never fully settling on an equilibrium state. Indeed there are no truly stable states now that the activation function is probabilistic. The probability of a state change in the network approaches 1 as t becomes arbitrarily large. For this reason we need to redefine our notion of stability to account for the certainty of thermal fluctuations in the network state.

We might reasonably make a distinction between states of the network which act as attractors during evolution and those that don't. A state $\vec{s_e}$ which becomes perfectly stable as T approaches 0 might be surrounded by an energy basin sufficiently wide and steep to ensure that for small enough values of T, a network within the basin will be likely to stay within the basin upon evolution, and hence its evolution will take to form of small fluctuations about $\vec{s_e}$. The average size of these fluctuations will tend to increase as T increases. When the fluctuations become large enough to reach above the walls of the basin, the network will be able to escape and move to different regions of the energy landscape. We might call a state stable in this sense if it is the locus of fluctuation of the network as it evolves. This definition is clearly less precise than the notion of complete stability, and it lends itself immediately to the idea of levels of stability. A state that tends to have small and infrequent fluctuations about it seems like it should be considered more stable than a state which acts as the locus of frequent and large fluctuations that stay within its energy basin. We will save a more in depth discussion of the notion of stability at high temperature for a later section. For now we only need understand that state stability is still an important phenomenon in the temperature Hopfield model.

6 Naive Recreation of the Phase Diagram

Perhaps the most important result relating to the Hopfield network is that it is solvable. In a series of papers published from 1985 to 1987, Amit et al. showed how mathematical methods developed in the study of magnets and other complex systems in condensed matter physics could be applied to the Hopfield model to make robust predictions about the memory recall capabilities of the network. The series of papers culminated in the analytical derivation of the phase diagram for the Hopfield network. A representation of the diagram is shown in figure 3. The diagram divides the (α, T) plane into four regions. The region bounded below the line T_C is labeled as "retrieval stable" indicating that the Hopfield network will function properly in retrieving stored memories when it is initialized with (α, T) in that region. Similarly, the region between T_M and T_C is labeled "retrieval metastable" indicating that when the network is initialized with (α, T) in that region, it will be able to retrieve the majority of its stored memories. The two other regions labeled "Spin Glass" and "Paramagnetic" represent

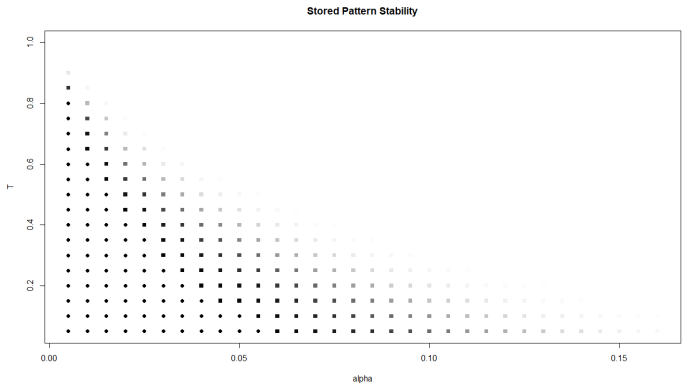
Figure 3: Hopfield Network Phase Diagram



the areas of the phase space which correspond to amnesic networks. Spin glass and paramagnetic are both terms ported over from the phase diagrams studied in condensed matter physics. In the spin glass region, the network still may have stable states, but they are "spurious states" rather than being the stored patterns. However, in the paramagnetic region above the line T_g the size of fluctuations about the energy minimum states become so large that no energy basin can reliably contain them, and as a consequence there are no stable points whatsoever in the state-space.

A version of this phase diagram can be recreated through simulation by explicitly testing the pattern storage capabilities of networks initialized at various points on the phase space. The results of such a simulation are displayed in figure 4, where the darkness of the plotted points is a function of the average proportion of stored patterns which are stable in networks initialized at that point on the phase space. Circular points correspond to regions where an average of at least 99 percent of stored patterns are stable. For the purposes of creating this diagram, a pattern $\vec{p}$ was considered stable if when $\vec{s} = \vec{p}$, every neuron had at least a 90 percent chance of preserving its state upon firing. The average stability of stored patterns at each point on the phase space was calculated by initializing and training 100 networks ($N = 500$) with randomized pattern sets at each point, categorizing the stability of up to 20 patterns for each of those 100 networks, and finding the proportion of these patterns that were stable. This naive recreation is surprisingly effective in approximating the retrieval

Figure 4: Phase Diagram Recreation



regions shown in the analytically derived phase diagram. The similarity of these figures suggests that perhaps the unjustified assumption that a network state $\vec{s_e}$ will be stable if every neuron has at least a 90 percent chance of preserving its state upon firing when $\vec{s} = \vec{s_e}$ is a good approximation of the truth. More compactly, we can define of a third vector $\vec{c}$ in addition to $\vec{s}$ and $\vec{a}$ whose ith entry is the probability that the ith neuron in $\vec{s}$ will preserve its state if chosen to fire. We can call $\vec{c}$ the stay-chance vector. In creating the figure 4 diagram, we implicitly put forward the hypothesis that

a network state $\vec{s_e}$ will be stable upon evolution if the minimum value of the stay chance vector $\vec{c}$ is greater than 0.9 when $\vec{s} = \vec{s_e}$.

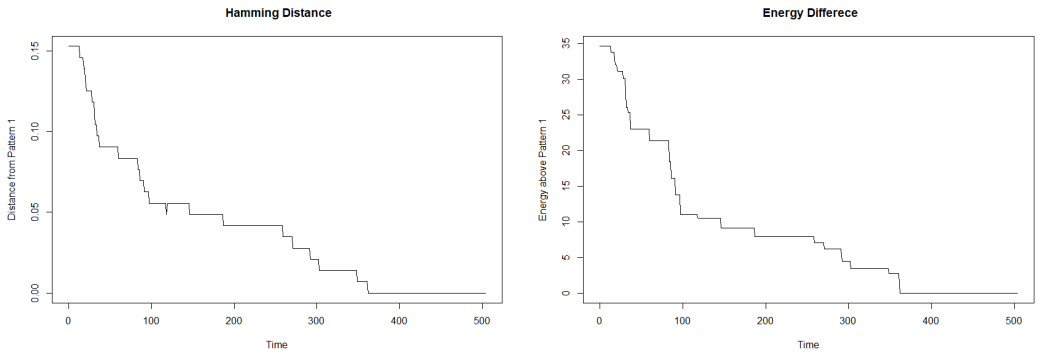
The veracity of this claim is not immediately clear. While we might find it useful to visualise the energy landscape of the Hopfield network as a smooth 2 dimensional surface in R^3, it is important to remember that this is just a heuristic. The actual energy landscape is quantized due to the fact that there are only finitely many possible network states, it has N degrees of freedom, and there's no guarantee that it is even remotely well approximated by a smooth function. There is nothing in the structure of the network that prevents shortcuts out of energy basins requiring only a few lucky updates to move to a wildly different point on the energy landscape. One might imagine therefore, that a single data point like the minimum value of $\vec{c}$ would be insufficient to capture the complexities in the structure of the energy basin surrounding a state–complexities which might have a significant affect on the state's long term stability. The question then naturally arises as to what information about a network state is predictive of its long term stability. Is it possible to construct a function that can take in the state of a network and reliably output information about its level of stability?

7 Directly Measuring State Stability

Before trying to define a function that predicts state stability, we would do well provide a more precise definition of what state stability is in the context of a positive temperature Hopfield model. Earlier, we gave the intuitive definition that a state is stable if it is the locus of fluctuation of the network as it evolves. In order to make this more precise and to further develop our intuition about network evolution, we need to look at some examples of what this evolution looks like. We can do this by creating graphs representing the state of the network as it evolves using two metrics–energy change, and hamming distance change.

We will create two graphs, one that shows the change in the energy as it evolves, and one that shows the hamming distance of the network form its original state. The hamming distance between two binary vectors is number of bits that are different between the two. For the purpose creating these graphs, we have represented hamming distance instead as a proportion of bits different, i.e. neuron states different between two vectors. Figure 5 shows an example of pattern retrieval at low temperature as visualized in hamming distance and energy change. The network started at a hamming

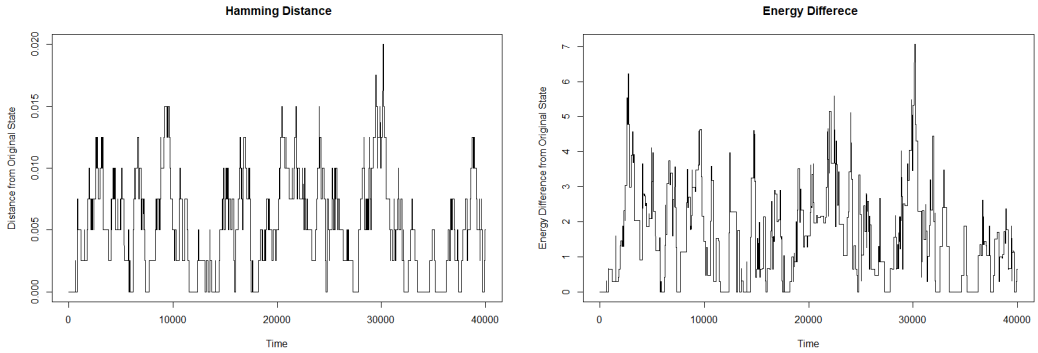
Figure 5: Memory Retrieval of Pattern 1



distance of 15 percent from the stored pattern and eventually evolved to be the stored pattern and stayed there. The energy graph shows that the network evolution was a continuous descent down an energy gradient in the energy landscape and confirms that the stored pattern is indeed a local energy minimum. These graphs are incredibly useful to us in that they represent the N dimensional path taken by the network along the energy landscape in a visualizable form.

Figure 6 shows typical behavior of an equilibrium state upon evolution at high temperatures. We can see in these graphs the state fluctuations discussed earlier. The Hamming distance graph gives us a good idea of how close to the original pattern the network tends to stay. In this example, the network state never wanders more than 2 percent in hamming distance from the original state. The energy fluctuation graph gives a good idea of how many different states the network visits over the course of its evolution since it's very unlikely that any two states share the same energy value. More importantly, it can also show us whether the original state is in fact a local energy minimum in the

Figure 6: High Temperature Stability



region it inhabits. In the case of the figure 6 original state, we can see that the network never found a lower point on the energy landscape that of its original state. Overall, the two graphs tell us that the original state of the network is the energy minimum of a basin of attraction deep enough to contain the network upon evolution. For this reason, it is reasonable to call it a stable state.

Looking at these graphs directly can give us an intuitive feel for the behavior of the network state upon evolution, but in order to quickly compare the stability of various states, we need to move to a quantitative rather that a qualitative measure of stability. The exact method we use to quantify stability in this context is necessarily arbitrary. The only goal is to create a stability metric which maps well onto our intuitive notion of stability. Of course our metric must assign a network state that is perfectly stable upon evolution to the value of 1, and it must assign a network state that immediately escapes the energy basin surrounding it a value of 0. However, the way in which it weights other aspects of the state's behavior upon evolution like the size and frequency of fluctuations from that state is entirely up to us. We might all start out with a base intuition about what stable network behavior looks like, but in order to strengthen these intuitions and begin to understand their practical upshot, it's necessary to look at hundreds of graphs depicting network evolution.

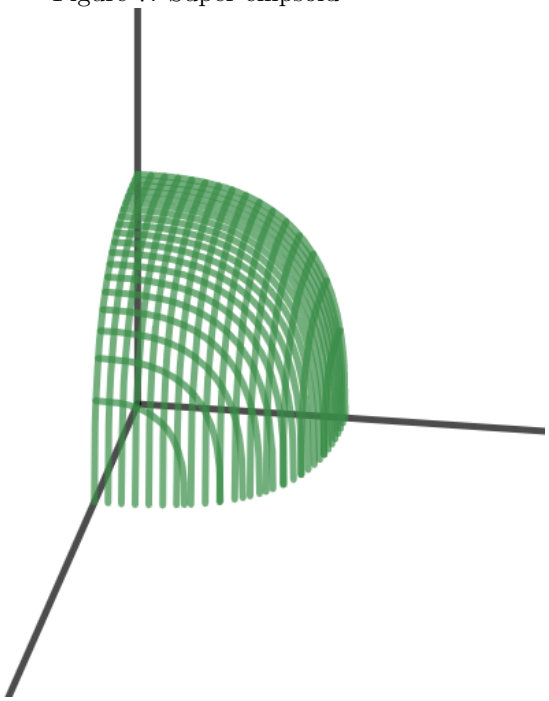
Having done this, I have developed a metric of network stability which is a function of four variables drawn from hamming distance and energy data representing long term state evolution. These variables are the average hamming distance from the original state as the network evolves, the average size of fluctuations (i.e. the maximum hamming distance they reach away from the state), the average proportion of time that the network spends fluctuating as opposed to occupying the original state, and the maximum duration of the fluctuations. The hamming distance data tells whether the network remains trapped in an energy basin surrounding the state, but it can't tell us whether or not the original state is in fact the locus of fluctuation. This will only be true when the original state is in fact the energy minimum of the basin it occupies, indicated by the fact that the energy of the network upon evolution never passes below that of the original state. Our stability metric will therefore be a function of four numeric variables and one Boolean variable indicating whether or not the state is a local energy minimum. We want this function in R^5 to have a value of 1 at the origin, a plateau around the origin ensuring that networks with some small fluctuations are still considered very stable, and an exponential drop-off to 0 value if the input strays too far from the origin along any of its 4 axes. One function in R^3 that has these basic properties is the super-ellipsoid function derived form the equation

$$1 = \left(\frac{x}{a}\right)^n + \left(\frac{y}{b}\right)^n + \left(\frac{z}{c}\right)^n.$$

The super-ellipsoid is a generalization of the unit sphere that adds the ability to set distinct boundaries along each of the three axes rather than a simple radius, and the ability to control curvature. Figure 7 shows a section of a super ellipse in the first quadrant with $n > 2$. We can see the similarities to a section form a regular ellipsoid, but notice that it is more flat around the origin and has a steeper drop-off. In fact, as n approaches infinity, the shape of the super ellipsoid closer and closer approximates that of a rectangular prism. Lucky for us, the equation for the super-ellipsoid in R^3 easily extents to R^5 like so:

$$1 = \left(\frac{x_1}{a_1}\right)^n + \left(\frac{x_2}{a_2}\right)^n + \left(\frac{x_3}{a_3}\right)^n + \left(\frac{x_4}{a_4}\right)^n + \left(\frac{x_5}{a_5}\right)^n.$$

Figure 7: Super-ellipsoid



Using this equation to form our stability measure allows us to exactly control the cutoffs for each of our 5 input variables and the size of the plateau around the origin.

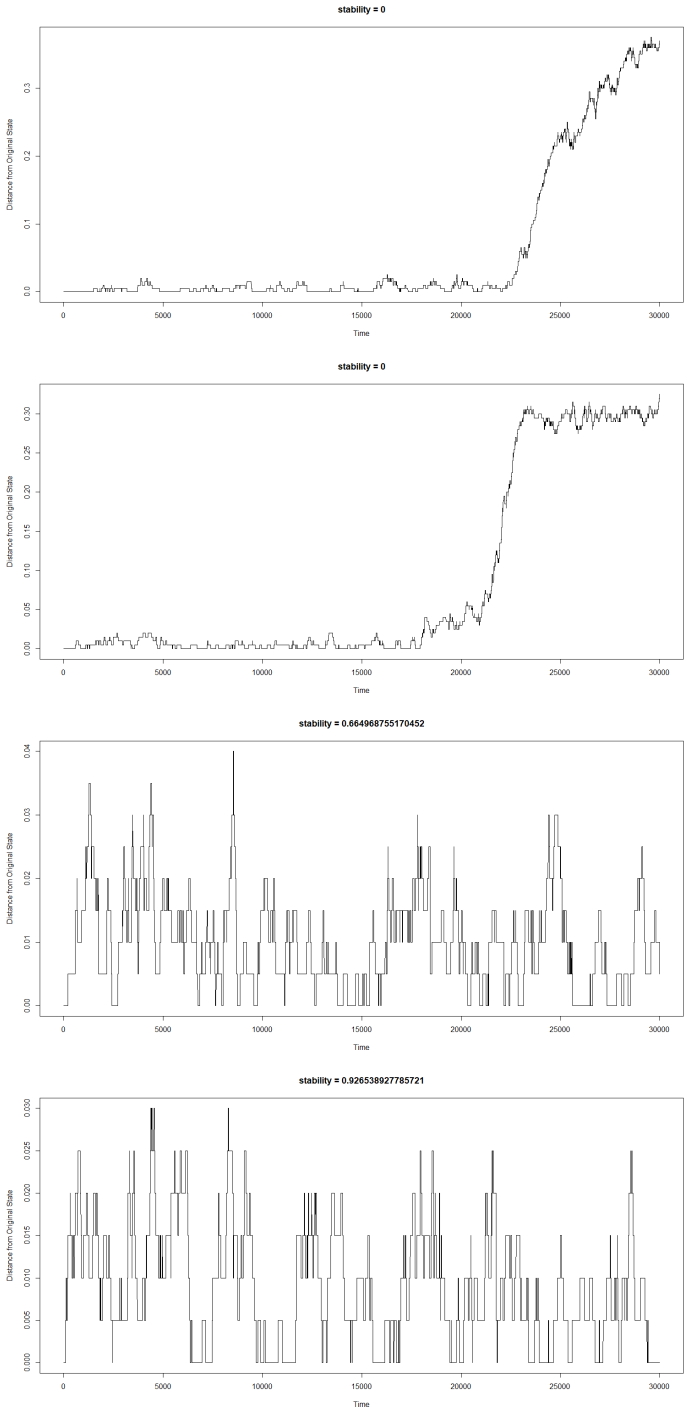
We find the proper cutoffs and curvature that conform to our intuition about network stability through trial and error. For the purposes of this paper I have been conservative in my definition of stability, choosing cutoffs that ensure that as the network evolves, it must stay within 1 percent hamming distance of its original state and spend at least 4 percent of its evolution time in this state to be assigned non-zero stability. I also assigned a penalty of 25 percent for states which appear pseudo-stable but which are not local energy minimums. Figure 8 shows the stability assigned to various network states above the graph of their evolution in hamming distance. We can see that the stability metric does well at detecting network states that are clearly unstable as well as assigning reasonable stability values to more ambiguous states. Note that throughout this paper we have been using $150N$ updates as a representative of long term network behavior. This number was arrived a partially for its convenience, but also because it my the condition that every neuron in the network is almost guaranteed to have fired at least 50 times is this time frame.

The computation for this sort of probability is surprisingly difficult. To make it more understandable, we can start by attacking a few simpler problems. To begin, we ask what is the probability that a particular neuron has been chosen to fire at least once after U updates. This can be found by finding the probability that the neuron is not chosen to fire for U consecutive updates and subtracting it from 1. The probability that a neuron is not chosen to fire in each update is $1 - \frac{1}{N}$, so we have

$$p_1(U) = 1 - \left(1 - \left(1 - \frac{1}{N}\right)^U\right)$$

A more difficult problem is finding the probability that every neuron has been chosen to fire it least once after U updates. A naive approach might simply take the expression $p_1(U)$ to the power N. This multiplies together the n probabilities that each individual neuron has fired at least once after U steps. This is the correct approach, but simply writing $p_1(U)^N$ ignores the fact that these N events are not independent. For example, if in the first 99 updates, we know that 99 neurons is a 100 neuron network have already fired, then the chance that the final neuron fires in the first 100 network updates is $\frac{1}{100} = 1\%$ rather than $1 - \left(1 - \left(1 - \frac{1}{100}\right)^{100}\right) \approx 37\%$. In order to find the probability that every neuron has been chosen, we multiply together the probability that the first has been chosen with the probability that the second has been chosen assuming that the first has been chosen and so on. The probability that a particular neuron has been chosen after U updates given that i other neurons were

Figure 8: Test of Stability Metric



also chosen is just $p_1(U-i)$, because each other neuron takes away one update from consideration. This means that our final answer for the probability to that every neuron has been chosen to fire at least once after U updates is given by

$$P_1(U) = \prod_{i=0}^{N-1} p_1(U-i).$$

We can use a similar procedure to find the probability that every neuron has fired at least k times after U updates. First we find an expression $p_k(U)$ for a particular neuron in the same way as above. The probability that a particular neuron has not fired at least k times after U updates is the probability that it has fired $k-1$ times plus the probability that it has fired $k-2$ times and so on.

The probability that a neuron has fired exactly k times after U updates is given by the expression

$$\binom{U}{k}\left(\frac{1}{N}\right)^k\left(1-\frac{1}{N}\right)^{U-k},$$

where $\binom{U}{k}$ is the number of ways to choose k elements from a set of U things. Combining these two facts gives us the expression

$$p_k(U) = 1 - \sum_{i=0}^{k}\binom{U}{i}\left(\frac{1}{N}\right)^i\left(1-\frac{1}{N}\right)^{U-i}.$$

From here the procedure to get to the to an expression for $P_k(U)$ is the same as before:

$$P_k(U) = \prod_{i=0}^{N-1} p_k(U-ki).$$

This is monstrously complex, making it impossible to analytically find an inverse for, but just limiting ourselves to calculating the function values directly, we can still compute that for a 1000 neuron network, the probability that every neuron has been chosen to fire at least 50 times after $150N = 150,000$ updates is 99.99991%. This probability remains relatively stable as N changes, justifying our choice of $150N$ as a representative evolution timescale.

8 Predicting Network Stability

Now that we have a much more robust definition of stability at nonzero temperature, we can address the question of how to predict state stability. We already implicitly hypothesized that state stability could be well approximated by a step function that input the minimum value of $\vec{c}$ and output a 1 if $\min \vec{c} > 0.9$ and a 0 otherwise. We can test this hypothesis by creating a data set with a column containing state stability data and one containing corresponding minimum stay chance data. We can then fit this data to a sigmoid shaped function that predicts stability based on $\min \vec{c}$:

$$S(\min \vec{c}) = \frac{1}{1+e^{-B(\min \vec{c}-A)}}.$$

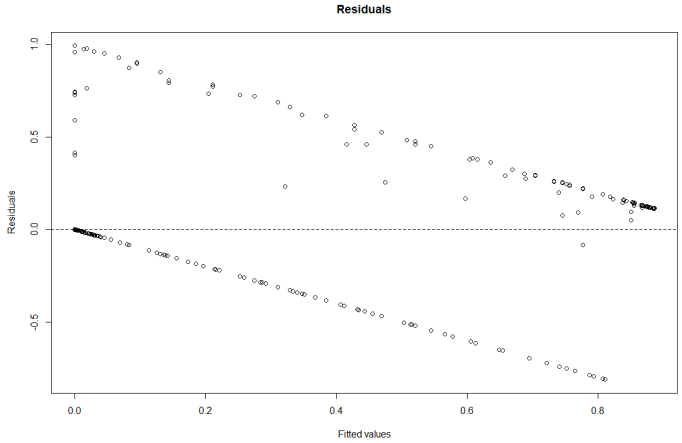
We apply a nonlinear regression algorithm to our 510 observation data set and find that the above expression best fits the data when $A \approx 0.93$ and $B \approx 30.33$. The fact that B is high tells us that the function is very nearly a step function, which supports our hypothesis, but the A value tells us that the appropriate cutoff for stability is closer to $\min \vec{c} > 0.93$ rather than 0.9. However, the most important value form this nonlinear regression algorithm is the residual standard error of 0.2868. This tells us the standard deviation of the model residuals, and considering that the nature of the data ensures that a residual can be at most 1, the number 0.2868 indicates that the model is a very poor fit. This becomes even more clear when looking at the graph of the model residuals shown in figure 9. We can see that the bulk of the 510 observations are clustered around predicted values of stability of 1 or stability of 0 where the model preforms relatively well. However the two downward sloping lines of residuals tell us that the model is fundamentally ill equipped to deal with the complexity of the classification problem it was fitted for. This finding somewhat undermines our previous work in recreating the phase diagram by showing us that the predictor of stability we used to create the graph was inaccurate.

In order to create a more accurate graph of network stability as a function of α and T we have two main options. First, we could collect data about pattern stability at each point on the phase space by directly testing the long term stability of each pattern in each network e initialize there. This is a nonstarter on the grounds that it would require tremendous computing power to preform these test at across the whole phase space. Or second, we could try to find a function that predicts network stability with significantly better accuracy than does our naive step function.

The obvious first step to doing this is to add more information as input to our predictive function. We can again fit a logistic function to our data, but this time one of the form

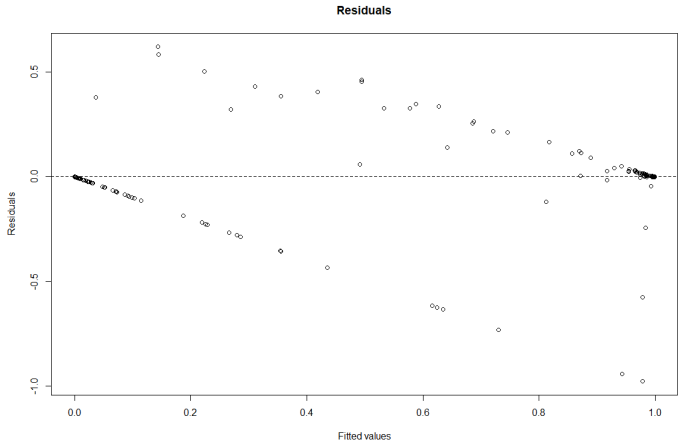
$$S(x_1,x_2,x_3,x_4,x_5,x_6) = \left(1+e^{-(Ax_1+Bx_2+Dx_3+Ex_4+Fx_5+Gx_6)}\right)^{-1},$$

Figure 9: Residuals for min $\vec{c}$ Logistic Function Model



where the six x_i input variables are the minimum and maximum values in $\vec{c}$, the first and second quartiles of the values in $\vec{c}$, and the first and second quartiles of the lowest 10 percent of the values in $\vec{c}$. Note that the median of the values in $\vec{c}$ and the and the median of the lowest 10 percent of the values in $\vec{c}$ were originally included in this function but added little predictive power to the model and so were removed in order to prevent overfitting of the data. Preforming nonlinear regression with this function gave us a model with a somewhat better residual standard error of 0.1262. The improvement can be seen in the model's residual plot in figure 10. Overall though, this is still a relatively minor

Figure 10: Residuals for Full Logistic Function Model



improvement in the predictive power of our model.

Here is where we reach the limitations of simple nonlinear modelling techniques, as adding many more parameters to our function would begin to significantly increase the chances of overfitting the data. This could be remedied by creating a much larger data set, but this would be prohibitively computationally intensive. It looks like for now the best we can get is a model with residual standard error of 0.1262–not exactly the value one might hope for. It looks like a high accuracy recreation of Amit et al.'s phase diagram is going to have to wait for a better model or a researcher with better computing resources.

9 Conclusion

The goal of this paper has been primarily to help develop an understanding of the inner workings of the Hopfield network model of associative memory through mathematical analysis and directly running the network evolution process. We discussed three primary questions about the network, each deeply related to each other. The first question was asking when the network would function as

a memory model and when it would be untrainable. To answer this we turned to the work of Amit et al. and made an attempt at recreating the network phase diagram by measuring the state stability of stored patterns in the network. The difficulty of this task led us to take on the two other main questions of this paper–how should one define the notion of stable states when network evolution becomes probabilistic, and how can we predict the long term stability of a network state based only upon its instantaneous description? The ultimate goal in in addressing these other two questions was to better understand the phase diagram of the Hopfield network. In regards to defining stability at high temperatures we have been reasonably successful if slightly arbitrary, but unfortunately we have been unable to use the methods of analysis available to us to find a means of accurately predicting the long term stability of a network state based purely upon its superficial characteristics. Finding function that could preform this prediction would be allow for the creation of a much more accurate phase diagram of the Hopfield network as well as being an achievement in its own right. There is more research to be done in this area seeing as we only explored a single method of multidimensional function fitting, neglecting the use of more computationally intensive methods like high parameter modeling and neural network training. Regardless of how successful we've been in answering these three questions, its important to remember that 15 page paper can only begin to scratch the surface of the wealth of mathematical analysis that has been done and is still to be done in attempting to understand the Hopfield network.

References

[1] Amit, Daniel J., et al. "Spin-Glass Models of Neural Networks." Physical Review A, vol. 32, no. 2, 1985, pp. 1007–1018., doi:10.1103/physreva.32.1007.

[2] Amit, Daniel J., et al. "Storing Infinite Numbers of Patterns in a Spin-Glass Model of Neural Networks." Physical Review Letters, vol. 55, no. 14, 1985, pp. 1530–1533., doi:10.1103/physrevlett.55.1530.

[3] Amit, Daniel J., et al. "Information Storage in Neural Networks with Low Levels of Activity." Physical Review A, vol. 35, no. 5, 1987, pp. 2293–2303., doi:10.1103/physreva.35.2293.

[4] Hopfield, J. J. "Neural Networks and Physical Systems with Emergent Collective Computational Abilities." Proceedings of the National Academy of Sciences, vol. 79, no. 8, 1982, pp. 2554–2558., doi:10.1073/pnas.79.8.2554.

[5] Mézard, Marc. "Mean-Field Message-Passing Equations in the Hopfield Model and Its Generalizations." Physical Review E, vol. 95, no. 2, 2017, doi:10.1103/physreve.95.022117.

[6] Wang, Ruye. "Hopfield Network." Hopfield Network, 10 Apr. 2018, fourier.eng.hmc.edu/e161/lectures/nn/node5.html.

[7] Weisstein, Eric. "Central Limit Theorem." From Wolfram MathWorld, mathworld.wolfram.com/CentralLimitTheorem.html.

[8] Weisstein, Eric. "Variance." From Wolfram MathWorld, mathworld.wolfram.com/Variance.html.