Modeling Cognitive Pattern Recognition Using Hopfield Neural Networks

Joseph Christianson, Wilbert Lam, Neeraj Joshi

March 12, 2016

Abstract

Hopfield Neural Networks are a simple model of biological cognitive processes. In this paper we introduce how the Hopfield Neural Network operates. We analyze the dynamics to show how information that is learned is stored in the stable points; we demonstrate how to build one from simply matrix products; and, we test our resulting model on the famous MNIST dataset. We were able to achieve 67% percent accuracy on the test data using Hopfield networks trained on pairs of the digits, arranged in a bracket, tournament style, structure.

1 Problem

The goal of this project is to create a model that can allow a machine to translate handwritten digits into their corresponding ASCII values. This will begin to solve the problem of translating handwritten documents to bits that a computer can understand and process. The model's primary goal is to be able to distinguish digits despite variation and noise in the handwritten input. We will be using the MNIST database of handwritten digits, which includes a training set of 60,000 examples and a test set of 10,000 examples. Due to the scope of this project, we will be limiting our testing and training data to a smaller subset of the digit images.

Other machine learning algorithms have been developed to recognize the inputs from the MNIST database into recognized digit outputs. Looking at the error rate for our testing set, we can compare and see how well our model does compared to the other machine learning algorithms implemented. A list available on the MNIST database homepage details different classifiers, the preprocessing done for each and their corresponding test error rate. The Hopfield Network has not been listed, which will allow us to contribute our findings that are novel to the database.

2 Assumptions

Before we create our Hopfield network, there are some assumptions and simplifications that we made to make our model more reasonable and feasible given the limitations of our available time and computing power.

Our neural network model is based upon the Hopfield network model. As a result, this is a fully recurrent network where each individual node is interconnected with every other possible node in the network (with the exception of itself), creating a collection net of directed cycles. Therefore each node will be both an input and output, which contrasts to other types of neural nets such as feedforward.

Nodes are updated randomly at each state asynchronously. Because of the nature of recurrent networks, each updated node relies on the previous state of the network. This echoes practical biological network models.

Our dataset is drawn from the MSINT dataset by Yann LeCun. It includes purely single digit numerical values, from 0 to 9, where all images are intended to contain only one number in each frame. The images from the dataset are centered and initially set to both pixel height and pixel width of 28, but we have truncated 4 pixels from the outside each image to reduce images to side length of 20 pixels (these pixels are almost all purely unnecessary with regards to the uniqueness of each image and are truncated to save computation time and space).

An additional simplification is that we only process individual single digit numbers. Our model is designed to learn and recognize a single digit at a time rather than a series of numbers. With this in mind, our project could potentially be expanded to include more digits if we had a system that could split an image of a many digit number into individual digits. Then each of these digits could be run through our model and combined digit by digit. However, due to time limitations and our focus on the modeling aspect, that is not a feasible option for this project.

Each node in our network will correlate to a single individual pixel for each image. This provides us a clean way of identifying our network nodes that remains consistent throughout all the images (all images will thus contain the same number of pixels and thus same number of nodes).

Each image is in grayscale, with intensity ranging from total absence of intensity 0 (white) to a total presence of intensity of 1 (black). This allows us to avoid having to worry about color and superfluous variations in each image, as well as allowing us to easily translate each image into high contrast through a threshold bound.

Another assumption we have made is that all images are not truly a random collection of pixels of varying intensities. Each image is pulled from handwriting taken from real-life users and can thus be correlated to an intended result. This guarantees our model is learning from real data.

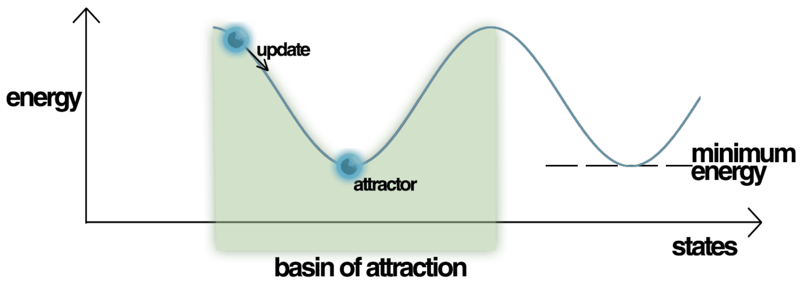
3 Model

In order to solve our problem, we will utilize some of the concepts and ideas from artificial neural networks, a family of models inspired by biological neural networks. Artificial neural networks serve as systems in which weighted input and output is exchanged between neurons to produce desirable outcomes from a set of specified inputs. The connections between neurons have numeric weights which signify the importance of the inputs and outputs translated, and allows the neural network to be adaptive and trainable for different input-output systems.

What separates the Hopfield network from other neural networks is its ability to retrieve certain patterns based on inputted cues. This is called Content Addressable Memory (CAM) and allows for noisy and often varied input cues to still retrieve the correct and desired patterns. To use the network, a pattern must be entered, setting certain nodes to specific numeric values, which then allows the network to update synchronously or asynchronously, finally arriving at a previously trained pattern. We chose to use the method of asynchronous random updating which consists of updating neurons immediately in a random order. When training a Hopfield network, certain patterns are stored within the weight matrix by recognizing the strength and weakness of the connections present between each of the nodes in the input. The dynamics of the network cause the inputted cues to converge to certain trained patterns in the weight matrix.

As stated above, there are certain restrictions that must be placed when determining the weight matrix of the network. The matrix must be symmetric, namely $w_{ij} = w_{ji}$, where w_{ij} represents the weight from neuron i to neuron j. Also, the matrix must not contain any self-connections ($w_{ii} = 0$). To determine what the weight value from a neuron i to neuron j would be, we chose to implement Hebb's Rule which defines $w_{ij} = x_i \times x_j$. A weight between two neurons that is positive will result in the neurons driving each other in the same direction of the value they currently possess. On the other hand, a negative weight between two neurons will drive each other in opposite directions which makes intuitive sense because they do not originally share the same value. This is the notion of "fire together, wire together" that is often associated with the Hebb Rule. Placing the two conditions on the weight matrix and implementing the weight between two neurons to be $w_{ij} = x_i x_j$ ensures that the network converges to a certain pattern in the network. This results directly from the value of the energy of the network.

Now energy can be defined for each node as $E_i = -\frac{1}{2}h_i x_i$. Ideally, the patterns we want to match with occur when the entire network's energy is at a minimum state. That is we will have a stable state where the sum of all energies $\sum_i -\frac{1}{2}h_i x_i = \sum_{ij} -\frac{1}{2}w_{ij}x_i x_j$ for each node in the system is of low energy. In our case, we iterate a given number of times over the network to a point where we achieve reasonable stable states at which the network's energy is then minimized.



In a Hopfield network, all energy in the system will decrease if values of the node changes. We can prove this by the following:

Now

$$E(t) = \sum_{ij} -\frac{1}{2} w_{ij} x_i x_j = \sum_{ij, i \neq p, j \neq p} -\frac{1}{2} w_{ij} x_i x_j - \frac{1}{2} \sum_j w_{pj} x_p x_j - \frac{1}{2} \sum_i w_{ip} x_i x_p$$

. If we find $E(t+1) - E(t)$, we can find calculate ΔE and show that the energy change decreases.

We find $E(t+1)$ and let our new x_p be x_p^*.

Then

$$E(t+1) = -\frac{1}{2} \sum_{ij, i \neq p, j \neq p} w_{ij} x_i x_j - \frac{1}{2} \sum_j w_{pj} x_p^* x_j - \frac{1}{2} \sum_i w_{ip} x_i x_p^*$$

. Since $\Delta E = E(t+1) - E(t)$, we can solve ΔE below:

$$\Delta E = -\frac{1}{2} \sum_j w_{pj} x_p^* x_j - \frac{1}{2} \sum_i w_{ip} x_i x_p^* + \frac{1}{2} \sum_j w_{pj} x_p x_j + \frac{1}{2} \sum_i w_{ip} x_i x_p$$

. By symmetry of the weight matrix, we find

$$\Delta E = \sum_i w_{pi} x_i (x_p - x_p^*)$$

. Now when we update there are two possibilities:

$$x_p : -1 \to 1$$

$$x_p : 1 \to -1$$

If we plug in the values for x_p and x_p^* into /triangle E, we find that in the first case $x_p - x_p^* = -2$, but since this can only occur when $\sum_i w_{pi} x_i$ is positive, then $\Delta E < 0$. The second case is swapped, and thus we also have $\Delta E < 0$.

Thus ΔE is always negative, and so energy in the system can only decrease.

4 Solution

Our solution had several steps. First we preprocessed the data by importing and calculating an average value for each pixel. We normalized these values so that they fell between 0 and 1. We used a threshold value to discretize the pixel values. 0.4 was chosen somewhat arbitrarily, because it was the largest value that left each number as a single, connected figure. Pixels above the threshold were set to 1 and pixels below to -1. We then transformed the 20×20 pixel arrays into 400×1 pixel vectors. These average digits can be seen in figure 1.

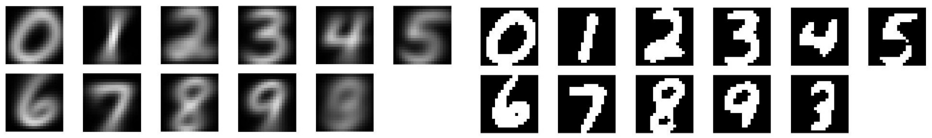


Figure 1: The average values calculated for each digit, followed by the overall average. The left shows the calculated values and the right shows them after being discretized by a threshold value.

Our first attempt at processing the data was to create a single Hopfield network and train it on the 10 image averages that we had calculated. This resulted in zero percent accuracy because a spurious minimum overpowered the trained patterns. These results can be seen in for ten random images in figure 2. The value they are converging to something similar to the overall average of every training digit. This makes sense because the digits are relatively similar patterns, when too many are put into one net, they can collapse to an average.

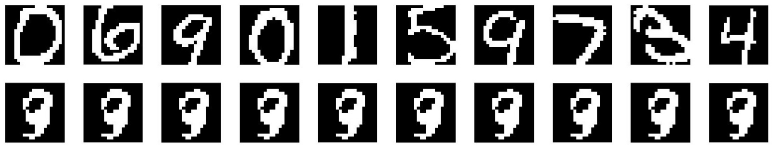


Figure 2: Ten random digits all converge to the same pattern. Investigation revealed that this pattern is the average of the individual digit patterns.

We then decided to try a different path. Rather than a single net that stores all the patterns, we would have multiple nets where each one was responsible for only two patterns. This method required $\begin{pmatrix} 10 \\ 2 \end{pmatrix} = 45$ nets. We would use a bracket-style system to discern the digit in the input vector. Because we used this bracket method, we only had to evaluate the convergence of nine separate Hopfield nets. We were able to correctly discern the digits on around 67% of the MNIST test data.

It is this most successful solution that we will describe in detail, by breaking down crucial sub-routines. The first is `initializeNet.m`

```
%% Parameters
inputPairs = combnk([1:10], 2);
numNets = size(inputPairs,1);
nodes = 400;

%Create Flat Net (3D Array)
weights = zeros(nodes,nodes,numNets);
neurons = zeros(nodes, 1, numNets);

%% learnPatterns
for netItr=1:numNets
    patterns = DiscreteImgs(:,inputPairs(netItr, :));
    for outItr=[1:nodes]
        for inItr=[outItr:nodes]
            if outItr ~= inItr
                val = sum(patterns(inItr, :).*patterns(outItr, :));
                weights(outItr, inItr, netItr) = val;
                weights(inItr, outItr, netItr) = val;
            else
                weights(outItr, inItr, netItr) = 0;
            end
        end
    end
end
```

`initializeNet.m` is responsible for the training of the Hopfield nets. It holds onto the weights and neurons of these nets as three dimensional arrays. Indexes one and two denote the element in a weight or neuron matrix and the third dimension can be thought of as an address for individual neural nets.

We chose to represent stimuli as 1 for on, and -1 for off. Therefore, as mentioned above, for a list of n patterns P, we can set the value of each weight w_{ij} as such:

$$w_{ij} = \sum_{k=1}^{n} (p_k)_i (p_k)_j, \quad i \neq j \qquad w_{ij} = 0, \quad i = j$$

where $(p_k)_i$ is the element at index i for pattern k.

The next sub-routine we wish to show is `processInput.m`

```
% Subroutine - For a set image and pre-initialized nets, updates the
% neurons for a set amount of iterations.

for netNum = netNums
    neurons(:,1,netNum) = image;
end

for iterations = 1:itrCount
    neuronNum = ceil(400*rand());
    for netNum = netNums
        neurons(neuronNum, 1, netNum) = ...
            biasFunc(weights(:, neuronNum, netNum)'*neurons(:,1,netNum));
    end
end
```

In `processInput.m` we describe how the Hopfield networks respond to stimulus. Before it is called, a vector is assigned to `netNums`. In the bracket structure multiple networks are responding to the same stimulus at once; the digit they converge to determine which networks are selected for the next round.

A single iteration involves choosing a neuron at random (`neuronNum`), then taking the inner product of the corresponding column in the `weights` matrix and the `neurons` vector, and finally passing that value through `biasFunc`. `biasFunc` is simply:

$$f(x) = \begin{cases} 1 & : x \geq 0 \\ -1 & : x < 0 \end{cases}$$

Conditions for stopping could be no change in the network energy, or no change in the neurons for several iterations. However, we opted to simply set an iteration count after which we observed convergence. (`itrCount = 2000` is set elsewhere).

Figure 3: A inputed four converges to the average four over 2000 iterations of a Hopfield net trained to recognize fours and nines. The images are in 200 iteration increments.

The final sub-routine we wish to focus on is `determineResults.m`.

```
% Subroutine - After neurons have converged, figures out with digit average
% they converged to. Stores the decisions in a reults vector.

results = [];

for resultsItr =netNums %netNums
    output = neurons(:,1,resultsItr);
    digits = combVec(resultsItr,:);

    [M,I] = min([sum(abs(output - DiscreteImgs(:,digits(1))));
        sum(abs(output - DiscreteImgs(:,digits(2))));
        sum(abs(-1*(output) - DiscreteImgs(:,digits(1))));
        sum(abs(-1*(output) - DiscreteImgs(:,digits(2))))]);
    results = [results,...
        (digits(1)*(I == 1 || I == 3) +...
        digits(2)*(I == 2 || I == 4))];
end
```

After the stable state for the neurons has been calculated, this routine determines which pattern each of the nets converges to. Because we chose to have an iteration count be our condition for convergence, the pattern may be off from the true average pattern by a few pixels. Even though our neuron vectors are ones and negative ones, rather than ones and zeros, we can still calculate what is effectively the Hamming Distance. Each net is only responsible for two patterns, however, when a pattern is trained in a Hopfield network, the inverse image is also implicitly trained. Therefore, we have to check the distance for both the output to each pattern and the inverse of the output to each pattern. We take the pattern that had the minimum Hamming Distance of these four comparisons and set that as our result for that network.

5 Improvements

The goal of this project was to read individual numbers and classify them. Naturally, given the limited scope of reading numbers, we could further expand the classifier to identify letters as well. At this point in time that would have proven too difficult to do with enough accuracy (especially given the similarities between some letters and numbers), but that would be a logical next step. Due to the fact that this was intended to be a mathematical modeling project, we limited our source of images to be single letter images. As stated earlier in the paper, we could provide more functionality by allowing the images to be multiple digit numbers, with an added mechanism to divide up letters and then run each letter through our model. As this is somewhat out of the purpose of our project, we chose not to implement this. However, for purposes of fleshing out and making this project more applicable to the real world, this would be a powerful improvement.

In the realm of how we implement the project, several further improvements could be made. We could do some more pre-processing on the averages, perhaps by subtracting the average or a more complex operation like principal component analysis. Perhaps that would allow us to store more information is a single network. In regards to the network operation, we could explore different methods for choosing which node fires. Finally, rather than organizing the networks into a bracket, we could explore applying them all to the input and taking "votes" from the results of each network.

6 Conclusion

While we did not achieve a high level of accuracy, we were able to provide a working implementation of Hopfield's neural network model. We verified that information can be stored in the weights of the networks and that it can be retrieved given certain stimuli. Spurious minima proved to be a significant issue which limited how many patterns could be trained into a network. Biological processes are not perfect and so we expected to deal with large inaccuracies when modeling the complex process of associative memory. While data processing accuracy was not the best, the Hopfield network displayed accurate representations of biological memories.

7 References

"Hopfield Networks." A Hopfield Net Example. Web. 09 Mar. 2016.

DeKampfs, Mark. "Hopfield Networks." Hopfield Networks. Leeds University. Web. 09 Mar. 2016.

"Notes on Neural Networks." Notes on Neural Networks. Web. 09 Mar. 2016.

"Hopfield Network." A Simple Neural Network. Web. 09 Mar. 2016. Energy Landscape. Digital image. Energy Landscape. Wikipedia, n.d. Web. 9 Mar. 2016. `<https://en.wikipedia.org/wiki/Hopfield_network#/media/File:Energy_landscape.png>`

8 Appendix

Here we include scripts required to run the model, but that do not provide insight into the mathematical model. The can be accessed at

https://github.com/Kristjansson/HopfieldNet/

8.1 main.m

```
%% Initialize data
clearvars -except weights

load('ImgAvgs.mat', 'ImgAvgs');
threshold = .4;
DiscreteImgs = (ImgAvgs > threshold) - (ImgAvgs <= threshold);

disp('Data Loaded...')
%% Create A "Flat" Hopfield Net
LookupTables; %generates tables and vectors for going back and forth between
              %neural net indexes and digit combos

FlatNet; %This gives us 'neurons' and 'weights' and trains them with DiscreteImgs
disp('Net Initialized...');
%% LoadTestingData
testDataSize = 100;
[Imgs_Test, Labels_Test] = readMNIST('C:\Users\Joseph\Documents\AMATH 383\FinalProject\t10k-images.i
Imgs_Test = reshape(Imgs_Test, [400, 1, testDataSize]);
DiscreteImgs_Test = (Imgs_Test > threshold) - (Imgs_Test <= threshold);
disp('Test Data Loaded...')

%% Digit Detector Using Bracket Method

correct = 0;
itrCount = 2000;
for imageItr=1:testDataSize
    disp(sprintf('Results for image: %d', imageItr));

    %First Bracket Level
    image = DiscreteImgs_Test(:,1,imageItr);
    netNums = [1 18 31 40 45];
    processInput;
    determineResults;
    disp(sprintf('\t\t1st Round: [%d %d %d %d %d]', results));

    %Second Bracket Level (01)vs(23) and (45)vs(67) with (89) getting a by.
    netNums = [combTable(results(1), results(2))...
        combTable(results(3), results(4))];
    processInput;
    got_A_by = results(5);
    determineResults;

    disp(sprintf('\t\t2nd Round: [%d %d] with %d on a by', results, got_A_by));

    %Third Bracket Level (457)vs(89) with (0123) getting a by.
    netNums = [combTable(results(2), got_A_by)];
    processInput;
    got_A_by = results(1);
    determineResults;

    disp(sprintf('\t\t3rd Round: [%d] with %d on a by', results, got_A_by));

    %Fourth Bracket Level (0123)vs(45789)
    netNums = [combTable(got_A_by, results)];
    processInput;
    determineResults;

    disp(sprintf('\t\t4th Round:%d', results));
    if Labels_Test(imageItr) == (results - 1)
        disp('Correct!');
        correct = correct + 1;
    else
        disp(sprintf('Incorrect. \t Answer: %d', Labels_Test(imageItr) + 1));
    end
    disp('----------------------------------------');
end

disp(sprintf('Accuracy: %d%%', 100*correct/testDataSize));
```

8.2 CalMNIST.m

```
%%Calculate the averages for eacb digit
[imgs, labels] = readMNIST(...
    'C:\Users\Joseph\Documents\AMATH 383\FinalProject\train-images.idx3-ubyte',...
    'C:\Users\Joseph\Documents\AMATH 383\FinalProject\train-labels.idx1-ubyte',60000, 0);
load('ImgAvgs.mat', 'ImgAvgs');
ImgAvgs = zeros(20*20,10);
ImgCounts = zeros(1,10);
for itr=1:60000
```

```
    num = labels(itr) + 1;
    temp = reshape(imgs(:,:,itr), [20*20,1]);
    ImgAvgs(:, num) = ImgAvgs(:, num) + temp;
    ImgCounts(1,num) = ImgCounts(1, num) + 1;
end
ImgAvgs = ImgAvgs./(ones(400,1)*ImgCounts);
save('ImgAvgs.mat', 'ImgAvgs');

OverallAvg = ImgAvgs*ones(10,1)./10;
```

8.3 LookupTables.m

```
combTable = ...
    [0 1 2 3 4 5 6 7 8 9;...
    0 0 10 11 12 13 14 15 16 17;...
    0 0 0 18 19 20 21 22 23 24;...
    0 0 0 0 25 26 27 28 29 30;...
    0 0 0 0 0 31 32 33 34 35;...
    0 0 0 0 0 0 36 37 38 39;...
    0 0 0 0 0 0 0 40 41 42;...
    0 0 0 0 0 0 0 0 43 44;...
    0 0 0 0 0 0 0 0 0 45;...
    0 0 0 0 0 0 0 0 0 0];
combTable = combTable' + combTable;
combVec = combnk([1:10], 2);
```

8.4 `readMNIST.m`

```
function [imgs, labels] = readMNIST(imgFile, labelFile, readDigits, offset)
```

This function was written by Siddharth Hegde, and distributed under the BSD License[1]