Memory Capacity of Hopfield Networks

Tony Fu

AMATH 383 Fall 2021 Final Term Paper

Contents

1 Introduction

Hopfield network is an important model for association memory (a.k.a. content-addressable memory), a system that is able to retrieve memorized patterns with partial cues (Hopfield, 1982). However, a Hopfield network has a limited memory capacity (α_c), beyond which the network performance degrades and its dynamics can be easily trapped in metastable energy states called spurious attractors/patterns (Hopfield, 1982; Amit et al., 1985). This capacity was suggested to be scaled linearly with the size of the network N with a factor of ~ 0.15 (Hopfield, 1982), 0.138 (Amit et al. 1985), or 0.144 (Crisanti et al., 1986). Others had suggested this α_c is proportional not to N but rather to $N/2\ln(N)$ (private communications quoted by Amit et al., 1985). In this term paper, I used the probability of a neuron to escape from a memorized pattern as an approximation for the original pattern's local stability. My calculations showed that, for an error rate of 0.5%, the capacity $\alpha_c \approx 0.15$ regardless of the network size N. The computer simulation resulted in similar results, but the error rates were found to increase with the network size N.

2 Problem Description

The ability of a neural network to possess long-term memory is endowed by the synaptic strength between the neurons, rather than by the activation states of the neurons (Gerstner et al., 2014). The connections between those neurons allow the memorized patterns to be the fixed points (a.k.a. attractors) of the network dynamics (Amit et al., 1985).

The Hopfield network is an important model for associative memory or content-addressable memory. Unlike perceptron, which is purely feedforward, the Hopfield network is highly recursive such that every neuron has the opportunity to be each other's input and output (Hopfield, 1982; Hopfield and Tank, 1985; Folli et al., 2017). The Hopfield network's ability to store and retrieve information comes from its unique training and update rules. The "training" of a Hopfield network coincides with Hebbian learning because it creates an auto-correlation matrix in which only the connections between the activated neurons are strengthened.

However, a Hopfield network's memory capacity is limited. The factor that limits this capacity is inherent to Hebbian learning as too many training patterns quickly complicate the energy landscape of the neuronal state space (Folli et al., 2017). Thus, an extensive Hopfield network can easily create so-called spurious attractors, which are erroneous fixed points that arise from training (Hopfield, 1982; Amit et al., 1985). The activation pattern can become trapped in one of those spurious attractors and fails to converge to the intended pattern. An example of a spurious pattern can be seen in Figure 1. The letters ”T”,”O”,”N”, and ”Y” were encoded in the network’s connection matrix. A spurious pattern arose in this matrix, and it was a mixture of the letters ”T” and ”Y” likely because of how similar the two patterns were. The network had a size of $N = 5 \times 5 = 25$, and it memorized $M = 4$ patterns. The pattern-to-neuron ratio is $\alpha = M/N = 4/25 \approx 0.16$.

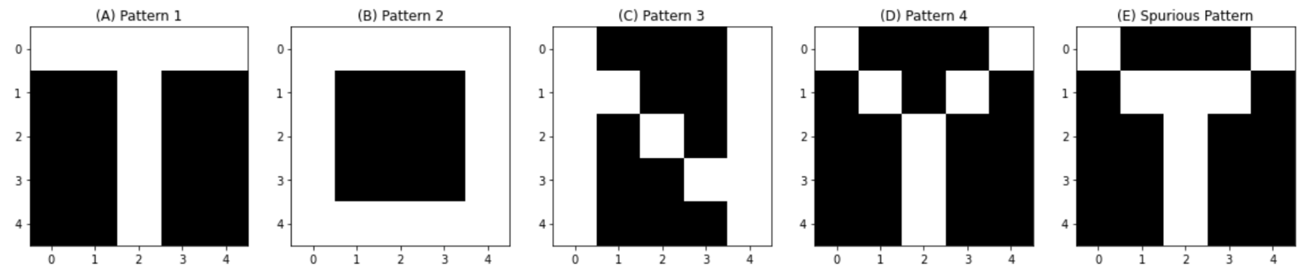

Figure 1. Example of a Spurious Pattern. (A)-(D) Letters ”T”,”O”,”N”, and ”Y” were encoded in the network’s connection matrix. (E) A spurious pattern that was a mixture of the letter ”T” and ”Y”. See **Appendix** for code.

Hopfield (1982) suggested that the memory capacity $\alpha_c \approx 0.15$. Other researchers used more advanced techniques and showed the α_c is smaller. Amit et al. (1985) suggested $\alpha_c = 0.138$, and Crisanti et al. (1986) revised the calculations and got $\alpha_c = 0.144$. However, the true relationship between the network size N and the number of memorized pattern M remains debatable today (Folli et al., 2017). The calculations of the aforementioned authors were also too obscure to be understood by an average undergraduate student. In this term paper, I used the modeling tools we learned in AMATH 383 (also with the help of references and my prior knowledge in MATH 394 & 395 and scientific programming) to estimate the memory capacity α_c and investigate its relationship with the network size N.

3 Simplifications

1. Weight Symmetry: I assume the connections within the Hopfield network to be symmetric. Despite its simplicity, symmetric neural networks are actually more biologically plausible than the conventional feedforward networks (e.g., perceptrons), and they are able to store more information with less neurons, thereby reducing the training time (Na and Park, 1992). It is also a default for Hopfield networks because the Hebbain encoding proposed by Hopfield (1982) is an auto-correlation, which is symmetric.

2. Binary Neuronal States: The states of the neurons are binary (either 1 or -1) because action potentials are all-or-nothing phenomenon. The magnitudes of action potentials also tends to saturate at a fixed level. By assuming the neuronal states to be two-valued, I factor out the inherent difference between neurons and put the emphasis on Hopfield network's algorithm.

3. Unspecified Time Step: The time intervals between successive updates is not relevant. It is usually left unspecified by researchers (Gerstner et al., 2014). This means that I do not consider the effects of refraction, etc.

4. Non-Stochastic Update: The outcome of a neuronal state after each update is perfectly predictable if I know the current neuronal states and the connection matrix. That is, the temperature (T) of the network is assumed to be zero (i.e., $\beta = T^{-1} = \infty$). Again, this is to put the emphasis on the network itself, as I would not like the timing to confound the relationship between the memory capacity and the error rate. Also, the original Hopfield network is time invariant.

5. Random Memorized Patterns: To avoid having to hard-code the patterns, I randomly create them with equal probabilities to be ± 1. That way, I can assume each neuronal states to be a i.i.d., two-valued random variable with a mean of zero. I argue that this would provide the most generalized analysis of the Hopfield network because manually curated patterns may introduce human bias.

6. Assume Large M and N: The theoretical modeling assumes the number of patterns M and the network size N to be sufficiently large. It is a reasonable simplification because a useful network that is able to memorize meaningful patterns would have to be moderately large.

7. Error Definition: An ”error” is defined in this paper according to Hertz et al.’s (1991) condition for a pattern’s local stability. The number of errors is the number of neurons that flips after the network has converged completed to a memorized pattern.

4 Variables

- N: the number of neurons.
- $\mathbf{S}(t)$: the network state vector of all the neurons.
- $S_i(t)$: the state of the i-th neuron ($1 \leq i \leq N$). Can be either 1 (active) or -1 (inactive).
- Δt: the time interval between each successive updates. Typically left unspecified.
- $\mathbf{J}$: the connection matrix of size $N \times N$.
- J_{ij}: the weight that connects from S_i to S_j.
- $h_i(t)$: the input potential of neuron i.
- β: the inverse of the ”temperature” (i.e., $\beta = T^{-1}$) of the network.
- M: the number of patterns p^μ memorized by the network.
- p_i^μ: the i-th neuron of the μ-th pattern ($1 \leq \mu \leq M$). Can be either 1 (active) or -1 (inactive).
- $m_\mu(t)$: the similarity (a.k.a. ”overlap”) between the current network state $\mathbf{S}$ and the μ-th pattern.
- $E(t)$: the energy of the network.
- x: $= p_i^\mu\, p_j^\mu\, p_i^\nu\, p_j^\nu$. The ”cross-talk” between two neurons in pattern ν and two neurons from another pattern $\mu \neq \nu$. Can also be seen as a step in random walk if each term that makes up the product (e.g., p_i^μ) are i.i.d..
- y: the final location of the random walk that is a sum of x.
- r: the number of steps that $x = +1$ (rightward movement).
- z: $= y/N$ the N normalized final location.
- α: the pattern-to-neuron ratio ($\alpha = M/M$).
- α_c: the capacity of the network.
- ϵ: the error rate (i.e., the probability of a neuron to flip after the network has converged completely to a pattern. It is also the proportion of neurons that will flip after the network has converged completely to a pattern.)

5 Mathematical Model

I approach the question of Hopfield network's memory capacity α_c with two methods:

1. Theoretical Modeling: estimates the pattern's local stability by determining the probability of a neuron to escape from a memorized pattern. The neuronal states of the patterns are assumed to be i.i.d. random variables, and the probability to escape (to err) can be treated as a random walk.

2. Computer Simulations: runs the Hopfield network algorithm with Python with different parameters N and M, then calculates the average error rates.

Subsections 5.1-5.3 introduce the basics of Hopfield networks. Those subsections provide important basis for both theoretical modeling and computer simulations. An additional subsection 5.4 covers the details of theoretical modeling.

5.1 Network Structure Inspired by Hebbain Learning

We begin with a network consisting of N neurons. Each neuron has two possible states: 1 (active) or -1 (inactive). Therefore the states of all the neurons in the network can be represented by a vector $\mathbf{S}(t) = [S_1, S_2, S_3, \ldots, S_N]$. The network is fully connected, meaning that, during an update, a neuron receives exactly one input from each of the neurons in the network. The means that all neurons have the opportunity to give exactly one output to all the other neurons. A neuron's state S_i is therefore a function of time, although the time step Δt is usually left unspecified (Gerstner et al., 2014). On the other hand, the connections are constant and are where the network's memories are stored. The connection matrix $\mathbf{J}$ is constructed such that the memorized patterns p are the fixed points of the network dynamics. To store a given pattern p^μ, we construct the connection matrix $\mathbf{J}$ in accordance to the Hebbain rule (1949), which states that neurons that fire together, wire [positively] together. It is reasonable to apply such a heuristic because we would expect the neurons with the opposite state in a pattern to connect with negative weights. The connection matrix $\mathbf{J}$ for a single pattern p is therefore defined as the N-normalized auto-correlation of the pattern p (Gerstner et al., 2014):

$$J_{ij} := \frac{1}{N} p_i \, p_j$$

To store M patterns, we simply sum up the all the auto-correlations and normalized the sum by N (Gerstner, 2014):

$$J_{ij} := \frac{1}{N} \sum_{\mu=1}^{M} p_i^{\mu} \, p_j^{\mu}. \tag{1}$$

5.2 Update Rule

The Hopfield network introduced in this term paper is discrete and asynchronous. Thus, only one neuron can be updated at a time. For example, to update neuron S_i, we do a dot product between the i-th row of the connection matrix **J** and the network state vector **S** (Gerstner et al., 2014). This gives us the input potential h of a neuron:

$$h_i(t) := \sum_{j=1}^{N} J_{ij} \, S_j$$

We know a biological neuron fires if the input potential exceed certain threshold, and each action potential of a neuron has similar strength regardless of the input magnitude (sodium channels begin to close as the neuron fires). Those are nonlinear behaviors that a dot product like the one above cannot reproduce. Thus, we need some kind of nonlinearity that binarize the input potential $h_i(t)$ to update the state to either 1 or -1. Additionally, there is firing may not be deterministic. To incorporate the stochasticity and the thresholding of an action potential, the probability of the next state of neuron i being +1 is therefore defined as (Gerstner et al., 2014):

$$\mathbb{P}\Big(S_i(t + \Delta t) = 1 \Big| h_i(t)\Big) := \frac{1}{2}\Big[1 + \tanh(\beta h_i)\Big] = \frac{1}{2}\Big[1 + \tanh(\beta \sum_{j=1}^{N} J_{ij} \, S_j)\Big],$$

The +1 offset and the one-half factor are there to shift the output of $\tanh(x)$ to between 0 and 1. The β is the inverse of the network's "temperature" T. That is, $\beta = T^{-1}$. Amit et al. (1985) have proposed to think to such a network as a spin glass with many magnetic spins. We know that at high temperatures, the heat energy allows the spins to switch their directions of their magnetic momentum and become misaligned more easily. Similarly, a high temperature allows the neurons in the network to change from their current state more easily. We can see that when the temperature is very high ($\beta \approx 0$), the $\tanh(x)$ becomes near zero, and the probabilities of either state become close to 0.5 (i.e., random). On the other hand, consider the case when the temperature of the network is 0, then $\beta = \infty$. Depending on the sign of the input potential $h_i(t)$, the probability of the current state becomes either 0 or 1. In other words, the sign function then is applied to the input potential $h_i(t)$ to make sure the result is limited to only two cases, namely, 1 or -1 (Folli et al., 2013). In math, the update rule is expressed as:

$$S_i(t + \Delta t) := \text{sgn}\left[\sum_{j=1}^{N} J_{ij} S_j(t)\right]. \tag{2}$$

For computer simulations: To make sure that every neuron is updated, the update rule will be applied pseudo-randomly. That is, for each "iterations", all the neurons will be updated once, but the updating sequence is random from iteration to iteration.

5.3 Similarity and Energy

The similarity measure (m) (sometimes named "overlap") between the current network state $\mathbf{S}$ and μ-th pattern p^μ is defined as the N-normalized dot product of the two (Gerstner et al., 2014):

$$m^\mu(t) := \frac{1}{N}\sum_{i=1}^{N} p_i^\mu\, S_i. \tag{3}$$

A pattern p^μ is said to be retrieved if the similarity between it and the current network state $\mathbf{S}(t)$ is 1 (perfectly similar). A state that is uncorrelated pattern p^μ would yield a similarity of 0. Similarity can be negative if the current state is inversely correlated with pattern p^μ.

The next concept is energy E. The network is modeled such that the memorized patterns are the fixed points of the network dynamics. To quantify how close the current network state $\mathbf{S}$ is to those patterns p (i.e., the fixed points), we can think of the network state space as a multi-dimensional landscape in which we are trying to find the valleys (i.e., the stable equilibria that correspond to the memorized patterns). Once again borrowing the spin-glass analogy from physics, the ”height” of this landscape is analogous to the energy of the ferromagnets (Amit et al., 1985). This energy is defined as follow (Hopfield, 1982):

$$E(t) := -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} J_{ij}\, S_i\, S_j. \tag{4}$$

For example, if $S_i = S_j = 1$, and the connection between J_{ij} them is positive, then the triple product $J_{ij}\, S_i\, S_j > 0$, contributing negatively to E because of the negative sign in Formula (4). This is expected because Hebbian rule encourages the two neurons to have the same sign if their connection is positive. Conversely, if $S_i = -1$ and $S_j = 1$, and the connection between J_{ij} them is still positive, then the triple product $J_{ij}\, S_i\, S_j < 0$, contributing positively to E, increasing the energy of the network. Substituting the definition of J_{ij} from Equation (1):

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} J_{ij}\, S_i\, S_j = -\frac{1}{2N}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\mu=1}^{M} p_i^{\mu}\, p_j^{\mu}\, S_i\, S_j$$

Substituting the definition of similarity m_{μ} from Equation (3):

$$E = -\frac{1}{2N}\sum_{\mu=1}^{M}(m^{\mu})^2. \tag{5}$$

We can see that the energy is proportional to the sum of all the squared similarities $(m^{\mu})^2$. Looking at Equation (5), we can immediately identify the problem with having too many patterns. If M is too large, then even if we are close to retrieving a pattern, the energy E can remain large (i.e., close to zero) because of the addition of all other similarities. To make the matter even worse, sometimes a network state occur at the ”intersection” of many memorized patterns can actually have a low energy, resulting in a spurious pattern that have some characteristics of the adjacent patterns. This limits the number of patterns that a network can store. *For computer simulations:* The concept of energy is important because it will be used as a convergence criteria. In other words, the computer simulation will stop when E does not change for, for example, 5 iterations.

5.4 Theoretical Modeling of memory Capacity

It has already been pointed out by J. J. Hopfield himself (1982) that retrieval errors can occur in networks like these. Sometimes the network state is trapped in a spurious pattern (metastable state) that is a ”well-defined mixtures of several patterns” (Amit et al., 1985). This may explain why we sometimes have memories that are combinations of several distinct episodes, and our memories are fallible as a result.

Here I introduce the theoretical limit of the number of patterns we can store before the network is overwhelmed by the spurious patterns. This limit was briefly introduced in Hopfield's original paper (1982), and Hopfield concluded the limit to be $0.15N$. Here we define retrieval error according to the condition put forward by Hertz et al. (1991), who stated that a network that was initialized with one of the memorized patterns should not change from that pattern. In other words, a network with an initial state $S(0) = p^\nu$ should not change a single neuron state $S_i(t)$ over time. *Every neuron that flips from its initial state is considered an error.* To start, I have introduced the dynamics of the Hopfield network at zero temperature ($\beta = \infty$) as the difference equation shown in Equation (2). Assuming the network is initialized with pattern ν, that is $S(0) = p^\nu$:

$$\begin{aligned}
S_i(t+\Delta t) &= \operatorname{sgn}\left[\sum_{j=1}^{N} J_{ij} p_j^\nu\right] \\
&= \operatorname{sgn}\left[\sum_{j=1}^{N}\left(\frac{1}{N}\sum_{\mu=1}^{M} p_i^\mu p_j^\mu\right) p_j^\nu\right] \\
&= \operatorname{sgn}\left[\frac{1}{N}\sum_{\mu=1}^{M}\sum_{j=1}^{N} p_i^\mu p_j^\mu p_j^\nu\right]
\end{aligned}$$

We can break this expression above into $\mu = \nu$ and $\mu \neq \nu$:

$$\begin{aligned}
S_i(t+\Delta t) &= \operatorname{sgn}\left[\frac{1}{N}\sum_{\mu=\nu}\sum_{j=1}^{N} p_i^\mu p_j^\mu p_j^\nu + \frac{1}{N}\sum_{\mu\neq\nu}^{M}\sum_{j=1}^{N} p_i^\mu p_j^\mu p_j^\nu\right] \\
&= \operatorname{sgn}\left[\frac{1}{N}\sum_{j=1}^{N} p_i^\nu p_j^\nu p_j^\nu + \frac{1}{N}\sum_{\mu\neq\nu}^{M}\sum_{j=1}^{N} p_i^\mu p_j^\mu p_j^\nu\right]
\end{aligned}$$

Because p_j can only be ± 1, therefore $p_j^\nu p_j^\nu = 1$. We can remove it from the above expression:

$$\begin{aligned} S_i(t+\Delta t) &= \text{sgn}\left[\frac{1}{N}\sum_{j=1}^{N} p_i^\nu + \frac{1}{N}\sum_{\mu\neq\nu}^{M}\sum_{j=1}^{N} p_i^\mu p_j^\mu p_j^\nu\right] \\ &= \text{sgn}\left[p_i^\nu + \frac{1}{N}\sum_{\mu\neq\nu}^{M}\sum_{j=1}^{N} p_i^\mu p_j^\mu p_j^\nu\right] \end{aligned}$$

For reason that will be clear in a minute, let's factor p_i^ν out of the sign function:

$$S_i(t+\Delta t) = p_i^\nu \, \text{sgn}\left[1 + \frac{1}{N}\sum_{\mu\neq\nu}^{M}\sum_{j=1}^{N} p_i^\mu p_j^\mu p_j^\nu \frac{1}{p_i^\nu}\right]$$

Because p_i^ν is only ± 1, dividing by it is equal to multiplying by it:

$$S_i(t+\Delta t) = p_i^\nu \, \text{sgn}\left[1 + \frac{1}{N}\sum_{\mu\neq\nu}^{M}\sum_{j=1}^{N} p_i^\mu p_j^\mu p_i^\nu p_j^\nu\right]$$

For an error *not* to occur, sng$[\ldots] = 1$. That is, we would like $\frac{1}{N}\sum_{\mu\neq\nu}^{M}\sum_{j=1}^{N} p_i^\mu p_j^\mu p_i^\nu p_j^\nu > -1$. Only in that case does this neuronal state belong to a stable fixed point ($S_i(t) = S_i(t+\Delta t) = p_i^\nu$). To generalize, let's assume every neuron state of the memorized patterns is an i.i.d. binary random variable with mean zero. That is, $\mathbb{P}(p_i^\mu = +1) = \mathbb{P}(p_i^\mu = -1) = 0.5$ and $\mathbb{P}(p_i^\mu = 1)\mathbb{P}(p_j^\mu = 1) = \mathbb{P}(p_i^\mu = 1, p_j^\mu = 1)$ for all μ. The question now is, does the product $p_i^\mu p_j^\mu p_i^\nu p_j^\nu$ also has the same distribution as individual p_i^μ (or p_j^μ, p_i^ν, or p_j^ν–they are all i.i.d.)? Let's exhaust all 16 possible products to find out its probability mass function (pmf):

p_i^μ	p_j^μ	p_i^ν	p_j^ν	$x_j^\mu = p_i^\mu\, p_j^\mu\, p_i^\nu\, p_j^\nu$
-1	-1	-1	-1	1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	-1	1	1	1
-1	1	-1	-1	-1
-1	1	-1	1	1
-1	1	1	-1	1
-1	1	1	1	-1
1	-1	-1	-1	-1
1	-1	-1	1	1
1	-1	1	-1	1
1	-1	1	1	-1
1	1	-1	-1	1
1	1	-1	1	-1
1	1	1	-1	-1
1	1	1	1	1

Table 1. All possible values for $p_i^\mu\, p_j^\mu\, p_i^\nu\, p_j^\nu$.

We see from Table 1 that the pmf turns out to be the same as individual p_i^μ. We know all combinations have equal probabilities because all individual p terms are i.i.d. with equal probabilities for ± 1. Therefore, the pmf: $\mathbb{P}(p_i^\mu\, p_j^\mu\, p_i^\nu\, p_j^\nu = 1) = \mathbb{P}(p_i^\mu\, p_j^\mu\, p_i^\nu\, p_j^\nu = -1) = \frac{1}{2}$. Furthermore, we can treat each $p_i^\mu\, p_j^\mu\, p_i^\nu\, p_j^\nu$ as an independent event. For simplicity, from now on let's rename this product as $x = p_i^\mu\, p_j^\mu\, p_i^\nu\, p_j^\nu$. The sum of this random, independent, two-valued events x is an one-dimensional lattice random walk with a step size of 1. There are $(M-1) \times N$ steps because there are N neuronal states in each pattern, and we are excluding the pattern ν. Since $\mathbb{P}(x = 1) = \mathbb{P}(x = -1) = 0.5$, we know the probability density of the final location y is going to center at $y = 0$. Let's assume r to be the number of steps that $x = 1$ (rightward movement). The number of steps that $x = -1$ thus becomes $(M-1)N - r$, and the final location $y = r - ((M-1)N - r) = 2r - (M-1)N$. This random walk results in a discrete probability distribution of r that is $\sim$ Binomial$((M-1)N, 0.5)$:

$$\mathbb{P}(r = k) = \binom{(M-1)N}{k} \left(\frac{1}{2}\right)^{(M-1)N}, \tag{6}$$

for $k = 0, 1, 2, \ldots, (M-1)N$. We know such a distribution would have a variance of Var$[r] = (M-1)N \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{(M-1)N}{4}$. What is then the variance of the final location $y = 2r - (M-1)N$? We know that Var$[cx] = c^2$Var$[x]$ and Var$[x - c] =$ Var$[x]$ (where c is a constant and x is a random variable). Therefore, the variance of the final location y becomes:

$$\text{Var}[y] = \text{Var}[2r - (M-1)N] = \text{Var}[2r] = 4\text{Var}[r] = (M-1)N$$

We have now modeled the final location $y = \sum x$ (i.e., the sum of all cross-talks x) as $\sim$ Binomial$((M-1)N, 0.5)$. Our goal is to find the probability that this final location y is less than -1. If so, the current pattern is not locally stable because a y less than -1 will result in a change in the neuronal state, meaning that $S(t+\Delta t) \neq S(t)$. The probability is calculated in **Solution of the Mathematical Problem**.
For computer simulations: Computer simulations will be carried out to see if the approximation from Equation (7) agrees with the simulated network dynamics.

6 Solution of the Mathematical Problem

6.1 Numerical Solution of Theoretical Model

I have modeled the sum of the cross-talks (or the final location of the random walk) y as $\sim$ Binomial$((M-1)N, 0.5)$. If this quantity y dropped below -1, the neuron will flip from the pattern, and an error was made. Now, assume $(M-1)N$ is large, we can then approximate this binomial distribution to a normal distribution $y \sim \mathcal{N}(0, (M-1)N)$. To find the probability of an error means finding the probability: $\mathbb{P}(\frac{y}{N} < -1)$. Again using Var$[cx] = c^2Var[x]$, we know the variance of $\frac{y}{N}$ is:

$$\text{Var}[\frac{y}{N}] = \frac{1}{N^2}\text{Var}[y] = \frac{M-1}{N}$$

Assuming $M >> 1$:

$$\text{Var}[\frac{y}{N}] \approx \frac{M}{N} = \alpha$$

, where $\alpha = M/N$. The normalized final location $z = \frac{y}{N} \sim \mathcal{N}(0, \alpha)$. And the probability $\mathbb{P}(\frac{y}{N} < -1)$ is the pdf of the Normal distribution integrated from $-\infty$ to -1.

$$\epsilon(\alpha) = \mathbb{P}(z < -1) = \int_{-\infty}^{-1} \frac{1}{\sqrt{2\pi\alpha}} e^{-\frac{z^2}{2\alpha}}\, dz, \tag{7}$$

$\epsilon(\alpha)$ is the probability of a neuron to flip from the initial pattern ν (i.e., to err). Because individual neurons err independently, this probability is also the ensemble probability and can be understood as the error rate of the network. This would also mean that the number of neurons that will err is $N \times \epsilon(\alpha)$. This error rate $\epsilon(\alpha)$ from Formula (7) is a function of α. We can define the capacity (α_c) of the network as the α that corresponds to a maximum tolerable error rate $\epsilon(\alpha)$. Assuming we can tolerate an error rate of $\epsilon = 0.5\%$:

$$0.005 = \int_{-\infty}^{-1} \frac{1}{\sqrt{2\pi\alpha_c}} e^{-\frac{z^2}{2\alpha_c}}\, dz$$

The above expression cannot be solved in closed form (at least not with the tools I learned in the Calculus series). Mathematica returns $\alpha_c \approx 0.151$. Using other error rates $\epsilon(\alpha)$, we get the following capacities α_c's:

error rate $\epsilon(\alpha)$	capacity $\alpha_c = M/N$
0.1%	0.105
0.2%	0.121
0.3%	0.132
0.4%	0.142
0.5%	**0.151**
1.0%	0.185
2.0%	0.237
3.0%	0.283
4.0%	0.326
5.0%	0.370
10.0%	0.609
15.0%	0.931
15.86%	**1.000**
20.0%	1.412
25.0%	2.198
30.0%	3.636

Table 2. Memory Capacities with the Corresponding Error Rates.

6.2 Computer Simulations

A set of M number of random patterns p of shape $N \times 1$ (column vector) were created. The vector form of Equation (1) was used to create the connection matrix $\mathbf{J}$:

$$J = \frac{1}{N}\left(\sum_{\mu=1}^{M} pp^T - \mathbb{I}\right), \tag{8}$$

where $\mathbb{I}$ is the identity matrix of shape $N \times N$. It was subtracted from the auto-correlation of each pattern to set the diagonal entries to zeros (i.e., remove autapses). One of those patterns was picked at random as the starting neuronal state: $\mathbf{S}(0) = p^\nu$. The network was updated according to Equation (2), and it was applied pseudo-randomly to ensure all neurons were updated exactly once before moving on to another iteration. The update would stop when the energy E stopped changing for 5 iterations, indicating the neuronal states $\mathbf{S}$ had found a stable fixed point. The energy E was calculated according to a vector form of Equation (4):

$$E(t) = -\frac{1}{2}\mathbf{S}^T\,\mathbf{J}\,\mathbf{S}. \tag{9}$$

The number of state changes was calculated by comparing the final neuronal states with its staring condition. The error rate ϵ was calculated as this number normalized by the network size N, and the algorithm was repeated 30 trials for the same N and M to find the average error rate. The entire scheme was repeated for different N (10, 50, 100, 200, and 1000), with varying M to cover a pattern-to-neuron ratio (α) from about 0.01 to 10. Finally, the average error rates for each choice N was plotted as a curve. The resulting plot can be found in **Results and Discussion** as Figure 4. For implementation details, see **Appendix**.

7 Results and Discussion

7.1 Numerical Solution of Theoretical Model

The calculation from the previous section shows that $\alpha_c \approx 0.15$. This means that a network of with $N = 100$ neurons may reliably retrieve $M \approx 15$ randomly selected patterns at $\epsilon = 0.5\%$. This is as predicted by Hopfield (1982) using computer simulations. Other researchers used more advanced techniques and showed the α_c is smaller. For example, Amit et al. (1985) used the replica-symmetric theory and calculated $\alpha_c = 0.138$. Crisanti et al. (1986) revised the calculation and got $\alpha_c = 0.144$. Their models are too advanced to be within the scope of this class. Nevertheless, it is interesting to see that the theoretical capacity α_c can actually exceed 1.0 for a maximally tolerable error rate of only 15.86% (Table 2). Figure 2 shows the relationship between the pattern-to-neuron ratio (α) and the theoretical error rate (ϵ) according to Equation (7). The error rate increased sharply after $\alpha \sim 0.1$ and seemed to plateau at $\epsilon \approx 0.35$. Folli et al. (2017) have also calculated the relationship between the α and the different error probabilities (Figure 3). Interestingly, their calculations suggested that the error would decrease after sufficiently large α (i.e., memory capacity could actually be much higher than the network size N). However, this behavior was not observed in Equation (7).

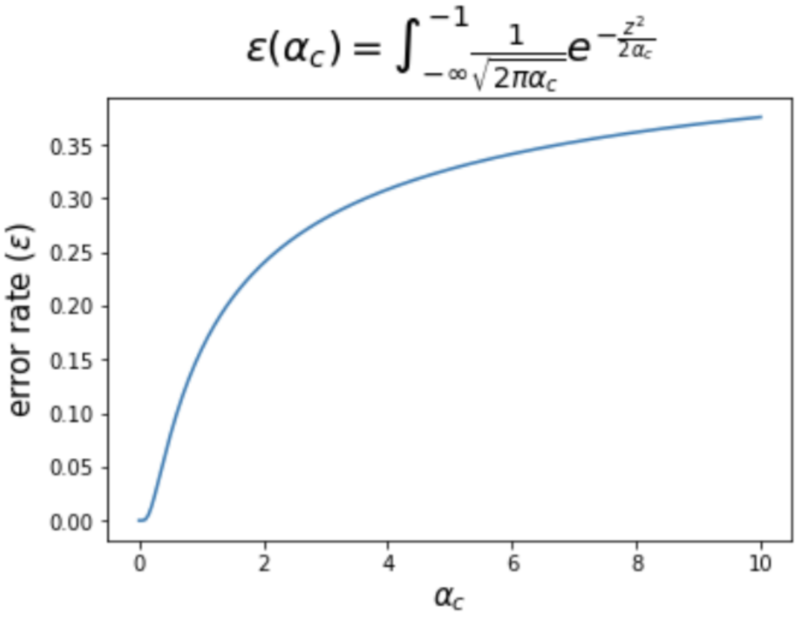

Figure 2. Relationship Between Memory Capacity (α_c) and Error Rate (ϵ) as Modeled with Equation (7). The error rate increased sharply after around $\alpha_c = 0.1$ and seemed to plateau at $\epsilon \approx 0.35$. See **Appendix** for code.

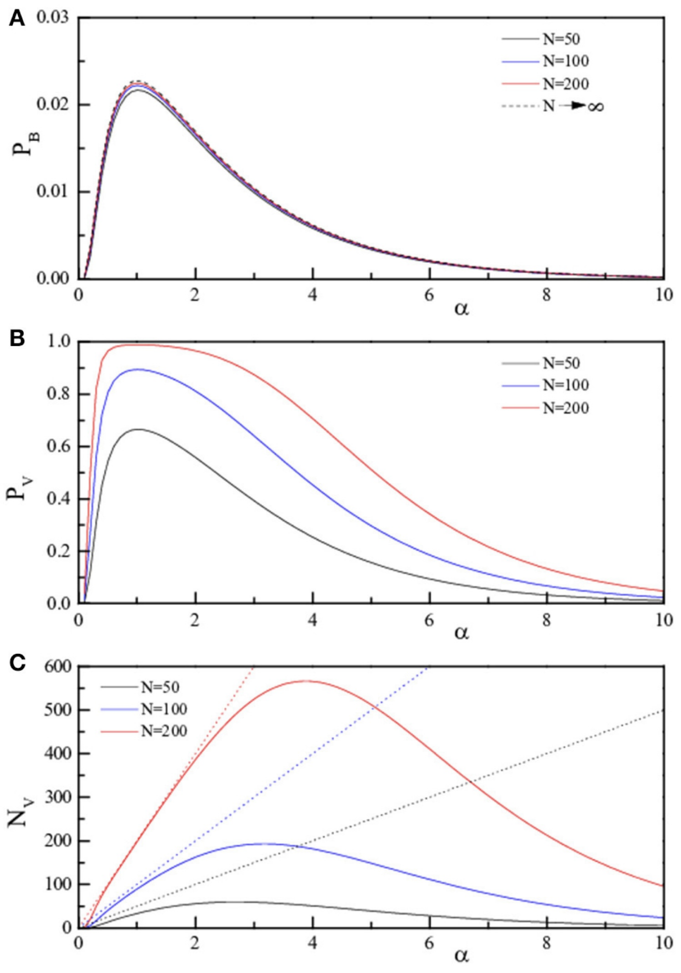

Figure 3. Folli et al.'s (2017) Theoretical Curves for Three Different Error Measurements. The three curves are plotted against α, the capacity M/N of the network. (A) p_B is the proportion of the memorized patterns that turn out to be an unstable fixed point (or not even a fixed point at all). (B) p_V is the probability of a pattern to correspond to a fixed point that is at least one-neuron off. We expect $p_V > p_B$ because some patterns can still converge to those fixed points but would be considered wrong by the criteria of p_V. (C) The number of patterns that cannot be recovered because their fixed points do not exist or are unstable.

7.2 Computer Simulations

Figure 4. shows the relationship between the pattern-to-neuron ratio (α) and the average error rate (ϵ) from the computer simulations. The results seem to agree well with the theoretical curve in Figure 2. However, computer simulation showed that the average error rate was dependent on the network size N. The maximally achievable error rate, as indicated by the plateaus of the curves in Figure 4, seemed to increase with N. At the same time, the noisiness of the trend decreased with larger N. The high variability at small N is understandable because the error rate is defined as the number of incorrect neurons divided by N. A smaller network would naturally have a more "discrete" error rate. Amit et al. mentioned in their paper (1985) that Weisbuch G. and Posner E. C. had privately communicated with them that, at zero temperature, the patterns would remain locally stable only when $M < N/(2\ln(N))$. This means that the memory capacity α_c is not scaled linear by N, but rather decreases with it. This may explain the curves observed in Figures 2 and 4. Unfortunately, it was unclear on what basis did Weisbuch G. and Posner E. C. came up with the inequality. The private communications were never published.

Once again, the behavior described by Folli et al. (Figure 3) was not observed in the computer simulation results. The error rates did not decrease but rather increased monotonically over increasing α (Figures 2 & 4). It may because they were using a more generalized version of Hopfield network that allowed for autapses. It would be interesting to investigate what role autapses play in increasing the memory capacity.

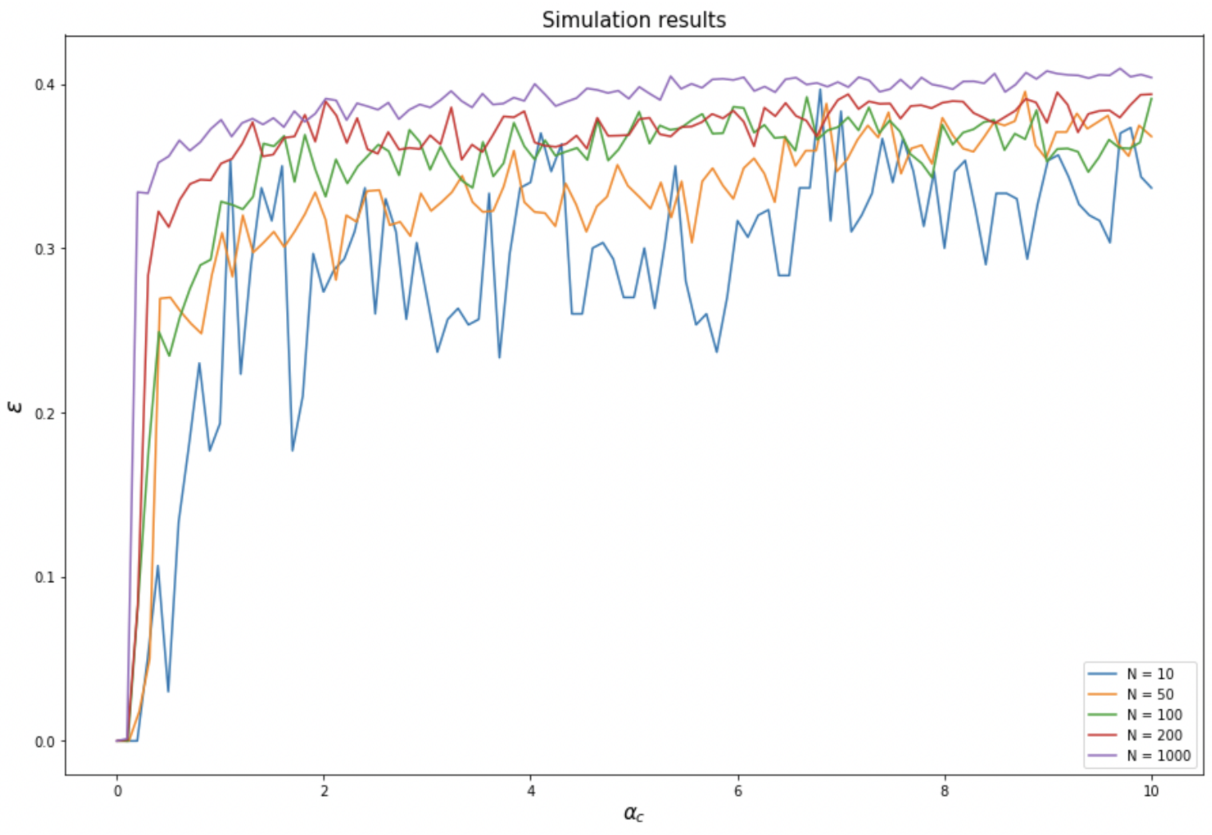

Figure 4. Relationship Between Memory Capacity (α_c) and Average Error Rate (ϵ) from Computer Simulations. Like Figure 2, The error rate increased sharply after around $\alpha_c = 0.1$ and seemed to plateau at $\epsilon \approx 0.35$. However, the maximally achievable error rate seemed to increase with the network size N. See **Appendix** for code.

8 Improvement

In this term paper, for both the theoretical modeling and computer simulations, I assume the patterns to be random and independent or each other. However, we are not sure if memories are stored the same way in the brain. Association memories may be stored ”closer” in the neuronal state space. For example, in Figure 1, the letter T and Y are very similar and therefore would be close in the multi-dimensional neuronal state space. Therefore, even with a pattern-to-neuron ratio of only 0.16, the spurious pattern (a mix of the letters T and Y) actually occurs quite frequently. One way to quantify how far each memorized patterns are from each other is using Hamming distance. It would be interesting to quantify the orthogonality of the pattern set and see what effect it has on the memory capacity.

We found the relationship between the error rate and the memory capacity is dependent on the network size N. This was observed in the theoretical calculation (Equation (7) and Figure 2), computer simulations (Figure 4), and Amit et al.'s (1985) private communications with Weisbuch G. and Posner E. C.. I suspect the low temperature (in fact, $T = 0$) of my model has caused the network dynamics to freeze in place at a local energy minimum, and this effect of temperature may have a dependence on the network size N. Incorporating a none-zero temperature in our model would require some advanced mathematics. Luckily, they are laid out by Amit et al. (1985) and Hertz et al. (1991) in details.

Finally, the Hopfield network introduced in this term paper did not allow autapses, that is, the neurons cannot form connections with themselves (all diagonal entries of the connection matrix **J** are zero). However, autapses are observed in biological networks (Van der Loos and Glaser, 1972). Although their roles are unclear, but a Hopfield network that allows for autapses could be considered to be a more generalized version of the Hopfield network (Folli et al., 2013).

9 Conclusions

In this term paper, I approximate the relationship between the memory capacity and the error rate of a zero-temperature Hopfield network using both the theoretical calculation of the probability of the pattern's local stability and computer simulations of the network dynamics. The theoretical calculation yielded a memory capacity of $\alpha_c \approx 0.15$ (at an error rate of 0.5%), similar to what Hopfield (1982) proposed. The computer simulation resulted in similar results, but the error rates were found to increase with the network size N. For both methods, the error rate increased sharply after $\alpha \approx 0.1$ and plateaued at around 0.35. Nevertheless, the error increase monotonically within the range of pattern-to-neuron ratio (α) covered (from 0.1 to 10). It did not decrease at a large α as observed by Folli et al. (2017).

10 References

Amit DJ, Gutfreund H, Sompolinsky H, Storing Infinite Numbers of Patterns in a Spin-Glass Model of Neural Networks. Physical Review Letters, 55(14):1530-1533 (1985).

Crisanti A, Amit DJ, Gutfreund H, Saturation Level of the Hopfield Model for Neural Network. Europhysics Letters, 2(4):337-341 (1986).

Folli V, Leonetti M, Ruocco G, On the Maximum Memory Capacity of the Hopfield Model. Frontiers in Computational Neuroscience, 10, 144 (2017). doi: 10.3389/fncom.2016.00144

Gerstner W, Kistler WM, Naud R, Paninski L, *Neuronal Dynamics* (2014).

Hebb DO, *The Organization of the Behavior* (1949).

Hertz J, Krogh A, Palmer RG, *Introduction to the Theory of Neural Computation* (1991).

Hopfield JJ, Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences USA 79, 2554–2558 (1982).

Hopfield JJ and Tank DW, Neural computation of decisions in optimization problems. Biological Cybernetics, 52, 141–152 (1985).

Na HS and Park YJ. Symmetric neural networks and its examples. [Proceedings 1992] IJCNN International Joint Conference on Neural Networks. doi:10.1109/ijcnn.1992.287176

Van der Loos H and Glaser EM, Autapses in neocortical cerebri: synapese between a pyramidal cell's axon and its own dendrites. Brain Research. 48:355-360. (1972). doi:10.1016/0006-8993(72)90189-8.

11 Appendix (Python Code)

11.1 Preliminaries

```
import math
import numpy as np
import scipy
import random
import matplotlib.pyplot as plt
from scipy.integrate import quad
```

11.2 Plotting Numerical Solution of Theoretical Model

```
alpha_c_np = np.linspace(0.001,10,200)
error_rates_np = np.zeros((len(alpha_c_np,)))
def integrand(z,alpha_c):
    """The left tail up to z=-1 for z~Norm(0,\alpha_c). Used for the scipy.quad
    general integration function.

    Parameters
    ----------
    z: float
        The Normal distributed variable z~Norm(0,\alpha_c).
    alpha_c: float
        The ratio M/N = the capacity.

    Returns
    -------
    The error rate (float): the probability that a neuron is flipped after the
    neuronal states converges completely to a memorized pattern.
    """
    return 1/np.sqrt(2*np.pi*alpha_c) * np.exp(-(z**2)/(2*alpha_c))

for i, alpha_c in enumerate(alpha_c_np):
    error_rates_np[i] = quad(integrand, -np.inf, -1, args=(alpha_c))[0] # integration

plt.figure(figsize=(6,4))
plt.plot(alpha_c_np, error_rates_np)
plt.xlabel(r'$\alpha_c$',fontsize=15)
plt.ylabel(r'error rate ($\epsilon$)',fontsize=15)
plt.title(r'$\epsilon(\alpha_c) = \int_{-\infty}^{-1} \frac{1}{\sqrt{2 \pi \alpha_c}} e^{-\frac{z^2}{2\alpha_c}}$', fontsize=20)
plt.show()
```

11.3 Computer Simulations

11.3.1 Helper Functions

```
def make_one_pattern(r,c):
    """Makes one numpy array with the given shape (r,c) consisting of a random -1
    and +1."""
    pattern = np.random.randint(2, size=(r,c))
    pattern[pattern == 0] = -1
    return pattern


def permutate_one_pattern(pattern, k):
    """Permute k randomly selected neuron in the pattern"""
    r, c = pattern.shape
    N = r * c
    indicies = np.random.choice(N, k, replace=False)

    permutated_pattern = pattern.copy().flatten()
    for i in indicies:
        if permutated_pattern[i] == 1:
            permutated_pattern[i] = -1
        else:
            permutated_pattern[i] = 1

    return permutated_pattern.reshape((r,c))


def make_connection_matrix(patterns):
    """Makes a square, symmetric matrix (J) that is the sum of the auto-
    correlations of all the patterns. (It uses auto-correlation because of
    Hebbian learning).

    Parameters
    ----------
    patterns: a list of 2d numpy.array

    Returns
    -------
    J: 2d np.array
        A square, symmetric connection matrix (J).
    """
    r, c = patterns[0].shape
    N = r * c
    J = np.zeros((N,N))

    for pattern in patterns:
        pattern_new = pattern.copy()
        pattern_new = pattern_new.reshape((N,1)) # reshape to column vector

        J = J + np.matmul(pattern_new, pattern_new.T) - np.identity(N)

    assert(np.allclose(J, J.T, rtol=1e-05, atol=1e-08)) # check if J is symmetric
    assert(np.allclose(J.diagonal(), 0, rtol=1e-05, atol=1e-08)) # check if all diagonals are zero
    return J/N


def calc_energy(S, J):
    """Calculates the energy associated with the current states (S) of the network.

    Parameters
    ----------
    S: 2d np.array
        Current neuronal states.
    J: 2d np.array
        Connection matrix.

    Returns
    -------
    E: float
        The energy associated with the current states (S) of the network.
    """
    # reshape S to column vector
    r, c = S.shape
    N = r * c
    S_new = S.copy()
    S_new = S_new.reshape((N,1))

    # E = (-1/2) * S^T * J * S
    return -0.5* np.matmul(S_new.T, np.matmul(J, S_new))


def update_network_once(S, J):
    """Update all neurons in the network in a random sequence. The random
    sequence will differ every time you call this function, and all neurons will
    be updated exactly once.

    Parameters
    ----------
    S: 2d np.array
        Current neuronal states.
    J: 2d np.array
        Connection matrix.

    Returns
    -------
    S_new: 2d np.array
        The updated neuronal states S(t + \Delta t).
    """

    def sign_function(h):
        if h >= 0:
            return 1
        else:
            return -1

    # reshape to column vector
    r, c = S.shape
    N = r * c
    S_new = S.copy()
    S_new = S_new.reshape((N,1))

    # shuffle indicies
    randomize_idx = np.arange(N)
    np.random.shuffle(randomize_idx)

    # pseudo-random update (all neurons are updated once in a random sequence)
    for idx in randomize_idx:
        h = np.matmul(J[idx,:],S_new)
        S_new[idx] = sign_function(h)

    return S_new.reshape((r,c)) # reshape back to original shape


def min_bit_diff(S, patterns):
    """Calculates the bit differences between the current state (S) and the
    memorized patterns. Returns the minimum (closest) difference and the idx of the
    pattern that is the most similar to S.

    Parameters
    ----------
    S: 2d np.array
        Current neuronal state.
    patterns: a list of 2d numpy.array

    Returns
    -------
    min_diff: int
        The bit differernce between the current state (S) and the pattern that is
        the most similar to it. If min_diff = 0, it means the current state (S)
        is identical to the pattern.
    pattern_idx: int
        The index of the pattern that is the most similar to the current state (S).
    """
    diff = []
    for i, pattern in enumerate(patterns):
        diff.append(np.sum(np.logical_not(S == pattern)))

    return np.min(diff), np.argmin(diff)


def run_hopfield(r, c, M, num_trials=30, r_seed=0, to_plot=False):
    """
    Simulates multiple trials of hopfield network with specied N (r*c) and M.
    Calculated the average performance (error rates)

    Parameters
    ----------
    r: int
        Number of rows in the patterns.
    r: int
        Number of rows in the patterns.
    M: int
        Number of patterns.
    num_trial: int
        Number of trials to run and average over.
    r_seed: int
        Seed of the random number generator.
    to_plot: bool
        To plot the neuronal states or not.

    Returns
    -------
    average_error_rate:
        The average error rate of the simulations.
    """
    # Initialization
    np.random.seed(r_seed)
    patterns = []
    error_rates = []
    N = r * c

    # Make patterns
    for i in range(M):
        patterns.append(make_one_pattern(r,c))

    J = make_connection_matrix(patterns)

    for trial in range(num_trials):
        E = [99, 89, 79, 69, 59, 49] # intialized with dummy values (deleted later)
        S0 = patterns[np.random.randint(0,M)]
        S = S0.copy()

        if to_plot:
            plt.figure()
            plt.subplot(1,2,1)
            plt.imshow(S, cmap='gray')
            plt.title("S0")

        # Update, and stop if the lastest 5 elements are close
        while not np.isclose(np.max(E[-5:]), np.min(E[-5:])):
            E.append(calc_energy(S, J))
            S = update_network_once(S, J)

        if to_plot:
            plt.subplot(1,2,2)
            plt.imshow(S, cmap='gray')
            plt.title("result")
            plt.suptitle(f"trial #{trial}")
            plt.show()

        # Calculated error
        neuron_diff = np.sum(np.logical_not(S == S0))
        error_rates.append(neuron_diff/N)

    return np.mean(error_rates)
```

11.3.2 Run Simulations

```
N_list = [10,50,100,200,1000]

plt.figure(figsize=(15,10))
for N in N_list:
    print(f"running N = {N}")
    alpha_c = []
    error_rates = []
    for M in np.linspace(1,10*N,100):
        alpha_c.append(int(M)/N)
        error_rates.append(run_hopfield(N,1,int(M)))
    plt.plot(alpha_c, error_rates, label=f"N = {N}")

plt.xlabel(r'$\alpha_c$', fontsize=15)
plt.ylabel(r'$\epsilon$', fontsize=18)
plt.title('Simulation results', fontsize=15)
plt.legend()
plt.show()
```

11.4 Demonstrating Spurious Patterns

```
# patterns
letter_T = np.array([[ 1, 1, 1, 1, 1],
                     [-1,-1, 1,-1,-1],
                     [-1,-1, 1,-1,-1],
                     [-1,-1, 1,-1,-1],
                     [-1,-1, 1,-1,-1]])
letter_O = np.array([[ 1, 1, 1, 1, 1],
                     [ 1,-1,-1,-1, 1],
                     [ 1,-1,-1,-1, 1],
                     [ 1,-1,-1,-1, 1],
                     [ 1, 1, 1, 1, 1]])
letter_N = np.array([[ 1,-1,-1,-1, 1],
                     [ 1, 1,-1,-1, 1],
                     [ 1,-1, 1,-1, 1],
                     [ 1,-1,-1, 1, 1],
                     [ 1,-1,-1,-1, 1]])
letter_Y = np.array([[ 1,-1,-1,-1, 1],
                     [-1, 1,-1, 1,-1],
                     [-1,-1, 1,-1,-1],
                     [-1,-1, 1,-1,-1],
                     [-1,-1, 1,-1,-1]])

# construct network
J = make_connection_matrix([letter_T, letter_O, letter_N, letter_Y])
S0 = make_one_pattern(5,5)

# plot initial condition
plt.imshow(letter_T, cmap='gray')
plt.tick_params(axis='both', left='off', bottom='off', labelleft='off', labelbottom='off')
plt.show()
S_new = S0.copy()


# run this cells to update S
S_new = update_network_once(S_new,J)
plt.imshow(S_new, cmap='gray')
plt.tick_params(axis='both', left='off', bottom='off', labelleft='off', labelbottom='off')
plt.show()


plt.figure(figsize=(20,5))
plt.subplot(1,5,1)
plt.imshow(letter_T, cmap='gray')
plt.tick_params(axis='both', left='off', bottom='off', labelleft='off', labelbottom='off')
plt.title("(A) Pattern 1")

plt.subplot(1,5,2)
plt.imshow(letter_O, cmap='gray')
plt.tick_params(axis='both', left='off', bottom='off', labelleft='off', labelbottom='off')
plt.title("(B) Pattern 2")

plt.subplot(1,5,3)
plt.imshow(letter_N, cmap='gray')
plt.tick_params(axis='both', left='off', bottom='off', labelleft='off', labelbottom='off')
plt.title("(C) Pattern 3")

plt.subplot(1,5,4)
plt.imshow(letter_Y, cmap='gray')
plt.tick_params(axis='both', left='off', bottom='off', labelleft='off', labelbottom='off')
plt.title("(D) Pattern 4")

plt.subplot(1,5,5)
plt.imshow(S_new, cmap='gray')
plt.tick_params(axis='both', left='off', bottom='off', labelleft='off', labelbottom='off')
plt.title("(E) Spurious Pattern")
plt.show()
```