PHY407 Final Project: Associative Memory Using Hopfield Neural Networks and Hebbian Learning

Lyndon Boone (1002384396)

December 18, 2019

1 INTRODUCTION

In this project, I used Hopfield networks as an associative memory structure to remember a small set of images depicting 5 letters of the alphabet. When presented with a new image resembling one of the original images, the network "remembers" the original image that it was trained on and converges to a stable state depicting the original image. To study this, I manually created a dataset of 5 images depicting the letters A, B, C, H, and T, trained the network to "remember" these images, and then presented the network with test images that were corrupted versions of the original images to see if the network could recognize them. To quantify this, I studied the recall accuracy of the network as a function of the amount of noise added to the original image. Source code as well as the dataset I used is attached as part of the Quercus submission, and the code should take roughly 10 minutes to run and reproduce results.

1.1 The Hopfield Network

The Hopfield network was popularized by John Hopfield in 1982 as a model for associative memory [1]. It is a type of fully-connected neural network where each node is connected to every other node in the network, and all connections are bidirectional, meaning that the connections between nodes go both ways [2]. In a Hopfield network, each node is both an input and an output. Typically, the weights of the network (strength of connections between nodes) are trained on a set of *retrieval states*, which in this case are the 5 letters A, B, C, H and T. Once the weights are trained, the network acts as a nonlinear dynamical system. The network nodes are provided

with an input pattern, and the nodes are repeatedly updated according to some update rule until the network converges to a stable state. The update rule works to minimize the "energy" of the system, which is given by

$$E = -\frac{1}{2}\sum_{i,j} w_{ij} s_i s_j \tag{1}$$

where s_i is the state (value) of node i, and w_{ij} is the weight connecting node i to node j. In this project, I used a binary Hopfield network where the states of individual neurons can only take values +1 or -1. The energy function above actually belongs to the class of *Ising models* that we studied in class, but now there is a unique interaction term (w_{ij}) between every node in the network. In the Ising model studied in class, the fixed interaction term between adjacent nodes resulted in the minimum energy state being that of all spins (nodes) having the same binary value. In a Hopfield network, the interaction terms (weights) are trained according to some *learning rule* so that the retrieval states of the network are minima, or *basins of attraction*, of the energy function. In the following subsections, I'll describe the learning rule used to calculate the weights of the network, and the update rule used to update the state of the system to reach a local minima of the energy function.

1.1.1 Learning Rule (Hebbian Learning)

Although there are multiple learning rules that can be used to train Hopfield networks, in this project I used the classic learning rule proposed by Donald Hebb in 1949 for neural networks, which lead to the term *Hebbian learning* [3]. Hebb inspired the popular phrase in neuroscience, *"neurons that fire together, wire together"*, which indicates that the strength of the connection between two neurons should change according to the correlation between the activity of the two neurons [2],

$$\frac{dw_{ij}}{dt} = \text{Correlation}(s_i, s_j) \tag{2}$$

In a binary Hopfield network, this learning rule is implemented in the following way when learning N binary patterns,

$$w_{ij} = \frac{1}{N}\sum_{n=1}^{N} s_i^{(n)} s_j^{(n)}, \quad i \neq j \tag{3}$$

$$w_{ii} = 0 \tag{4}$$

After inspecting this equation, we can see that the weight between two nodes will be strong when the two nodes often have the same binary value in the patterns to be stored. Note that the weights are bidirectional and symmetric, i.e. $w_{ij} = w_{ji}$.

1.1.2 Update Rule

After learning the weights, the network can be used as a nonlinear dynamical system to "remember" a retrieval state after being fed with an input pattern resembling one of the retrieval states. During this phase, nodes in the network are updated according to the rule,

$$s_j = \theta\Big(\sum_{i=1}^{I} w_{ij} s_i\Big) \tag{5}$$

where $\theta(\cdot)$ is a nonlinear activation function. In this project, I used a simple step activation function,

$$\theta(a) = \begin{cases} 1, & a >= 0 \\ -1, & a < 0 \end{cases} \tag{6}$$

The nodes of a Hopfield network can either be updated in a *synchronous* or *asynchronous* fashion, meaning either all at once, or one at a time. Certain properties of the Hopfield network depend on which type of update rule is chosen. In particular, there is a proof that states that the dynamics of the network will always converge to a stable fixed point of the energy function, but this proof depends on the fact that updates are performed asynchronously [2]. For this reason, I chose to only use asynchronous updates, although exploring the difference in network dynamics between the two update rules would be an interesting project in itself. Asynchronous updates go as follows: one node of the network is selected at random and updated according to the rule in equation 5. This is reapeated until all the nodes in the network have been updated once, and then the process is repeated all over again until convergence.

2 METHODS

In this project, I used two experiments to study the capabilities of the Hopfield network. Throughout the project, I used a dataset of five 10×10 images consisting of binary values which depict the letters A, B, C, H, and T, which I created myself (Figure 1). The letters were purposely chosen

Figure 1: The five binary images used as retrieval states for the Hopfield network. Each image has 10 × 10 binary pixel values where yellow is +1 and purple is -1.

to not look similar to each other to make it easy on the network. The original plan for this project was to first investigate the associative memory properties of the network on these five images and then add more to see how it might affect the networks performance, but I decided not to do this in the interest of time. It would have been interesting to add letters like "I" and "G" to the dataset which resemble "T" and "C". The data for the images can be found in the 'templates.npy' file submitted with the assignment. Before both experiments, the network was trained on this dataset to obtain the weights according to equations 3 and 4.

The first experiment involved generating corrupted images from the templates and testing to see whether the network could recognize what template image they were generated from. To generate a corrupted image, I set up a function to go over each pixel in the template image, and flip the binary value with probability p. For the experiment, I tested the network on p values ranging between 0.1 and 0.5 in increments of 0.1. Once the network was trained, I generated 18,444 test images for each letter and each corruption level p to test the network on to see if it would converge to the retrieval state that the test image was generated from. I modeled this as a Bernoulli trial where for each combination of letter and corruption level there was a probability μ of the network associating the test image with the correct retrieval state. My goal was to estimate the parameter μ by calculating the average number of correct associations ν over a series of trials. I used Hoeffding's inequality [4] to calculate the number of samples N required to have ν, the estimate for μ, within $\epsilon = 1\%$ of μ, at a $1-\delta = 95\%$ confidence level ($\delta = 0.05$),

$$\Pr\big(|\nu - \mu| > \epsilon\big) \leq 2e^{-2N\epsilon^2} = \delta \tag{7}$$

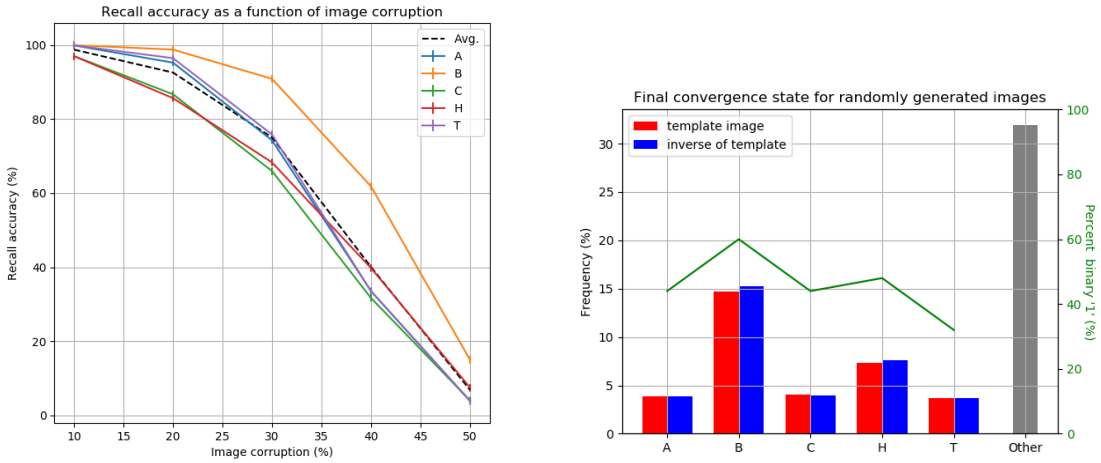
where $\Pr(\cdot)$ indicates probability. Solving this equation for N with $\epsilon = 0.01$ and $\delta = 0.05$ resulted in $N = 18,444$. It should be noted that Hoeffding's inequality can be loose, and its possible that having this many samples may have been overkill. But I wanted to have some robust estimate of what the error was on my estimate for the recall accuracy of the network, and Hoeffding's inequality assures that it is smaller than some amount at a desired level of confidence.

For the second experiment, I wanted to see what state the network would converge to if provided with a completely random input (i.e. not generated from one of the templates). To do this, I generated noisy images where each pixel value had a randomly generated value of +1 or -1 from a discrete uniform distribution. This was accomplished using the numpy *randint* function. This time just because I thought the number was large enough, I again generated 18,444 of these images, fed them into the network, allowed the system to converge to a stable state, and recorded whether the final state was one of the template images or their inverse where +1's and -1's are flipped (note from equation 1 that both the retrieval state and its inverse have the same energy, and should thus both be minima of the energy function). This allowed me to investigate what were the common final convergence states for a completely random, noisy input image.

3 RESULTS

The results of the two experiments described above are shown in Figure 2. Specifically, the left side of the figure shows the recall accuracy of the network as a function of image corruption for the different letter templates. As expected, the general trend is that recall accuracy decreases with increasing image corruption. Some letters, such as the letter "B", consistently have a higher recall accuracy than the other letters (more on this later), but in general the recall accuracy drops off sharply after $p = 0.3$. By the time we reach 50% image corruption, the average recall accuracy across all letters is around 7%. A sample of 5 correctly associated image pairs are shown in Figure 3 for $p = 0.1$ and $p = 0.3$. Examples of correctly associated image pairs for other levels of image corruption are provided in the Appendix.

The right side of Figure 2 shows the results of the second experiment, where pixel values were generated randomly from a discrete uniform distribution consisting of values +1 and -1. The objective of this experiment was to see if certain retrieval states were favored as a final convergence state when starting with a very noisy input. Indeed, the distribution of final convergence states was not uniform across the letters, as shown by the bar chart in Figure 2.



(a) Experiment A

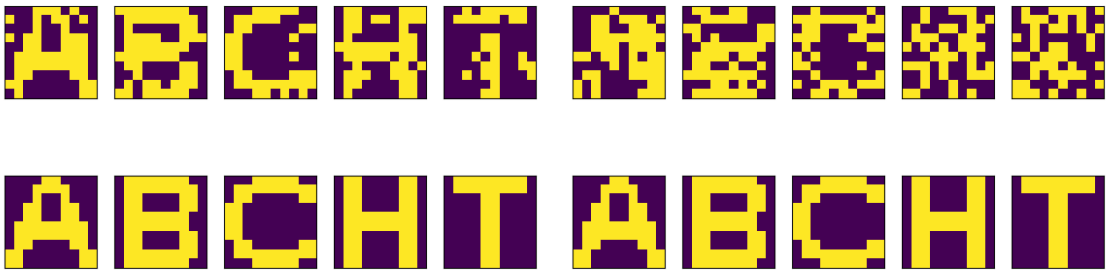
(b) Experiment B

Figure 2: Left: Recall accuracy as a function of image corruption in the first experiment. Error bars indicate a 95% confidence interval (as described in methods section). Right: Converged state frequency from the second experiment involving randomly generated images from a random discrete distribution. The green line indicates the percentage of the template image occupied by +1 pixel values.

The letter "B" (and its inverse) occurred much more frequently as a final convergence state than all the other letters, with "H" coming in second place and the other letters occupying the rest of the distribution with a similar low frequency. This seems to be related to the number of +1 bits in each template image, which is shown by the green line plotted above the bar chart on a separate axis indicating the percentage of each template image occupied by "+1" pixel values. The large grey bar in the figure corresponds to "other" final convergence states besides the template letters and their inverses. This "other" bar takes up a little over 30% of the distribution, and corresponds to "spurious" patterns which could also be local minima of the energy function and often resemble some mix of the desired retrieval states. A selection of these patterns are shown in Figure 4.

4 Discussion

The results of the second experiment indicate that there is a correlation between the number of +1 pixel values in the pattern of a retrieval state and the frequency with which a randomly generated pattern converges to that state. This could have some underlying mearning, or perhaps



(a) $p = 0.1$ **(b)** $p = 0.3$

Figure 3: Selection of correctly associated image pairs from the first experiment. Images on the top row were fed into the Hopfield network, which subsequently converged to the state corresponding to the images on the bottom row. The images on the top row were generated from corrupting a fraction p of the pixels in the template images.

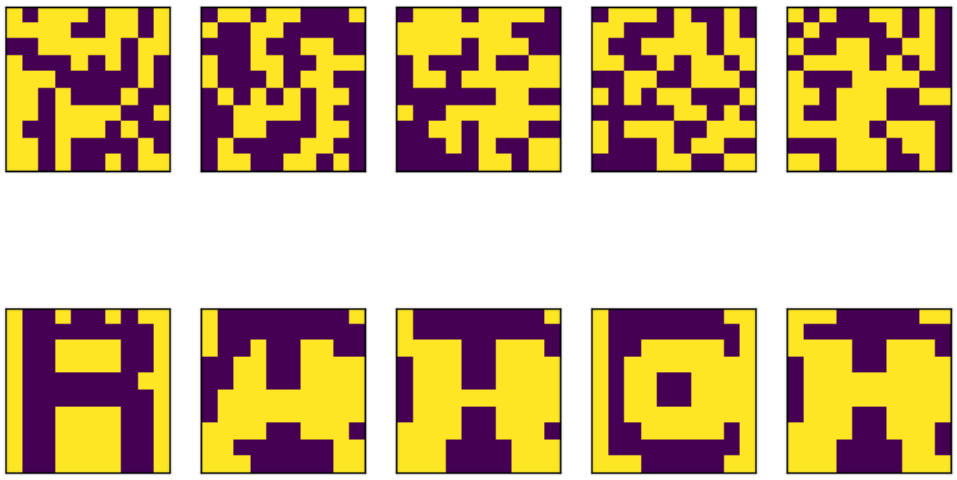


Figure 4: Selection of randomly generated images resulting in a final converged state with a "spurious" pattern. Several of the spurious patterns resemble mixtures of the retrieval states and their inverses.

it is just a coincidence and a more likely reason for certain patterns having a higher convergence frequency is that some patterns may have more similarities to others. For example, the letter "H" and the inverse of letter "T" seem to combine quite well (Figure 4) and this could indicate why the letter "T" had a relatively high convergence frequency for the number of +1 pixel values it has in its template pattern. Another spurious pattern in Figure 4 seems to be a mix of "B" and "H". It would be an interesting project to investigate what causes certain patterns to be more common attractors in the energy function than others. This observation from the second experiment also is in good agreement with the observation from the first experiment that "B" has a consistently higher recall accuracy.

It would be interesting to compare the results of the first experiment to human performance in guessing which of the letters the corrupted image was generated from. In Figure 3, five correctly associated image pairs are shown for $p = 0.1$ (one for each letter). From these images, it is safe to assume that a human would likely get all of these associations correct at the level $p = 0.1$. On the other hand, I suspect that a human would have difficulty achieving the 75% average recall accuracy exhibited by the network on images with 30% pixel corruption (a sample of these images are also shown in Figure 3). I won't go into a deep discussion here, but it would be interesting to investigate the difference between how humans perform on this task in comparison to Hopfield networks.

To sum up, this project demonstrated a working Hopfield network and its ability to function as an associative memory structure. In addition, two experiments were performed to investigate the recall accuracy and convergence properties of the network. The recall accuracy experiment showed that the network is capable of associating highly corrupted images with similar-looking "memories" preserved in the weights of the network, and the random image experiment showed that certain retrieval states were more common than others as a final converged state of the network after input of a noisy image. This was just a small project, and there are surely a great number of further investigations that could be done, as I'm sure there have been in the literature. But even this project raised some questions that could lead to further pursuit, such as what makes certain retrieval states more likely to be a final convergence state (this is probably well understood but would require some digging in the literature), and how does human performance on the corrupted image recognition task compare to the Hopfield network. All in all, this was an enjoyable final project and I'm glad I got the opportunity through this course to explore a topic I was interested in and actually try implementing it in code.

REFERENCES

[1] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[2] D. J. MacKay, *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[3] D. O. Hebb, *The organization of behavior*, vol. 65. Wiley New York, 1949.

[4] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning from data*, vol. 4. AMLBook New York, NY, USA:, 2012.

APPENDIX A - EXTRA FIGURES

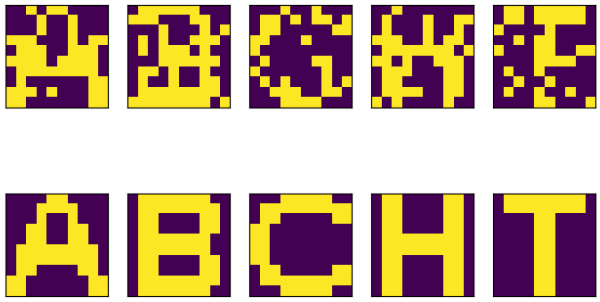


Figure 5: Selection of correctly associated image pairs for corruption level $p = 0.2$.

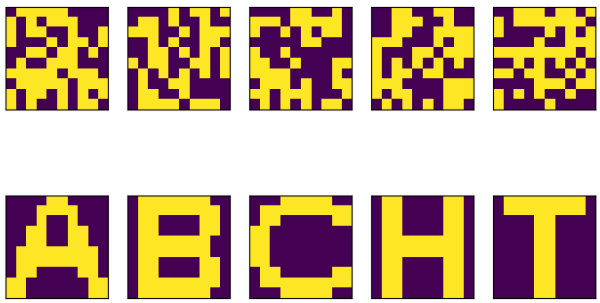


Figure 6: Selection of correctly associated image pairs for corruption level $p = 0.4$.

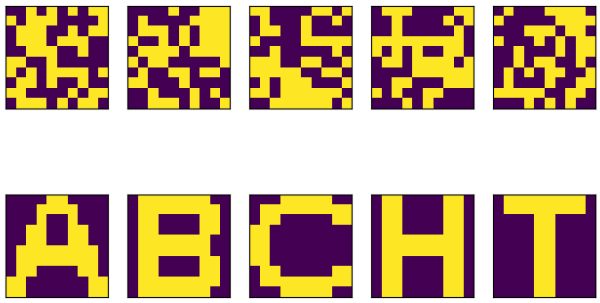


Figure 7: Selection of correctly associated image pairs for corruption level $p = 0.5$.