

Implementation of HopField Neural Network for generating complete pattern using Hebbian Learning Algorithm

-AN EXPERIMENTAL APPROACH



INTERNATIONAL INSTITUTE OF INFORMATION
TECHNOLOGY, HYDERABAD
MAY, 2020

RISHAB(2019201050) | AMAN DEV NAUTIYAL(2019201095) |
DIVYANSH SHRIVASTAVA(2019201048)

INTRODUCTION

Learning is to associate two events with each other. In the Hebbian type of learning, both presynaptic and postsynaptic neurons are involved. The main brain organ for learning/explicit memory is the hippocampus (of the limbic system) using Hebbian type.

An event in the hippocampus is sculpted by a group of firing neurons. In Hebbian learning, synaptic modification only occurs between two firing neurons. Human memory thus works in an associative or content-addressable way. There is no location in the neural network in the brain for a particular memory say of an individual.

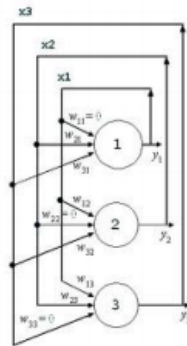
The memory of the individual is retrieved by a string of associations about the physical features, personality characteristics and social relations of that individual, which are dealt with by different parts of the brain.

- In this project we aim to store patterns in a Hopfield Network using the Hebbian Rule.
- This is an experiment based project .We show how the model performs in different types of conditions, which patterns are stored the best, which patterns tend to confuse the model, which pattern doesn't let the model converge at all.
- The patterns we tried are emoji's (facial emotions expressed in an animated form), Bugs Bunny in 3 different poses ,2 human faces and digits from 0 to 9 and have presented the results for them.

We use a Hopfield Network for our task. It is a recurrent network composed of a set of n nodes and behaves as an auto-associator (content addressable memory) with a set of K patterns $\{y_k ; k=1..K\}$ stored in it.

The network is first trained in the fashion similar to that of Hebbian Learning , and then used as an auto-associator. When the incomplete or noisy version of one of the stored patterns is presented as the input, the complete pattern will be generated as the output after some iterative computation.

An example Image of a 4 unit Hopfield Network is shown below:



PREVIOUS WORK:

1. *Storage Capacity of the Hopfield Network Associative Memory by Yue Wu, Jianqing Hu, Wei Wu, Yong Zhou, K.-L. Du*

Presented as a Conference Paper in 2012, they demonstrated their investigation of the various aspects of the Hopfield Model for associative memory. They conducted a systematic simulation investigation of various storage algorithms for Hopfield Networks and compared these storage schemes with the Hebbian Learning based algorithms.

2. *PERFORMANCE EVALUATION OF HOPFIELD ASSOCIATIVE MEMORY FOR COMPRESSED IMAGES by Manu Pratap Singh, Dr. S. S. Pandey, Vikas Pandey*

The paper is designed to analyze the performance of a Hopfield neural network for storage and recall of compressed images. In this paper they considered the images of different sizes. These images are first compressed by using wavelet transformation. The compressed images are then preprocessed and the feature vectors of these images are obtained. The training set consists with all the pattern information of the preprocessed and compressed images. Here each input pattern is of size 900 X 1. Each pattern of training set is encoded into Hopfield neural network using Hebbian and pseudo inverse learning rules.

METHODS:

HEBBIAN LEARNING RULE FOR HOPFIELD NETWORKS:

The **Hebbian Theory** was introduced by Donald Hebb in 1949, in order to explain "associative learning", in which simultaneous activation of neuron cells leads to pronounced increases in synaptic strength between those cells.

It is often summarized as "Neurons that fire together, wire together. Neurons that fire out of sync, fail to link".

The Hebbian rule is both local and incremental. For the Hopfield Networks, it is implemented in the following manner, when learning n binary patterns, if the corresponding to i and j are equal in pattern μ , then product of the epsilon terms corresponding to each μ , will be positive. This would in turn, have a positive effect on the weight W_{ij} and the values of i and j will tend to become equal. The opposite happens if the bits corresponding to neurons i and j are different.

The Hebbian Learning rule is implemented shown below:

$$w_{ij} = \frac{1}{n} \sum_{\mu=1}^n \epsilon_i^{\mu} \epsilon_j^{\mu}$$

where ϵ_i^{μ} represents bit i from pattern μ .

where ϵ is the pattern, μ is pattern number and i, j are elements of pattern vector and p is total no of patterns.

THE TRAINING METHOD:

The training process is essentially the same as the Hebbian learning, except here the two associated patterns in each pair are the same (self-association), i.e., $x_k = y_k$, and the input and output patterns all have the same dimension $m = n$. After the network is trained by Hebbian learning its weight matrix is obtained as the sum of the outer-products of the K patterns to be stored:

$$W_{n \times n} = \frac{1}{n} \sum_{k=1}^K y_k x_k^T = \frac{1}{n} \sum_{k=1}^K y_k y_k^T = \frac{1}{n} \sum_{k=1}^K \begin{bmatrix} y_1^{(k)} \\ \vdots \\ y_n^{(k)} \end{bmatrix} [y_1^{(k)}, \dots, y_n^{(k)}]$$

The weights connecting node i and node j is such that $W_{ij} = W_{ji}$

Also, there exists no self - connection i.e $W_{ii} = 0$. The weights vectors (matrix) look like the following :

$$W = \begin{pmatrix} 0 & \omega_{12} & \cdots & \omega_{1i} & \cdots & \omega_{1n} \\ \omega_{21} & 0 & \cdots & \omega_{2i} & \cdots & \omega_{2n} \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ \omega_{i1} & \omega_{i2} & \cdots & 0 & \cdots & \omega_{in} \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ \omega_{n1} & \omega_{n2} & \cdots & \omega_{ni} & \cdots & 0 \end{pmatrix}$$

Here is a snippet in Python which we used in our project.

```

1 def hebbian_train_hopfield(epsilon2,num_patterns):
2     W=np.zeros((sz,sz))
3     h=np.zeros((sz))
4     for i in tqdm(range(sz)):
5         for j in range(sz):
6             for p_idx in range(num_patterns):
7                 W[i,j]=W[i,j]+(epsilon2[p_idx,i]*epsilon2[p_idx,j]).sum()
8
9             if(i==j):
10                W[i,j]=0
11     W=W/sz
12     return W

```

Here 'sz' variable is the number of pixels in the input image.

THE TESTING METHOD:

Now for testing the trained Neurons we need to use the weights that we got after training on the set of patterns. In many memory retrieval experiments, a cue with partial information is given at the beginning of a recall trial. The retrieval of a memory item is verified by the completion of the missing information.

Now for every time-step, we will update our pattern vector. We randomly choose an element i and update it according to this equation $I_i = \text{Sum}(W_{ij}S_j)$.

Now we apply sign function to it (h in code) and we take threshold θ to be 0, h is the input potential, which is broken as follows.

$$h_i(t) = \sum_j w_{ij} S_j(t) = c \sum_{j=1}^N \sum_{\mu=1}^M p_i^\mu p_j^\mu S_j(t) = c N \sum_{\mu=1}^M p_i^\mu m^\mu(t)$$

Here is a snippet in Python which we implemented for regenerating/reconstructing a full image given a partial image/input state. Since we already have stored the W_{ij} 's we don't need to use the patterns again and can use the weights directly.

```

1 def generate_test_images(test_img,W,num_iterations=10):
2     fig = plt.figure(figsize = (8, 8))
3     #hamming_distance = np.zeros((30, num_patterns))
4     h=np.zeros((sz))
5     for iteration in tqdm(range(num_iterations)):
6         for i in range(sz):
7             i = np.random.randint(sz)
8             h[i] = 0
9             for j in range(sz):
10                h[i] += W[i, j]*test_img[j]
11     new_test_img = np.where(h<0, -1, 1)

```

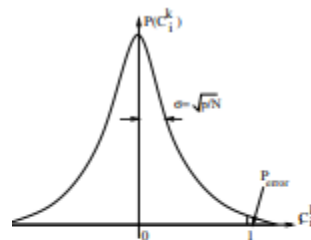
Things to notice about Hebbian type of learning:

- It has been observed that the Hebbian Learning algorithm with Hopfield Networks can store at max $0.138 * N$ patterns which means if there are N nodes in the Hopfield Network then it can store at max 13.8% of the total number of Nodes. i.e A network of 1000 nodes can store at max 138 patterns with some error probability
- If we store p patterns and N is the number of nodes in the Hopfield network, then the probability of error increases with the ratio of p/N .

The p/N ratio and the error probability

From various sources and research it was found that that P_{error} depends on the p/N ratio as follows.

The table shows the error for some values of p/N .



P_{error}	p/N
0.001	0.105
0.0036	0.138
0.01	0.185
0.05	0.37
0.1	0.61

RESULTS:

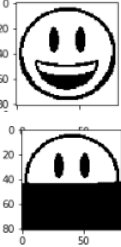









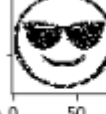

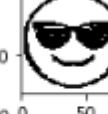



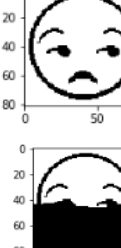

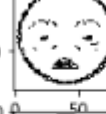
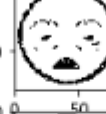
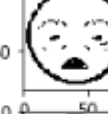
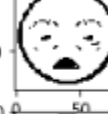


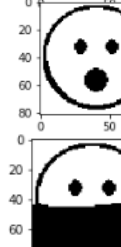



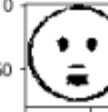



In our project we tried storing different types of patterns we tried to store are

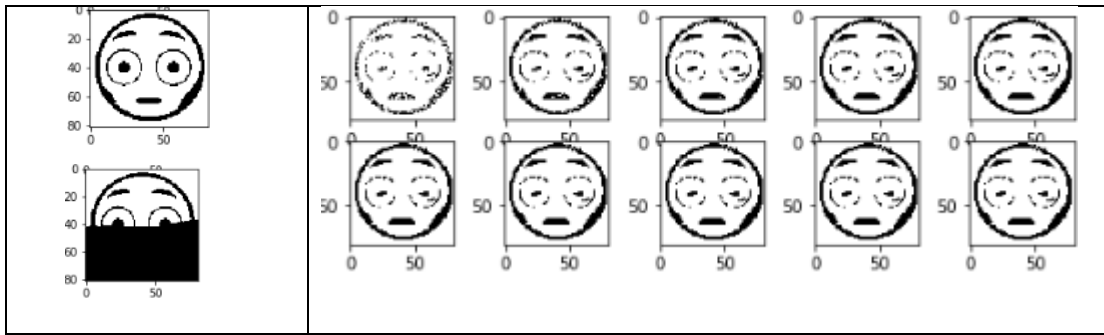
1. 5 Emojis
2. 3 Cartoon Images(Bugs Bunny)
3. 2 Human Faces
4. 9 Digits

The results we got for the Emojis and Human Faces were excellent and satisfactory for the Cartoon Images. The learning algorithm didn't seem to work well for extracting multiple digits by learning multiple digit patterns.

The results are displayed as follows:

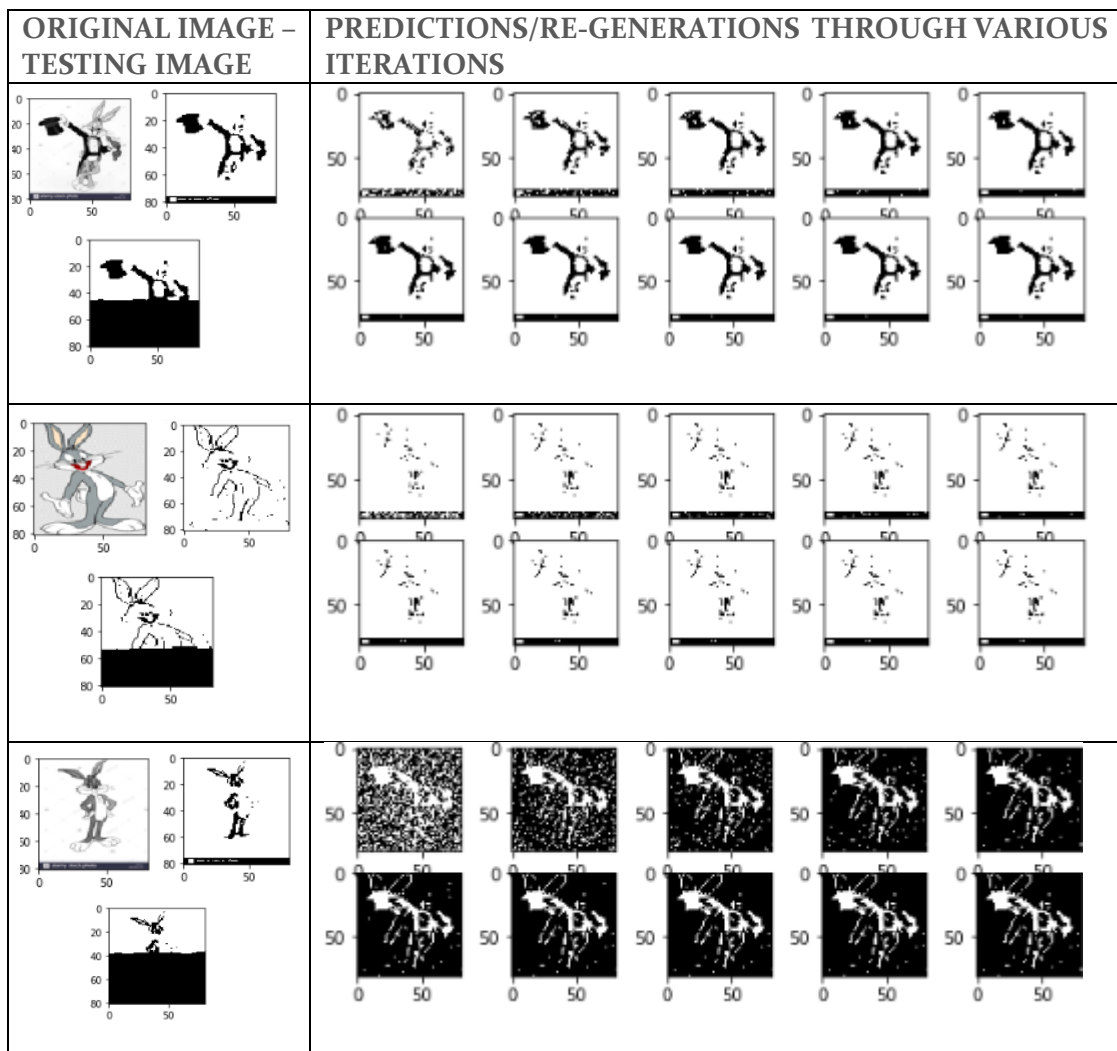
1. EMOJIS

ORIGINAL IMAGE – TESTING IMAGE	PREDICTIONS/RE-GENERATIONS THROUGH VARIOUS ITERATIONS									
										
										
										
										



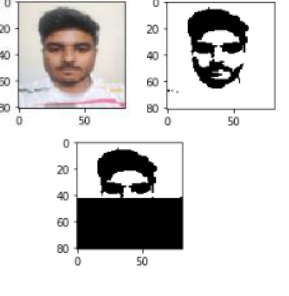
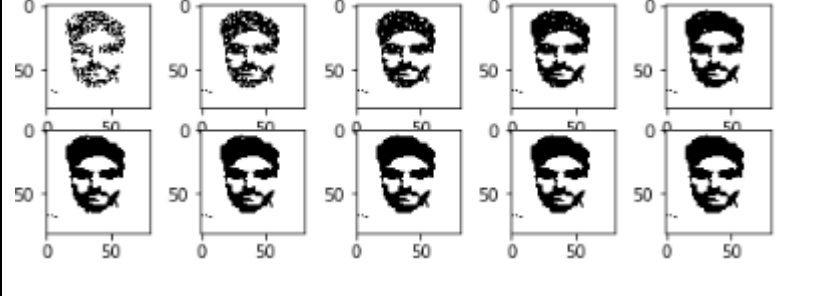
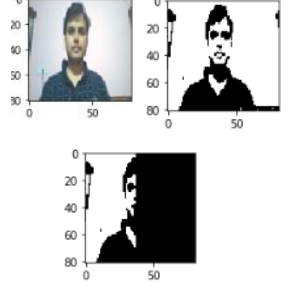
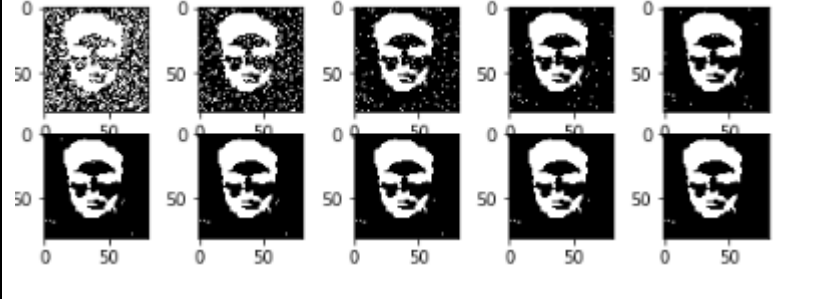
We see that most of the emojis are predicted correctly. Such patterns usually perform well in this type of Learning. Such patterns are also called Strong Patterns.

2. CARTOON IMAGES (BUGS BUNNY)



From the above results we see that the first image is perfectly recreated but the order two distort a little. The third image is black as Hopfield networks tend to converge/settle in a local minima , and there may be more than one local minima.

3. HUMAN FACES

ORIGINAL IMAGE – TESTING IMAGE	PREDICTIONS/RE-GENERATIONS THROUGH VARIOUS ITERATIONS									
										
										

We see that the second image wasn't recalled very well and settled in a wrong local minima.

This may be because the second image had a lot of details as compared to first one and the number of nodes weren't enough to store the correct set of weights.

- We also tried one more experiment with the digits from 1-9 but they didn't give good results so we are not mentioning it in this report. However it is shown in the code notebook shared along this report.

CONCLUSION:

So in this report we demonstrated the mathematics behind Hebbian Learning and how they were implemented in Python and also showed some experiments which we carried out with the network model. We also showed in which cases it gave excellent performance and in which of them it failed to give the desired results.

REFERENCES:

- [1] [https://www.researchgate.net/publication/254052169 Storage Capacity of the Hopfield Neural Network Associative Memory/link/00b49530dc5fc421c6000000/download](https://www.researchgate.net/publication/254052169_Storage_Capacity_of_the_Hopfield_Neural_Network_Associative_Memory/link/00b49530dc5fc421c6000000/download)
- [2] <https://www.elkjournals.com/MasterAdmin/UploadFolder/2-PERFORMANCE-EVALUATION-OF-HOPFIELD-ASSOCIATIVE-2-3/2-PERFORMANCE-EVALUATION-OF-HOPFIELD-ASSOCIATIVE-2-3.pdf>
- [3] https://en.wikipedia.org/wiki/Hopfield_network#Hebbian_learning_rule_for_Hopfield_networks
- [4] <http://fourier.eng.hmc.edu/e161/lectures/nn/node5.html>
- [5] <https://neurondynamics.epfl.ch/online/Ch17.S2.html>
- [6] [http://www.philadelphia.edu.jo/academics/qhamarsheh/uploads/Lecture%2012%20-Supervised%20Learning%20 Hopfield%20Networks%20-\(Part%205\).pdf](http://www.philadelphia.edu.jo/academics/qhamarsheh/uploads/Lecture%2012%20-Supervised%20Learning%20 Hopfield%20Networks%20-(Part%205).pdf)
- [7] Jankowski S., Lozowski A., Zurada J. M., “Complex Valued Multistate Neural Associative Memory, IEEE Transactions on Neural Networks, 7(4), 1491 – 1496 (1996)
- [8] McEliece, R. J., Posner, E. C., Rodemich, E. R. and Venkatesh, S. S., “The capacity of the Hopfield associative memory”, IEEE Trans Information Theory IT-33 4, pp. 461-482, (1987).
- [9] Kumar, S. and Singh, M. P., “Pattern Recall Analysis of the Hopfield Neural Network with a Genetic Algorithm”, Computers and Mathematics with Applications, vol. 60(4), pp. 1049-1057, (2010)