Progress Report
September 14 to October 3, 2010

Raj K. Kumar, Amir Hesam Salavati
E-mail: raj.kumar@epfl.ch, hesam.salavati@epfl.ch

Supervisor: Prof. Amin Shokrollahi
E-mail: amin.shokrollahi@epfl.ch

Algorithmics Laboratory (ALGO)
Ecole Polytechnique Federale de Lausanne (EPFL)

October 4, 2010

1 Introduction

Hopfield networks are artificial neural networks that mimic associative memory mechanism in brain, i.e. they memorize a number of patterns during the training phase and then recall the closest pattern to any input given to the network. In that regard, Hopfield networks are kind of like an error correcting decoder.

In this report, we try to analyze the behavior of Hopfield networks using density evolution techniques. We consider two scenarios: in the first we assume the original Hopfield framework and just select the patterns from codewords of an error correcting code (instead of completely at random). We would like to see if we can increase the storage capacity of Hopfield networks, which is at best proportional to the number of nodes in its current format. In the second scenario, we consider a different weight distribution than the original Hopfield model and use density evolution to investigate how the network and probability of error evolve in this situation.

The principles of Hopfield networks are briefly discussed in the next section. Section 4 addresses the case of enhanced Hopfield networks, i.e. the network in which patterns are selected from patterns with relatively large minimum distance. Biologically-inspired weight distributions and their effects on Hopfield networks are considered in section 5. Finally, section 6 concludes this reports and identifies future steps.

2 Brief Overview of Hopfield Networks

A Hopfield network is a *complete* graph with n nodes (neurons) in which links are weighted and each node can have a binary state (± 1) [1]. If weights are carefully chosen, Hopfield networks are able to "memorize" a number of patterns with length n. Here, memorizing means if we do a training phase, memorized patterns are stable states of the network, i.e. if we feed them as inputs the network does not evolve. Furthermore, in certain types of Hopfield networks, we can achieve some degree of error correction. In these cases, the network converges to the closest memorize pattern (stable state) for a given input. We will get back to these types later on.

In traditional Hopfield networks, if we denote the state of node k by x_k,

the weights between nodes i and j, w_{ij}, is determined as follows:

$$w_{ij} = \frac{1}{n} \sum_{m=1}^{M} x_i^m x_j^m \tag{1}$$

where x_i^m is the i^{th} bit, i.e. state of the i^{th} neuron, for the m^{th} memorized pattern and M is the total number of such patterns.

Given the weights, neurons update their state according to equation (2). In words, each neuron calculates the weighted sum over its input links and if the sum was larger than a threshold θ (which can vary over time in general), neuron fires, i.e. its state is changed to $+1$, and remains silent otherwise.

$$x_i = \begin{cases} 1, & \sum_{j=1}^{n} w_{ij} x_j > \theta \\ -1, & \text{Otherwise} \end{cases} \tag{2}$$

3 Capacity of Hopfield Networks

In original Hopfield networks, patterns are selected *randomly* and the goal is to memorize as many such random patterns as possible so that correct recall is guaranteed (with probability 1). In other words, we are interested in maximizing M such that we are able to correctly recall all M random patterns with probability one.

It is shown that in this setup, Hopfield networks are able to memorize at most $M_{max} \propto n/\log(n)$ patterns (with error correction capability of less than or equal to $n/2$ errors in the input pattern) [4]. Without error correction requirement, the maximum number of patterns is $M_{max} \simeq 0.14n$.

Comparing the above values with the number of codewords decoder with n codebits can "memorize", i.e. $2^k = 2^{rn}$, we see that original Hopfield networks are totally inefficient from storage point of view. However, this inefficiency is not a result of the network structure but mostly because of the assumption that the memorized patterns are chosen completely at random. At this point, an interesting issue to consider is to investigate the gain we get in terms of storage capacity if we choose the memorized patterns not completely at random, but carefully such that they have a relatively big minimum distance. For instance, we could select the codewords of an LDGM code as the set of patterns to be memorized. This is similar to what Berrou et al. has done in [2], in which they consider the patterns to be Walsh-Hadamard codewrods. The question would then be how many such patterns

we can safely store without causing recall errors? In the following sections, we try to take the first steps in answering this question.

Another interesting subject is to investigate dynamical behavior Hopfield networks when the weights are not determined according to equation (1) but drawn randomly from a given ensemble. What is interesting is to see how network evolves for a set of biologically-meaningful ensembles. In particular, we would like to see if network can achieve certain degree of error correction in these cases. We will partly address this issue in a later section.

4 Hopfield Networks with Larger Minimum Distance

In this section, we consider a Hopfield networks in which weights are determined according to equation (1) and patterns are drawn from an LDGM code with node degree distribution $(\Lambda(x), \Omega(x))$, i.e. Λ_i is the number of left (message) nodes with degree i and Ω_j is the number of right (codeword) nodes with degree j.

The goal is now to see how many such patterns we can safely store with guaranteed correct recall. There are two different issues that we should consider. The first is that all codewords must be stable states of the network, i.e. if we give one of them as input, the network should not evolve to another state. The next issue is if the proposed method can achieve error correction as well, i.e. if we provide an erroneous input, can the network converge to the closest stable state?

To address the above issues, we have to investigate the probability that a neuron fires in any iteration. Let $z(r)$ denote the probability that a generic neuron (say neuron i) fires at round r. In other words:

$$z(r) = \Pr\{x_i = 1 \text{ in round r}\} = \Pr\{\sum_{j=1}^{n} w_{ij}x_j > \theta, \text{ in round r}\} \qquad (3)$$

Our approach is based on analyzing $z(r)$ as given by equation (3) to see how it is related to pattern properties. By conditioning on the number of neurons that fired in previous rounds and rewriting equation (3) we obtain a recursive expression for $z(r)$ as follows:

$$z(r) = \Pr\{\sum_{j=1}^{n} w_{ij}x_j > \theta, \text{ in round r}\}$$

$$= \sum_{k=0}^{n} \Pr\{\sum_{j=1}^{n} w_{ij}x_j > \theta, \text{ in round r|k neurons fired in round (r-1)}\}$$
$$\times \Pr\{\text{k neurons fired in round r-1}\}$$
$$= \sum_{k=0}^{n} \binom{n}{k} (z(r-1))^k (1-z(r-1))^{n-k} \Pr\{\sum_{j=1}^{k} w_{ij} - \sum_{j=k+1}^{n} w_{ij} > \theta\} \quad (4)$$

In which for simplicity we have assumed that neurons $1, ..., k$ fired in the previous round.

To calculate $\Pr\{\sum_{j=1}^{k} w_{ij} - \sum_{j=k+1}^{n} w_{ij} > \theta\}$ in equation (4), we must obtain the probability distribution for the sums of form $\sum_{j=1}^{k} w_{ij}$. If w_{ij}'s are i.i.d. Gaussian random variables, then the sum would also be Gaussian. Although it seems that Gaussian assumption is not that unrealistic from a biological point of view[1], it is not necessarily valid in our case. Hence, we must either find the probability distribution of w_{ij}'s or approximate the sum using central limit theorem in which case we need the mean and variance of w_{ij}.

Determining the probability distribution of w_{ij} is somehow straightforward if we assume different codewords to be independent of each other. Then, all we need to do is to compute the distribution of $x_i^m x_j^m$ and do convolution M times. As it is shown in section A, the probability distribution of $x_i^m x_j^m$ is close to uniform.

However, computing the distribution of w_{ij} is a cumbersome task. Instead, we use central limit theorem to estimate it as a Gaussian random variable with mean μ and variance σ^2. In appendix A, we provide a way to obtain μ and σ^2 from code properties.

Having assumed that w_{ij}'s are i.i.d. $N(\mu, \sigma^2)$ random variables, equation (4) simplifies to:

$$z(r) = \sum_{k=0}^{n} \binom{n}{k} z(r-1)^k (1-z(r-1))^{n-k} \Pr\{\sum_{j=1}^{k} w_{ij} - \sum_{j=k+1}^{n} w_{ij} > \theta\}$$
$$= \sum_{k=0}^{n} \binom{n}{k} z(r-1)^k (1-z(r-1))^{n-k} Q\left(\frac{\theta + (n-2k)\mu}{\sqrt{k}\sigma}\right) \quad (5)$$

In which Q is the well-known Q-function.

[1]We will address this case in the next section

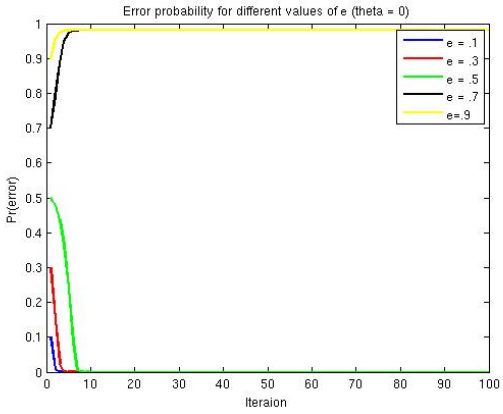

Figure 1: Evolution of error probability for different values of ϵ and $\theta = 0$

4.1 Numerical analysis

Equation (5) provides us with a means to analyze dynamical behavior of Hopfield networks. For instance, if we assume the all -1 codeword was given to the network, the probability of error at round r equals to $z(r)$. If the input pattern does not contain any errors, i.e. $z(0) = 0$, then $z(r) = 0, \forall r > 0$. Hence, the network is able to correctly recall this pattern if there are no errors. On the other hand, if we assume a fraction ϵ of input bits are erroneous, then we have $z(0) = \epsilon$ and we can track $z(r)$ recursively. Ideally, we would like to see $z(r)$ vanishes for reasonable values of ϵ. This is what we have done in this section.

In figures below, you will find evolution of $z(r)$ over iterations assuming different values of ϵ and θ. It is clear from figures that for a given value of θ, $z(r)$ converges to zero if ϵ is small enough. In other words, similar to the threshold we have in decoding algorith above which the algorithm diverges, there is critical calue here as well which determines if the algorithm converges. In our analysis, we have assume the code rate to be $1/2$, i.e. $M = 2^{n/2}$. Furthermore, we have assumed $\mu = 0.1M/n$ and $\sigma^2 = M/n^2 - \mu^2$ (see appendix A).

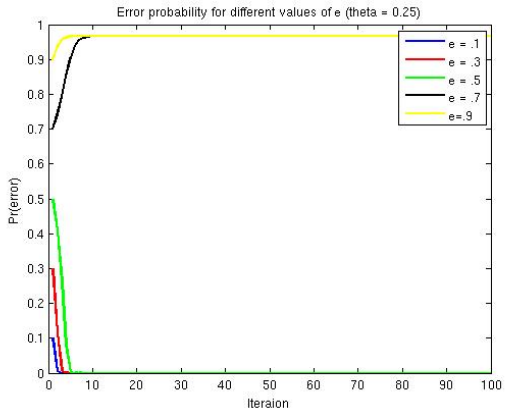

Figure 2: Evolution of error probability for different values of ϵ and $\theta = 0.25$

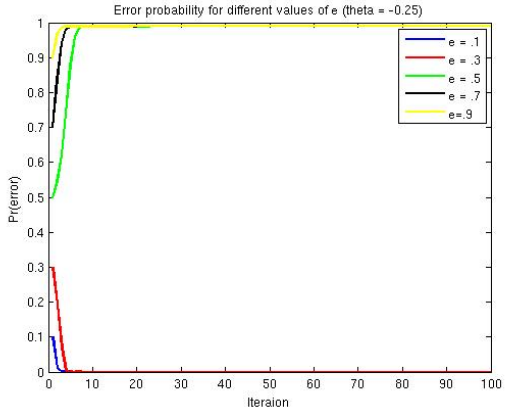

Figure 3: Evolution of error probability for different values of ϵ and $\theta = -0.25$

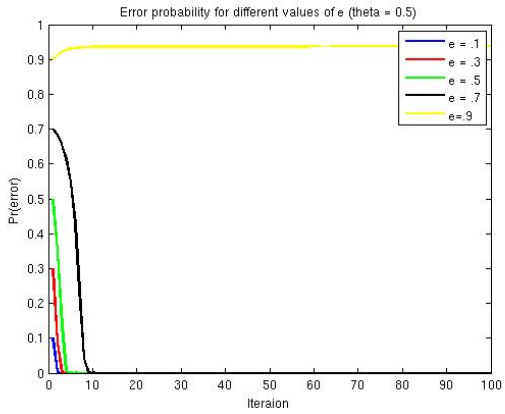

Figure 4: Evolution of error probability for different values of ϵ and $\theta = 0.5$

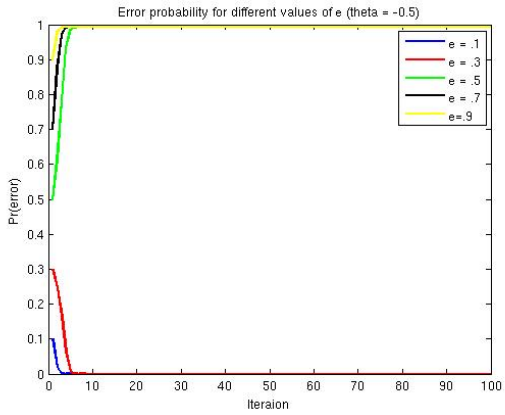

Figure 5: Evolution of error probability for different values of ϵ and $\theta = -0.5$

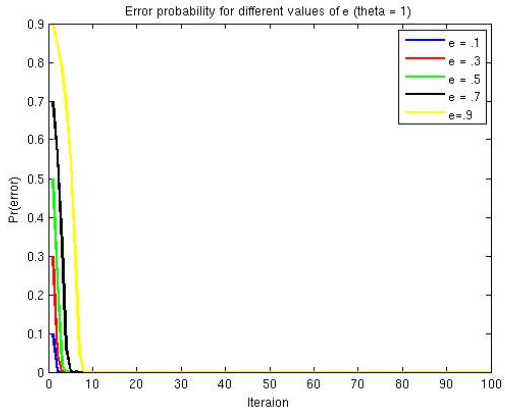

Figure 6: Evolution of error probability for different values of ϵ and $\theta = 1$

5 Hopfield Networks with Biologically Meaningful Weight Distributions

So far, we have considered the case where we have selected input patterns from codewords of an error correcting code. In this section, we consider the other extreme, i.e. we do not determine network weights according to equation (1) but assume a biologically meaningful weight distribution and investigate the behavior of the network in this way. The goal is to see if network can perform error correction in this case.

Brunel et al. [3] have considered a *perceptron*, which is an n-to-1 network. During the training phase of a perceptron, patterns of n bit are given to the network and an output is also fixed. The perceptron should memorize the relationship between input bits and the output bit by adjusting its weights according to the input patterns. In this regard, perceptron is identical to a Hopfield network except for an "output" bit which determines if recall was correct or not. In this setup and without error correction, a perceptron is able to memorize at most $M_{max} \simeq 0.14n$ patterns, which is exactly the same number as that of a Hopfield network.

Brunel et al. have investigated the weight distribution of a perceptron at its maximum capacity and shown that the weight distribution is composed

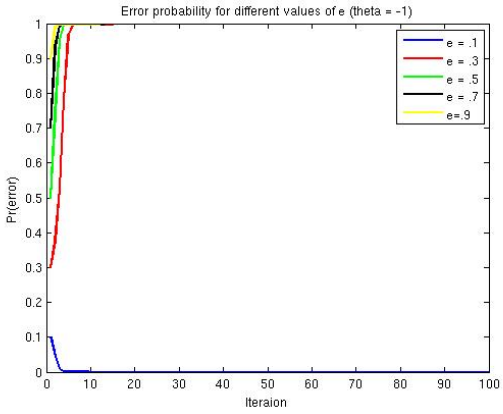

Figure 7: Evolution of error probability for different values of ϵ and $\theta = -1$

of two parts: a big dirac delta function at zero, meaning that a large number of weights are identical to zero, plus a rectified-Gaussian distribution with negative mean and a variance which depends on network parameters. By rectified here we mean that the Gaussian part is limited to positive values, i.e. inhibitory neurons are not considered. The authors have shown that their finding is compatible with biological findings.

However, other recent papers show that the weight distribution in neuronal networks in general follow a log-normal distribution [5]- [7]. In this report though, we only consider the first case and try to analyze the network behavior in this case.

5.1 Mixed-Gaussian Ensemble analysis

Following the results of [3], we assume weights to be i.i.d. random variables distributed according to:

$$f_w(x) = \alpha\delta(x) + \frac{\beta}{\sqrt{2\pi\Sigma^2}} e^{-\frac{(x-\gamma)^2}{2\Sigma^2}} u(x) \tag{6}$$

In which α and β are normalization constants, γ is the mean of the Gaussian, Σ is its variance and $u(x)$ is the step function.

We saw in the previous section that in order to investigate the dynamical behavior of Hopfield networks, we have to analyze the sums of the form $W = \sum_{i=1}^{k} w_i$ (see equation (4) for example). Assuming w_i's to be i.i.d. random variables distributed according to $f_w(x)$, as given by equation (6), the pdf of W would then be:

$$f_W(x) = f_w(x)^{*k}$$

Where, $*k$ means convoluting k times. By moving to the Fourier space, we get:

$$F_W(\omega) = F_w(\omega)^k \tag{7}$$

where capital letters indicate Fourier transforms. A few lines of calculations show that:

$$F_w(\omega) = \alpha + \beta e^{-j\omega\gamma - .5\omega^2\Sigma^2} Q\left(\frac{j\omega\Sigma^2 - \gamma}{\Sigma}\right) \tag{8}$$

Replacing equation (8) into (7), we obtain:

$$\begin{aligned} F_W(\omega) &= F_w(\omega)^k \\ &= \sum_{t=0}^{k} \binom{k}{t} \alpha^{k-t} \beta^t e^{-jt\omega\gamma - .5t\omega^2\Sigma^2} Q\left(\frac{j\omega\Sigma^2 - \gamma}{\Sigma}\right)^t \end{aligned} \tag{9}$$

If we use the approximation $Q(x) \simeq \frac{1}{\sqrt{2\pi}x} e^{-x^2/2}$, we will have:

$$\begin{aligned} F_W(\omega) &\simeq \sum_{t=0}^{k} \binom{k}{t} \alpha^{k-t} \left[\beta e^{-j\omega\gamma - .5\omega^2\Sigma^2} \frac{1}{2\pi y} e^{-y^2/2}\right]^t \\ &= \left[\alpha + \frac{\beta}{2\pi y} e^{-\gamma^2/2\Sigma^2}\right]^k \end{aligned} \tag{10}$$

Where $y = \frac{j\omega\Sigma^2 - \gamma}{\Sigma}$. Equation (10) is quite complex and although one can compute reverse Fourier transform (which would in terms of convolution of step functions), it is not easy to track.

Note that if we drop the rectification assumption, i.e. we consider original Gaussian ensembles, then the analysis becomes much more simpler as there will be no Q-function in the relationships. In this case, instead of equation (9), one gets the following relationship:

$$F_W(\omega) = \sum_{t=0}^{k} \binom{k}{t} \alpha^{k-t} \beta^t e^{-jt\omega\gamma - .5t\omega^2\Sigma^2} \tag{11}$$

Calculating the reverse Fourier transform, we get a delta function plus a summation over Gaussian random variables with different means and variances. Combining these Gaussian variables, we obtain the following distribution:

$$f_W(x) = \alpha^k \delta(x) + N(\gamma', \Sigma'^2) \tag{12}$$

Where $\gamma' = k\gamma\beta(\alpha+\beta)^{k-1}$ and $\Sigma'^2 = \sum_{t=1}^{k} \left(\binom{k}{t}\beta^t\alpha^{k-t}\right)^2 t\Sigma^2$. Using equation (12), we can derive closed form probabilities for equation (4) in terms of Q-function.

6 Conclusions and Future Works

In this report, we have mentioned preliminary steps in analyzing dynamical behavior of Hopfield networks. We considered two cases: in the first case, we selected input patterns to the network from the codewords of an error correcting code and analyzed the behavior of network in this scenario compared to the one in which patterns are selected completely at random. The goal is to show that in the modified version, the network is able to memorize more patterns and perform error to a certain degree as well.

In the second case, we considered a special weight distribution, which has some roots in biology, and made some effort to analyze the behavior of the network in this setup.

However, there remain a number of unaddressed issues which must be considered in future. The first one is calculating the storage capacity of the Hopfield network when patterns are selected from codewords of an error correcting code. Furthermore, we have to show error correcting capabilities more rigorously in this scenario. Another issue is analyzing the behavior of network in presence of log-normal weight distribution, which is the most biologically-meaningful distribution found so far. In addition, our analysis for mixed Gaussian distribution is not complete yet and we must find a way to put it in a nicer format than the one we have now. Finally, calculating mean and variance of the weights, as given in equation (1) in a simpler way (compared to the one given in the appendix) needs further attention.

A Computing Expectation and Variance of Weights

Recall from equation (1) that $w_{ij} = \frac{1}{n}\sum_{m=1}^{M} x_i^m x_j^m$. In other words, for a given codeword, we multiply the i^{th} and j^{th} bit and do a summation over all codewords to obtain w_{ij}. As a result:

$$\mu = E[w_{ij}] = \frac{1}{n}\sum_{m=1}^{M} E[x_i^m x_j^m] \tag{13}$$

To calculate $E[x_i^m x_j^m]$ we have to obtain $P_1 = \Pr\{x_i^m x_j^m = 1\}$. Denoting $\Pr\{x_i \perp\!\!\!\perp x_j\}$ by P_{ind} we obtain:

$$\begin{aligned} P_1 &= \Pr\{x_i^m x_j^m = 1\} \\ &= P_{ind} \times (\Pr\{x_i^m = 1\}\Pr\{x_j^m = 1\}\Pr\{x_i^m = -1\}\Pr\{x_j^m = -1\}) \quad (14) \\ &+ (1 - P_{ind}) \times (\Pr\{x_i^m x_j^m = 1 | x_i \text{ and } x_j \text{ are dependent}\}) \quad (15) \end{aligned}$$

Where two codewords x_i and x_j are independent if they do not share a common message bit. Note that when x_i and x_j are independent, then $\Pr\{x_i^m = 1\} = \Pr\{x_j^m = 1\} = 0.5$ (assuming uniform message probability). Furthermore, even if x_i and x_j are dependent, the probability that their product is equal to 1 would still be 0.5 unless they share exact set of message bits, i.e. we have redundant equations in constructing the code.

To determine the probability of dependence between two code bits, we consider an LDGM code with node degree distribution $(\Lambda(x), \Omega(x))$. We construct the ensemble by considering $E = \Lambda'(1) = \Omega'(1)$ sockets on both sides of the graph and picking two sockets, one from left and one from right side, uniformly at random in each iteration and connecting them. We repeat this process E times until no socket is left. Finally, if two nodes are connected an even number of times to
In the next subsection, we determine the probability that a message node with degree ℓ and a check node with degree r are connected to each other.

A.1 Connection Probability Between a Message and a Check Node

Let $p_\ell^r(s)$ denote the probability that a message node with degree ℓ is connected to a codebit with degree r at round s of constructing the code graph.

If we indicate the expected number of message bit (code bit) sockets which was selected in previous s rounds by $\alpha(s)$ ($\beta(s)$), then a simple line of reasoning shows that the following recursive solution identifies $p_\ell^r(s)$:

$$\begin{aligned} p_\ell^r(s) &= p_\ell^r(s-1)(1 - \frac{\ell - \alpha(s-1)}{E-(s-1)}\frac{r-\beta(s-1)}{E-(s-1)}) \\ &\quad + (1 - p_\ell^r(s-1))\frac{\ell - \alpha(s-1)}{E-(s-1)}\frac{r-\beta(s-1)}{E-(s-1)} \end{aligned} \tag{16}$$

and $p_\ell^r(1) = \frac{\ell}{E-(s-1)}\frac{r}{E-(s-1)}$. Lemma 1 shows that $\alpha(s+1) = \frac{\ell s}{E}$ and $\beta(s+1) = \frac{rs}{E}$.

Lemma 1. *Defining $\alpha(s)$ and $\beta(s)$ as the expected number of sockets selected from the a given message and codeword node in previous s rounds (not including round s), we find out that $\alpha(s+1) = \frac{\ell s}{E}$ and $\beta(s+1) = \frac{rs}{E}$, where ℓ and r are the degrees of the message and codeword node respectively.*

Proof. We use induction to prove the lemma. For brevity, we only consider the message bit as the same proof applies to the set of right nodes without any major changes.

In the first iteration, we know that $\alpha(1) = 0$ and $\alpha(2) = 1 \times \Pr\{\text{being selected}\} + 0 \times (1 - \Pr\{\text{being selected}\}) = \ell/E$, which satisfies the induction invariant.

Now suppose the induction holds for some s and we would like to show it also holds for $s+1$. The procedure is given below:

$$\begin{aligned} \alpha(s+1) &= \alpha(s) \times \Pr\{\text{No socket being selected in round s+1}\} \\ &\quad + (1 + \alpha(s)) \times \Pr\{\text{A socket being selected in round s+1}\} \\ &= \alpha(s) + \Pr\{\text{A socket being selected in round s+1}\} \\ &= \alpha(s) + \frac{\ell - \alpha(s)}{E - s} \\ &= \frac{\ell(s+1)}{E} \end{aligned}$$

which proves the lemma. □

Plugging the result of lemma 1 into equation (16), we can derive the closed from relationship for $p_\ell^r(s)$, as given in theorem 2.

Theorem 2. *Define $p_\ell^r(s)$ to be the probability that a message node with degree ℓ is connected to a code node of degree r at round s of graph construction. Then:*

$$p_\ell^r(s) = \frac{1 - \left(1 - \frac{2\ell r}{E^2}\right)^s}{2}$$

Proof. We use induction to prove the theorem. In the first round, the probability of being connected simply is $p_\ell^r(1) = \ell r/E^2$. Hence, induction invariant is valid for $s = 1$.

Now suppose the invariant holds for some s. We would like to show that it also holds for $s + 1$. Following equation (16) we have:

$$\begin{aligned}
p_\ell^r(s+1) &= p_\ell^r(s)(1 - \frac{\ell - \alpha(s)}{E - (s)}\frac{r - \beta(s)}{E - (s)}) \\
&\quad + (1 - p_\ell^r(s))\frac{\ell - \alpha(s)}{E - (s)}\frac{r - \beta(s)}{E - (s)} \\
&= p_\ell^r(s)(1 - \frac{\ell r}{E^2}) + (1 - p_\ell^r(s))(\frac{\ell r}{E^2}) \\
&= p_\ell^r(s)(1 - 2\frac{\ell r}{E^2}) + \frac{\ell r}{E^2} \\
&= \frac{1 - \left(1 - \frac{2\ell r}{E^2}\right)^{s+1}}{2}
\end{aligned}$$

which proves the theorem. □

Theorem 2 provides us with necessary tools to calculate the probability that two right nodes are dependent. Consider two right nodes with degrees r_i and r_j respectively. The probability that a left node is connected to both of these nodes at the end of the construction process is $p_\ell^{r_i}(E) \times p_\ell^{r_j}(E)$. Hence, the probability that two code nodes do not share <u>any</u> message node, and hence are independent, is:

$$\begin{aligned}
P_{ind} &= \left(1 - \sum_\ell \sum_{r_i} \sum_{r_j} (\frac{\Lambda_\ell}{k})(\frac{\Omega_{r_i}}{n})(\frac{\Omega_{r_j}}{n}) p_\ell^{r_i}(E) \times p_\ell^{r_j}(E)\right)^k \\
&= \left(1 - \sum_\ell \sum_{r_i} \sum_{r_j} (\frac{\Lambda_\ell}{k})(\frac{\Omega_{r_i}}{n})(\frac{\Omega_{r_j}}{n}) \frac{1 - \left(1 - \frac{2\ell r_i}{E^2}\right)^E}{2} \frac{1 - \left(1 - \frac{2\ell r_j}{E^2}\right)^E}{2}\right)^k \quad (17)
\end{aligned}$$

Where k is the number of message nodes.

Now what is the probability that two nodes share the same set of message bits? The probability that two right nodes with degree r share the same message node with degree ℓ is $p_\ell^r(E)^2$. Hence, on average, the probability that two right nodes with degree r share a left node is $P = \sum_\ell \frac{\Lambda_\ell}{k} p_\ell^r(E)^2$. As a result, the probability that they share all their r nodes together is P^r. Finally, the probability that any two right nodes have the same degree and share the same set of message bits is:

$$\begin{aligned} P_{redun} &= \sum_r (\frac{\Omega_r}{2})^2 \binom{k}{r} P^r \\ &= \sum_r (\frac{\Omega_r}{2})^2 \binom{k}{r} \left(\sum_\ell \frac{\Lambda_\ell}{k} p_\ell^r(E)^2 \right)^r \end{aligned} \tag{18}$$

Combining the results of equations (14), (17) and (18) we see that:

$$\begin{aligned} P_1 &= .5P_{ind} + (1 - P_{ind})(1 \times P_{redun} + .5 \times (1 - P_{redun})) \\ &= .5 + .5(1 - P_{ind})P_{redun} \end{aligned}$$

Having P_1 we can calculate the expectation of $x_i^m x_j^m$, and hence w_{ij} which would be:

$$\begin{aligned} \mu = E[w_{ij}] = \frac{1}{n} \sum_{m=1}^{M} E[x_i^m x_j^m] = \frac{1}{n} \sum_{m=1}^{M} (1 \times P_1 + (-1) \times (1 - P_1)) = \frac{M}{n}(2P_1 - 1) \\ = \frac{M}{n}(1 - P_{ind})P_{redun} \end{aligned} \tag{19}$$

The same line of reasoning can also apply to calculate variance of w_{ij}. If we assume different codewords to be independent of each other, then:

$$E[w_{ij}^2] = \frac{1}{n^2} \sum_{m=1}^{M} E[(x_i^m x_j^m)^2] = \frac{1}{n^2} \sum_{m=1}^{M} (1 \times P_1 + (1) \times (1 - P_1)) = \frac{M}{n^2} \tag{20}$$

And hence $\sigma^2 = E_[w_{ij}^2] - \mu^2 = \frac{M - (M(1-P_{ind})P_{redun})^2}{n^2}$.

References

[1] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities", National Acad Science, 1982.

[2] C. Berrou, V. Gripon, "Coded Hopfield Networks", Proc. Symposium on Turbo Codes and Iterative Information Processing, pp. 15, 2010.

[3] N. Brunel, V. Hakim, P. Isope, J. P. Nadal, B. Barbour, "Optimal Information Storage and the Distribution of Synaptic Weights:: Perceptron versus Purkinje Cell", Vol. 43, No. 5, 2004, pp. 745-757.

[4] R. McEliece, E. Posner, E. Rodemich, S. Venkatesh, "The capacity of the Hopfield associative memory", IEEE Transaction on Information Theory, Vol. 33, No. 4, 1987, pp. 461-482.

[5] Tom Hromdka, Michael R. DeWeese, Anthony M. Zador, "Sparse Representation of Sounds in the Unanesthetized Auditory Cortex", PLoS Biology, 6, 2008,

[6] C. Holmgren, T. Harkany, B. Svennenfors, Y. Zilberte, "Pyramidal cell communication within local networks in layer 2/3 of rat neocortex", The J. Physiology, 551, 2003, pp. 139-153.

[7] S. Lefort, C. Tomm, J. C. F. Sarria, C. C.H. Petersen, " The Excitatory Neuronal Network of the C2 Barrel Column in Mouse Primary Somatosensory Cortex", Neuron, Vol. 61, No. 2, 2009, pp. 301-316.