

## PS8: Content Addressable Memory

For the purpose of this assignment, we were given parts of a Hopfield neural network designed to correct erroneous memories (cues) of images. We were provided code that would read in correct bitmap images to a memory bank using the sum of outer products of themselves. We were to write code that could fix a provided erroneous bitmap image (cue) using the provided memory bank and some randomness. The process to fix the erroneous memory is the main part of the Hopfield network that were desired to implement. The reason that the Hopfield network can be applied to correct erroneous memory is that it acts as a content addressable memory. Given properties of content addressable memory, one can use outer products of the sum of correct memories to retrieve the correct representation of a memory (or image) given some time and randomness along with a bipolar activation function.

I modified the provided image reading code in order to make it handle multiple images easier. I made a function called “add\_memory\_to\_matrix” that will take an input image file name in the “bitmap” directory and a matrix of memory bits to complete all the necessary tasks to return an encoded, diagonally-suppressed sum of outer products of the previous and new images in a matrix memory bank. First, it reads in the image and converts it from a matrix to a vector. Second, the outer product of the vector is calculated to return a 512x512 matrix. Then a matrix diagonal suppressant (1’s on diagonal, 0’s elsewhere) is subtracted from the outer product of the read image. Having a suppressed diagonal makes correcting erroneous bits much easier in the Hopfield network. The memory is finally added to the matrix memory bank and returned to the calling program.

I modified the code provided in the “content\_addressable\_memory.m” file and refactored the error correction loop into a function called “error\_correction\_strategy.” I created a list of strings of the file names for error cues and have these called upon in an error correction loop. Each file will be written to the “output” directory with the same file name as input. Output will be the corrected images, a product of the provided input matrix memories in the memory bank.

A bipolar activation function works well in our case being that the images are monochrome bitmaps and are easy to deal with in terms of -1 and 1. Really, the guts of the Hopfield network consist of the memory bank that stores the correct memories and the function that corrects the erroneous bits given the memory bank and some randomness. The way I designed the function to correct an erroneous node was as follows:

1. Pick a random node index between 1 and the length of the recall vector (512) in our case
2. Initialize delta to the value of the pixel at that random node index
3. Assign the output of the signum function on the product of the matrix memory vector at that random node and the recall vector to the recall vector at the random node index
4. Assign delta to the absolute value of the difference between delta and the recall vector at the random node index divided by two

This function is called within a loop that will update for the max number of updates assigned. This loop runs within another loop that will run for the max number of iterations assigned. I have found that 100 is a good number of max iterations and 30 is a good number of updates. These

values could obviously be interchanged or made into one loop altogether. The reason I kept them separate was to be able to visualize the changes after every 30 or so updates for debugging purposes when using the pause function.
















#### Number of Recoverable Images:

I have found that I can encode and recover up to 5 memories perfectly from flawed cues.

The memories that I was able to recover perfectly given the completed Hopfield network were the following:

1. clubspade.bmp
2. handheart.bmp
3. happyworld.bmp
4. printtrash.bmp
5. winhelp.bmp






















When testing, I found that leaving out “computersum.bmp” and “notespell.bmp” were necessary for the rest of the images to be recovered. This phenomenon has to do with the premise of orthogonality between memories in the Hopfield network. A Hopfield network requires that memories be quite orthogonal to each other or they will wipe each other out. When the memories wipe each other out, this means that those memories will only be partially recoverable at best. With this in mind, here are the results of the content addressable memory algorithm on each of the erroneous cues of these images, provided that all but the two mentioned images were in the memory bank:

Image Name	Erroneous Image	Corrected Image	Actual Image
clubspade.bmp			
handheart.bmp			
happyworld.bmp			
printtrash.bmp			
winhelp.bmp			



















One can see that the corrected images are exactly the same as the actual images. The reasoning behind this is that all of these images are above a certain threshold of orthogonality which is optimal for the Hopfield network’s maximum benefit. I have found that ~35% (~0.65) is the optimal ratio for this when comparing angles between two images with a cosine function. A zero output from the cosine function means the two images are fully orthogonal. We desire the orthogonality that divides the images into separate areas of the matrix memory stored in the Hopfield network. We do not wish to wipe out memories that overlap with condensing values. The images of “computersum.bmp” and “notespell.bmp” were closely related to each other. These images had a low orthogonality with the other images in the provided set.

#### Intermediate Steps to Perfect Convergence:

First we have my initial results of reading all memories into the memory bank of the Hopfield network.

Image Name	Erroneous Image	Corrected Image	Actual Image
clubspade.bmp			
computersum.bmp			
handheart.bmp			
happyworld.bmp			
notespell.bmp			
printtrash.bmp			
winhelp.bmp			

These results can be viewed as partially converged. Two of the corrections have converged to the correct image and some have not. The ratio of correct to incorrect images is unacceptable using all possible memories in our memory bank. We can see that some memories are wiping others out due to orthogonality. The two images that were corrected must have been most orthogonal from the others in the memory bank. We must continue to remove one memory from the bank. We will choose the computer sum memory because it is less orthogonal to the others when compared to the rest.

Image Name	Erroneous Image	Corrected Image	Actual Image
clubspade.bmp			
computersum.bmp			
handheart.bmp			
happyworld.bmp			
printtrash.bmp			
winhelp.bmp			

We are getting closer to the correct results but we have not attained them yet. We can see that all of the images except “happyworld.bmp” have nearly converged. If we remove the “notespell.bmp” image from our network’s memory, we will attain the perfect results presented above. We can see that elimination of the two least orthogonal memories from memory bank has allowed the Hopfield network to converge to the correct solutions.

In conclusion, we have found that Hopfield networks successfully work as content addressable memories so long as the memories have a certain threshold of orthogonal encodings. More memories can be stored provided greater orthogonality. We see that the Hopfield network operates correctly in our situation when the encoded memories are quite orthogonal (~35%). We conclude that the Hopfield network is sensitive to the orthogonality of memories, giving it an “all or nothing” ability to fix erroneous memories with an encoded and correct memory matrix.