AMATH383: Hopfield Neural Network for Machine Learning

Hyesu Lee, Aiden Liu, Shari Sung

December 2019

1 Abstract

Developing deep neural networks is a very powerful tool and some of the neural networks today are able to understand complex functions that human brains are unable to comprehend. It is, thus, crucial for us to understand the algorithm behind different types of neural networks to find out the pros and cons of each type of neural network and to utilize them in an efficient manner. Unlike many neural networks used for deep learning, Hopfield Neural Network does not use Multilayer Perceptron (MLP), which aroused the question of how Hopfield Neural Network still suitable for the purpose of deep learning. It uses recurrence execution and Hebbian learning, which allows Content Addressable Memory (CAM). Since we are interested in pursuing a higher degree in machine learning, computer science and quantitative finance related fields, the fact that Hopfield Networks has contributed to so many fields is the reason why it is of interest to us. Therefore, we wanted to dig deeper into Hopfield Networks and see what we could achieve by understanding and evaluating it.

2 Background Information

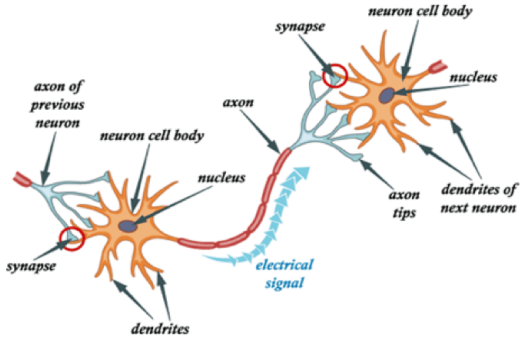
2.1 What are neural networks:

Neural networks are made up with algorithms and are intended to be modeled similarly to the human brain neurons. A Neural Network is made up of the composition of the nonlinear functions of two or more neurons. There are two types of Neural Networks: Feedforward network and Recurrent network. In Feedforward neural networks, the information flows in one direction, while in Recurrent neural networks, it is possible to revisit neurons from previous executions. One of the many uses of neural networks is to recognize patterns by inputting numbers stored in vectors. These neural networks are made to train machines and to predict the output based on corrupted data.

2.2 Terms Used in Neural Networks

2.2.1 Neurons:

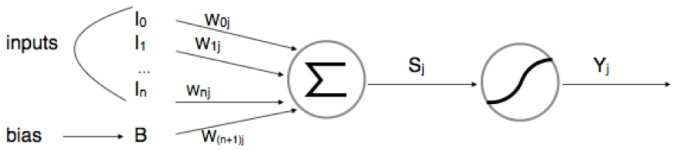
Neurons are the basic units of a neural network, which is made to imitate biological neurons. Biological neurons contain a number of dendrites (inputs), a cell nucleus (processor) and an axon (output). When a neuron activates, it collects all of its incoming inputs, and if the incoming inputs go over a certain threshold, it will fire a signal through the axon. In a artificial neural network, neurons are nonlinear, parameterized, and bounded functions.



(11)

2.2.2 Inputs:

Input is a vector that stores the original values (x0, x1, x2, ..., xn). We then multiply the input vector with its individual weights (w0, w1, w2, ...,wn) for each activation.



(10)

2.3 Types of Training for Artificial Neural Networks

The learning process for an ANN (Artificial Neural Network) is the process through which the weights of the network are determined. This is achieved by adjusting the weights until certain criteria are satisfied. There are three main types of learning:

2.3.1 Supervised learning

The artificial neural network (ANN) takes in a training data set that contains the input vectors with a target (which is the desired output) associated. The weights of the ANN are adjusted each iteration, so that the error between the actual output of the ANN and the target is minimized.

2.3.2 Unsupervised learning

The ANN takes in a data set which contains only input vectors. The weights of the ANN are adjusted such that the output provides a clustering of the input vectors based on certain criteria. This type of learning allows the discovery of patterns in the data. Hopfield Neural Networks falls into this category.

2.3.3 Reinforcement learning

The ANN takes in a data set that contains only input vectors. The weights of the ANN are adjusted so that the parts of the ANN performing functionally (based on certain criteria) are rewarded while the other parts are penalized.

2.4 Neuron Network Applications in Different Fields

2.4.1 Finance

Neural networks apply mostly to actuarial predicting, pricing and hedging, future price forecasting, and stock performances. Economists mostly combine statistical techniques with neural networks to get useful data that will drive decision making.

2.4.2 Medicine

Neural networks are mostly implemented to model parts of the human body. It recognizes diseases by using various scans which are implemented with different neural networks. Diseases such as cancer or tumors could be identified through neural network applications in machine learning.

2.4.3 Border Safety and Fraud Detection

With neural network's ability to take in a lot of inputs, process and infer hidden non-linear relationships, neural networks are implemented for character and image recognition. Banks uses character recognition for handwriting to detect bank frauds and even national security assessments. The application for image recognition ranges from facial recognition in social media to satellite imagery processing for international defense usage.

3 Problem Description

As John Hopfield describes, Hopfield networks contain computational electrochemical properties such that neurons are interconnected. Unlike many Neural Networks used during mid-1900s, Hopfield Networks do not use the idea of Multilayer Perceptron (MLP). The biggest difference is that MLP only allows forward direction neural connection, while Hopfield doesn't. This recurrence execution allows Hopfield networks to "remember" pattern with unsupervised learning. This arouse the question of the paper: How does Hopfield Neural Networks suit for the purpose of machine learning? In order to show that, the neural network needs to guarantee that any given pattern will be converged to one of the trained patterns. In this paper, we will show the mechanism of the Hopfield Neural Network during each update, why it is guaranteed to converge, and how it converges to the trained pattern. Also, since Hopfield Networks are widely used for machine learning in collective property, we will also focus on how we could benefit from Hopfield Network's ability to produce Content Addressable Memory (CAM).

4 Hopfield Neural Network System

The idea of Hopfield Neural Network is that it will have a system of neurons and for each update, every neuron attributes to the state of other neurons. After a number of iteration, the system of neurons should converge to one of the known patterns that was learned before.
Similar to other neural networks, Hopfield contains neurons which contain two states: fired or not fired. Let the fired state of the neuron be represented by +1 and not fired state of the neuron be represented by -1. Let each neuron be v_i when N is the number of total neurons and $i \in [1, N]$. The system of neurons could be represented as row vector V.

$$V = \begin{pmatrix} v_1 & v_2 & v_3 & ... & v_N \end{pmatrix}$$

The instantaneous state of V would be described with a list of N values of V_i such that it represents binary of N bits of data. Then, there is a weight matrix that indicates the amount each neuron contributes to the state of other neurons in the next iteration, this contribution is called "connection". For N neurons, there are N^2 connections. However, there is no contribution from a neuron to itself, which allows $N*(N-1)$ non-zero connections. Let T be the N by N matrix which contains connections for every neuron such that value t_{ij} represents the

connection from v_i to v_j, when $i, j \in [1, N]$.

$$T = \begin{pmatrix} 0 & t_{1,2} & t_{1,3} & ... & t_{1,N} \\ t_{2,1} & 0 & t_{2,3} & ... & t_{2,N} \\ t_{3,1} & & & & . \\ . & & & & . \\ . & & & & . \\ . & & & & . \\ t_{N,1} & t_{N,2} & t_{N,3} & ... & 0 \end{pmatrix}$$

Each connection is declared by the following formula. Let m be the number of different neuron systems that the network would remember.

$$t_{ij} = \sum_{s=1}^{m} (2v_i^s - 1)(2v_j^s - 1)$$

Then for each iteration, the update would follow the Hebbian learning algorithm. Let $\theta(z)$ be the function as follows,

$$\theta(z) = \begin{cases} +1, \text{if } z > 0 \\ -1, \text{if } z \leq 0 \end{cases}$$

Then,

$$v_i^{new} = \theta(\sum_{j \neq i} t_{ji} * v_j)$$

Therefore, the following is true when one neuron changes its state.

$$v_i(\sum_{j} t_{ji} * v_j) < 0$$

Since the new neuron state contributes to the updates of other neuron states, the flipping of the signs will continue until it converges to one neuron system and that does not make further changes. This converged state should be one of the "learned" state, and is said to be a "stationary" state. For the next sections, we will explain how the system is guaranteed to converge instead of looping infinitely and how the system converges to one of the "learned" patterns.

5 Simplification

For the simplicity of the model, the biases of neural networks will not be considered in the model and we will disregard any noise from $v_i \neq v_j$. Bias is considered as a phenomena of observing results that are systematically prejudiced due to faulty assumptions. Also, the connection between two different neurons are considered to be same regardless of their order.

$$t_{ij} = t_{ji}$$

This is explanatory from the definition of connection between two neurons

$$t_{ij} = (2v_i - 1)(2v_j - 1) = (2v_j - 1)(2v_i - 1) = t_{ji}$$

Therefore, the matrix of the connections would be a symmetric matrix with 0 at its diagonal indices. Let N to be the number of data bits. Then the T matrix would look like the following:

$$T = \begin{pmatrix} 0 & T_{12} & T_{13} & T_{14} & ... & T_{1N} \\ T_{21} & 0 & T_{23} & T_{24} & ... & T_{2N} \\ T_{31} & T_{32} & 0 & T_{24} & ... & T_{3N} \\ . & & & & & . \\ . & & & & & . \\ . & & & & & . \\ T_{(N-1),1} & & & ... & 0 & T_{(N-1)N} \\ T_{N1} & T_{N2} & T_{N3} & T_{N4} & ... & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & T_{12} & T_{13} & T_{14} & ... & T_{1N} \\ T_{12} & 0 & T_{23} & T_{24} & ... & T_{2N} \\ T_{13} & T_{23} & 0 & T_{24} & ... & T_{3N} \\ . & & & & & . \\ . & & & & & . \\ . & & & & & . \\ T_{1,(N-1)} & & & ... & 0 & T_{(N-1)N} \\ T_{1N} & T_{2N} & T_{3N} & T_{4N} & ... & 0 \end{pmatrix}$$

This allows $\frac{N(N-1)}{2}$ unique connections between two neurons.

6 Energy State Model

First, we will prove how a neuron system is guaranteed to come to a stop. Let v_i^{new} be the updated neuron and v_i^{old} be the neuron before the update. Assume the update results in a flip in the sign of the state such that $v_i^{old} \neq sign(v_i(\sum_{j \neq i} t_{ji} * v_j))$ Then, $v_i^{new} = -v_i^{old}$, therefore we can derive the difference of two states.

$$\Delta v_i = v_i^{new}(\sum_{j \neq i} t_{ji} * v_j) - v_i^{old}(\sum_{j \neq i} t_{ji} * v_j)$$

$$\Delta v_i = (v_i^{new} - v_i^{old})(\sum_{j \neq i} t_{ji} * v_j)$$

$$v_i^{new} - v_i^{old} = v_i^{new} + v_i^{new} = 2 * v_i^{new} < 0$$

Since the sign of the state flipped,

$$(\sum_{j \neq i} t_{ji} * v_j) < 0$$

Therefore,

$$\Delta v_i = (v^{new}_i - v^{old}_i)(\sum_{j \neq i} t_{ji} * v_j) > 0$$

Thus, for every flip, the state of the neuron is increasing. Similarly, let S be the sum of the state of all neurons in the system.

$$S = \sum_i (v_i * \sum_{j<i} t_{ji} * v_j)$$

$$= \sum_{i,j<i} (v_i * t_{ji} * v_j)$$

Let S^{old} be the S before one neuron (v_k) get flipped and S^{new} be the S after one neuron flipped. Since every neuron except v_k remains the same in S^{old} and S^{new}, the difference between S^{old} and S^{new} is shown below.

$$\Delta S = S^{old} - S^{new} = (v^{new}_k - v^{old}_k)(\sum_{j \neq i} t_{ji} * v_j)$$

This expression is the same as the previous computation. Thus,

$$\Delta S > 0$$

We can conclude that the state of the system of all neurons also increased. If S does not contain an upper bound, then it could infinitely increase the state of the neural system.

$$Max\{S\} = Max\{\sum_{i,j<i} (v_i * t_{ji} * v_j)\} = \sum_{i,j<i} (|t_{ji}|)$$

Since $\sum_{i,j<i}(t_{ji})$ is a finite numbers of connection weights, S has an upper bound. We also need to make sure that ΔS is not infinitely small such that there are infinitely many executions towards the upper bound.

$$Min\{\Delta S\} = \min_{i\{v_i, i=1...N\}} 2 * |\sum_{j<i} t_{ji} * v_j|$$

Then, using Archimedian Property, there exists a natural number B such that

$$Min\{\Delta S\} \geq \frac{1}{B} > \epsilon > 0$$

The S has an upper bound and the minimum of ΔS is greater than ϵ, meaning that it is not infinitely small. The series of S over time is monotonically increasing and converges to the upper bound. Then, what exactly is the significance of the upper bound? John Hopfield introduces the energy equation as the following. With some manipulation, the given energy equation turns into -S.

$$E = -\frac{1}{2} \sum_i \sum_j (t_{ji} * v_i * v_j)$$

$$= -\sum_{i}\sum_{j<i}(t_{ji} * v_i * v_j)$$

$$= -\sum_{i,j<i}(t_{ji} * v_i * v_j)$$

$$= -S$$

Therefore, we can conclude that for every flip, the energy of neural systems decrease and converge to the lowest energy level. As learned from the AMATH383 lectures, any given data near the lowest energy point would slide down to the global minimum, because it is the stable point of the equation. Then, how does the storing process correlate with the energy map?

To learn patterns, the Hopfield Neural Network utilizes the idea of Hebbian Learning. Hebbian learning is one of the oldest learning algorithms, and is based in large part on the dynamics of biological systems. A synapse between two neurons is strengthened when the neurons on either side of the synapse (input and output) have highly correlated outputs. In essence, when an input neuron fires, if it frequently leads to the firing of the output neuron, the synapse is strengthened. Following the analogy to an artificial system, the connection matrix (the weight) would be the product of two neurons.

Let P be a single pattern that needs to be learned. Then, we are essentially designing a connection matrix with elements $\{t_{ij}\}$ such that the energy is a local minimum at the desired $P = V$, when V is the system of neurons.

As mentioned above, the neuron will only flip if the weighted sum of the other neurons' output is of the opposite sign. Therefore, the pattern should be at a stationary state where the neurons will no longer flip, which means:

$$sign\{\sum_{j\neq i} t_{ji} * v_j\} = v_i \forall i$$

To achieve this, the idea of Hebbian Learning can be implemented, where we let

$$t_{ij} = v_i v_j$$

where t_{ij} is the weight that connects neuron v_i and v_j. It could be written as the product of the two neurons. Substituting $v_i v_j$ for t_{ij} in sign equation.

$$sign\{\sum_{j\neq i} t_{ji} * v_j\}$$

$$= sign(\sum_{j\neq i} v_j * v_i * v_j\}$$

$$= sign\{\sum_{j\neq i} v_j^2 * v_i\}$$

Since $v_j^2 > 0$, the sign would depend on v_i, which could be pulled out from summation.

$$= sign\{v_i\} = v_i$$

Thus, the Hebbian learning guarantees the pattern to be stationary.
Now consider the energy of the Hopfield Network. When the Hebbian Learning equation is applied,

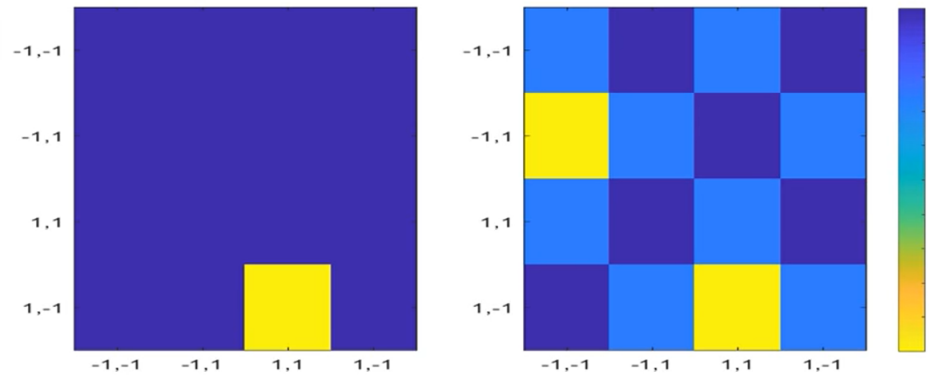
$$E = -\sum_i \sum_{j<i} w_{ji} * y_i * y_i$$

$$= -\sum_i \sum_{j<i} y_i^2 * y_j^2$$

$$= -\sum_i \sum_{j<i} 1$$

$$= -0.5 * N(N-1)$$

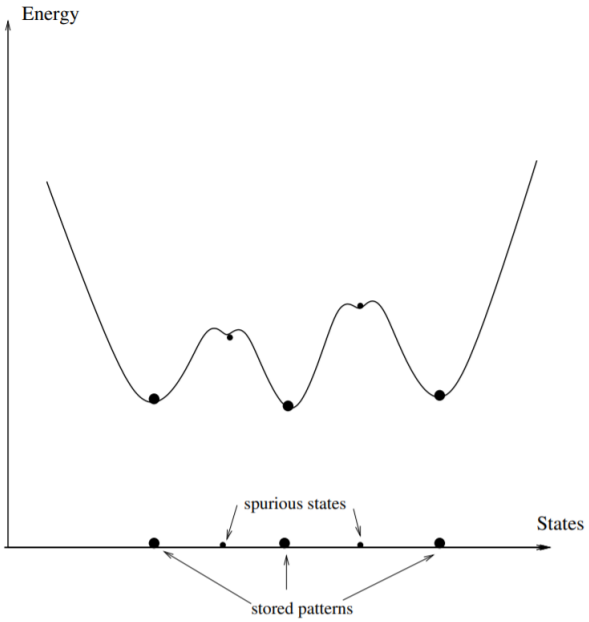
This is the lower bound (global minimum) of the network's energy. Hence, the pattern is stable.
As shown, Hopfield Neural Networks use Hebbian learning which creates weights that would have the lowest energy level in the given patterns. Therefore, given any pattern that is close to the stored pattern, that is, where the energy of the network is close to a local/global minimum, it will inevitably evolve to the previously trained pattern.



The left imagery(8) shows the the bit pattern that is stored, and the right imagery is the energy map which indicates the stored pattern has the lowest energy. Since the connection between two neurons is the same regardless of order, the energy map would have two global minima for one stored pattern. Depending on the order of execution, the state of the system of neurons would stop at either one of the global minima, which stores the same value.

7 Discussion

Unfortunately, Hopfield Neural Networks could cause errors due to parasitic patterns and saving too many patterns. The reason why we get parasitic patterns, which are fake patterns that gets remembered in the weight matrix, is because $v_{parasitic} = sign(v_a + v_b + v_c)$. In other words, the faulty pattern is created due to the combination of many different patterns. When we sum up the three patterns, v_a, v_b and v_c, it ends up being a local minimum in the energy contour(7).



This local minimum "pulls" any given data, instead of letting it fall to the global minimum. This parasite pattern leads to an error by remembering the wrong pattern which was not intended to be learned.
There is a threshold for the number of patterns that a Hopfield Neural Network could contain for accurate output. Having an infinite number of patterns would cause vague border between the minima and potential error.
Nonetheless, it is still possible to use the neural network and prevent those potential errors.

8 Improvement

For parasitic patterns, or often called spurious local minima, it is possible to avoid it by adding more noise to the patterns or adding more neurons.

Theoretically, it could only hold up to the 0.14∗Number of neuron patterns. Following is a chart with potential error percentage and the ratio between numbers of pattern and numbers of neurons(7)

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

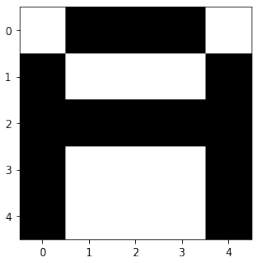
This is based on the assumption the patterns are orthogonal and that the prob(any bits being 1)=0.5, which means that the number of matched bits in any pair equals the number of matched bits in any pair. However, this is actually considered as the worst-case scenario. If we use non-orthogonal patterns instead of using strictly orthogonal ones like John Hopfield did in his assumption , the neuron network could actually store more patterns, since the distance is closer between each neural network. This makes it easier for the network to remember patterns.
Another way of adding more patterns is to add more neurons to what we originally have. For example, if we add another K number of neurons to the N number of neurons. The capacity of neurons would then be $N * (N + K)$, resulting in better capacity and less spurious memories.

9 Demonstration

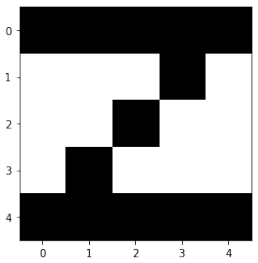
```
# This is a demonstration of how a simple Hopfield Network
# can be implemented in training and
# recognizing optical characters
# Objective: Store two letters and try to recognize
# the noisy version of those letters

from numpy import array

# Implementing functions to set up the input to the
# Hopfield Network

# Since Hopfield Networks are only represented in binary vectors,
# Create a function to convert the image string representation
# to binary vectors, with 'X' being +1, and '.' being -1.
def get_pattern(letter):
```

```
    return array([+1 if c=='X' else -1
                  for c in letter.replace('\n','')])

# Create a function that displays the letter
# by reshaping the binary vectors into a matrix
def get_display(pattern):
    from pylab import imshow, cm, show
    imshow(pattern.reshape((5,5)),cmap=cm.binary,
    interpolation='nearest')
    show()

# Implementing functions of the Hopfield Network

# Implementing the training formula
def train(patterns):
    from numpy import zeros, outer, diag_indices
    r,c = patterns.shape
    W = zeros((c,c))
    for p in patterns:
        W = W + outer(p,p) # Outer product
    # Setting the diagonal elements of the matrix to zero
    W[diag_indices(c)] = 0
    return W/r # Normalize the weight matrix and return it


# Implementing a function to recall a stored pattern/letter
# In order to keep this simple,
# the operation stops after 5 steps, which is usually
# sufficent to recognize simple patterns like 'A' or 'Z'
def recall(W, patterns, steps=5):
    from numpy import vectorize, dot
    sgn = vectorize(lambda x: -1 if x<0 else +1)
    for _ in range(steps):
        patterns = sgn(dot(patterns,W))
    return patterns

# Create images of 'A' and 'Z' from a string representation
A = """
.XXX.
X...X
XXXXX
X...X
X...X
"""
Z = """
XXXXX
```

```
...X.
..X..
.X...
XXXXX
"""

# Displaying the letters
get_display(get_pattern(A))
```

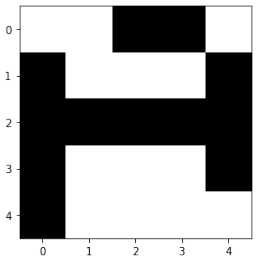



```
get_display(get_pattern(Z))
```

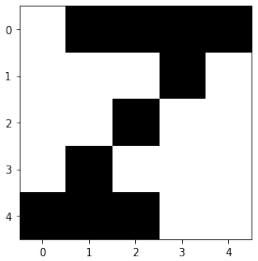



```
# Store the two letter arrays into a single array
letter_patterns = array([get_pattern(A), get_pattern(Z)])

# Train/Store the two letter patterns in the Hopfield Network
stored_patterns = train(letter_patterns)

# Introduce distorted 'A' and distorted 'Z'
# as string representations
distorted_a = """
..XX.
X...X
XXXXX
X...X
X....
```

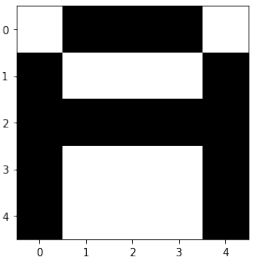
```
"""
distorted_z = """
.XXXX
...X.
..X..
.X...
XXX..
"""

# Displaying the distorted letters
get_display(get_pattern(distorted_a))
```



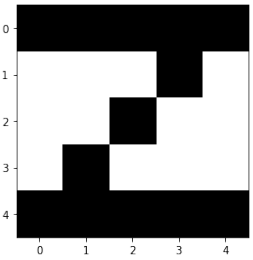

```
get_display(get_pattern(distorted_z))
```




```
# Store the two distorted letter arrays into a single array
distorted_patterns = array([get_pattern(distorted_a),
get_pattern(distorted_z)])

# Correct the distorted patterns by using the recall() method
# in the Hopfield Network
corrected_patterns = recall(stored_patterns, distorted_patterns,
steps=5)

# Display the corrected patterns and
# compare it with the original letter images created
get_display(corrected_patterns[0])
```




```
get_display(corrected_patterns[1])
```



10 Conclusion

As shown throughout the paper, Hopfield Neural Networks promise to converge to any stored patterns within a finite number of iterations by using the concept of Hebbian Learning. It is thus suitable to utilize Hopfield Networks in machine learning due to the existence of the training/storing and learning functions of the network, and how easy it is to be used in simple image processing and handwriting recognition. However, Hopfield Networks does have its limitations, including its limited memory to store patterns, and the fact that it could potentially result in errors in its predictions if there are stored patterns that are closely associated.

11 Reference

(1) Sandhu, Jaspreet. "Understanding and Reducing Bias in Machine Learning". April 5, 2019.
(2) Nielsen, and Michael A. “Neural Networks and Deep Learning.” Neural Networks and Deep Learning, Determination Press, 1 Jan. 1970,
(3) Fadelli, Ingrid. “A Ferroelectric Ternary Content-Addressable Memory to Enhance Deep Learning Models.” Tech Xplore - Technology and Engineering News, Tech Xplore, 5 Dec. 2019,
(4) Krebs, P R, and W K Theumann."Generalization in a Hopfield network with noise",Journal of Physics A: Mathematical and General,
(5) Hopfield, J J. "Neural networks and physical systems with emergent collective computational abilities.",Proc Natl Acad Sci USA. 1982 Apr,
(6) Unknown. “Fun with Hopfield and Numpy". code-affectionate, 1 Jan. 1970
(7) Unknown. "Hopfield-networks-15. ppt". Imperial College London,
(8) Raj, Bhiksha. "Lecture 20: Hopfield Networks 1". Carnegie Mellon University Deep Learning Department, April 4, 2018
(9) Unknown, Hopfield Networks". UCLA Computer
(10)Dreyfus Gerard. Neural Networks Methodology and Applications. Springer, 2005.
(11) Unknown, "CONDUCTION OF NERVE IMPULSE", simplebiology.