LAB REPORT 5

November 22, 2020

Submitted by:

Kazi Amit Hasan
Roll: 1503089
Department of Computer Science & Engineering
Rajshahi University of Engineering & Technology

November 22, 2020

0.1 Title

Implementation of **Hopfield Neural Network** algorithm.

0.2 Objectives

1. Learning the concept of Hopfield neural network algorithm
2. Learning the maths behind the algorithm.
3. Learning and implementing in a dataset.

0.3 Methodology

Hopfield neural network is a very typical feedback neural network. In addition to the feedforward connection between the same neurons as the feedforward nervous system, there is obviously a feedback connection. There are several characteristics of hopfield algorithm. The neurons are fully connected and are a single layer neural network.
Each neuron is both an input and an output, resulting in a relative weight matrix, which saves computation. Under the input excitation, its output will produce constant state changes, and this feedback process will be repeated. If the Hopfield neural network is a convergent stable network, the variation of this feedback and iterative calculation process becomes smaller and smaller. Once the stable equilibrium state is reached, the Hopfield network will output a stable constant value.
The Hopfield network can store a set of balance points such that when a given set of initial states of the network, the network traverses to the equilibrium of the design by itself. Of course, according to thermodynamics, the equilibrium state is divided into a stable state and a metastable state, which are very likely to be in the convergence process of the network.
For a recursive network, the state at time t is related to the output state at time t-1. The subsequent neuron update process also uses Asynchronous.

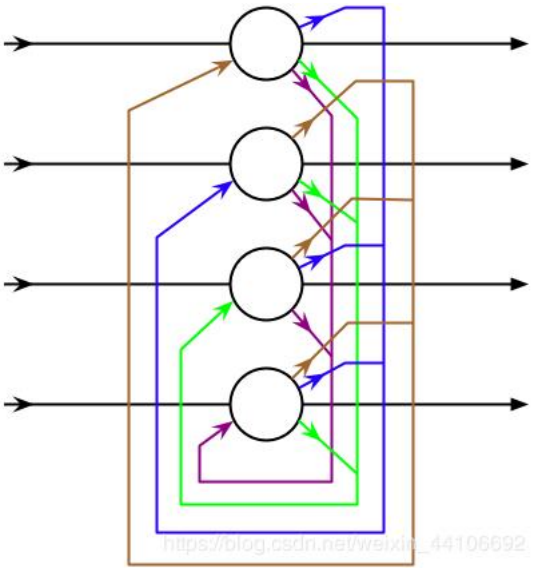

Figure 1: Hopfield algorithm

0.4 Implementation

The tools that I used for the implementation purpose of this lab are Jupyter notebook. The codes are written in Python language. The algorithm steps were discussed in Methodology section.

0.5 Code

0.5.1 Implementation the Hopfield Neural Network algorithm in Python

```
import numpy as np

nb_patterns = 4
pattern_width = 5
pattern_height = 9

X = np.zeros((nb_patterns, pattern_width * pattern_height))

#Number 1
```

```
X[0] = [
    -1, -1,  1,  1, -1,
    -1,  1,  1,  1, -1,
    -1, -1,  1,  1, -1,
    -1, -1,  1,  1, -1,
    -1, -1,  1,  1, -1,
    -1, -1,  1,  1, -1,
    -1, -1,  1,  1, -1,
    -1, -1,  1,  1, -1,
    -1, -1,  1,  1, -1
]

#Number 2
X[1] = [
     1,  1,  1,  1,  1,
     1,  1,  1,  1,  1,
    -1, -1, -1,  1,  1,
    -1, -1, -1,  1,  1,
     1,  1,  1,  1,  1,
     1,  1, -1, -1, -1,
     1,  1, -1, -1, -1,
     1,  1,  1,  1,  1,
     1,  1,  1,  1,  1
]

#Number 3
X[2] = [
     1,  1,  1,  1,  1,
     1,  1,  1,  1,  1,
    -1, -1, -1,  1,  1,
    -1, -1, -1,  1,  1,
     1,  1,  1,  1,  1,
    -1, -1, -1,  1,  1,
    -1, -1, -1,  1,  1,
     1,  1,  1,  1,  1,
```

```
     1,  1,  1,  1,  1
]

#Number 4
X[3] = [
     1,  1, -1,  1,  1,
     1,  1, -1,  1,  1,
     1,  1, -1,  1,  1,
     1,  1,  1,  1,  1,
     1,  1,  1,  1,  1,
    -1, -1, -1,  1,  1,
    -1, -1, -1,  1,  1,
    -1, -1, -1,  1,  1,
    -1, -1, -1,  1,  1
]

import matplotlib.pyplot as plt
def plot(X):
    fig, ax = plt.subplots(1, nb_patterns, figsize=(10, 5))

    fig.suptitle('Embedded Patterns for Training', fontsize=16)

    for i in range(nb_patterns):
        ax[i].matshow(X[i].reshape((pattern_height, pattern_width)), cmap='binary')
        ax[i].set_xticks([])
        ax[i].set_yticks([])

    fig.tight_layout()
    plt.show()
plot(X)

def training(X):
    return np.dot(X.T,X)/len(X[0]) -(np.identity(pattern_width*pattern_height)*
    (len(X)/len(X[0])))
```

```
W=training(X)

W

def genNoise(num, porc):

    vet = np.copy(num)
    size = len(vet)
    qtdPixels = int((size * porc) / 100)
    pixelsToModify = random.sample(range(size), qtdPixels)
    for i in range(qtdPixels):
        p = pixelsToModify[i]
        vet[p] = -1 if vet[p] == 1 else 1

    return vet

def activation(X,w):
    u = X.copy()
    for i in range(len(u)):
         if (np.dot(X,w[i])>0):
                u[i]=1
         else:
                u[i]= -1
    return u

def predict(X,w):
    vCurr=X.copy()
    epoch=0
    vPrev=0
    while(np.array_equal(vCurr,vPrev)==False):
        print(epoch+1)
        epoch+=1
        vPrev=vCurr.copy()
        vCurr=activation(X,w)
```

```
    return vCurr

def showPatterns(titlePattern, patternOriginal, patternCorrupted,
patternRecovered):

    fig, ax = plt.subplots(1, 3, figsize=(10, 5))

    fig.suptitle(titlePattern, fontsize=16)

    ax[0].matshow(patternOriginal.reshape(pattern_height, pattern_width),
    cmap='binary')
    ax[0].set_title('Transmitted Image \n (no noise)')
    ax[0].set_xticks([])
    ax[0].set_yticks([])

    ax[1].matshow(patternCorrupted.reshape(pattern_height, pattern_width),
    cmap='binary')
    ax[1].set_title('Image Received \n (with noises)')
    ax[1].set_xticks([])
    ax[1].set_yticks([])

    ax[2].matshow(patternRecovered.reshape(pattern_height, pattern_width),
    cmap='binary')
    ax[2].set_title('Recovered Image \n (no noise)')
    ax[2].set_xticks([])
    ax[2].set_yticks([])

    fig.tight_layout()
    fig.subplots_adjust(top=0.75)

    plt.show()

def test(titlePattern, pattern, noiseProc):
    original = pattern.copy()
    noise = gerarRuido(original, noiseProc)
```

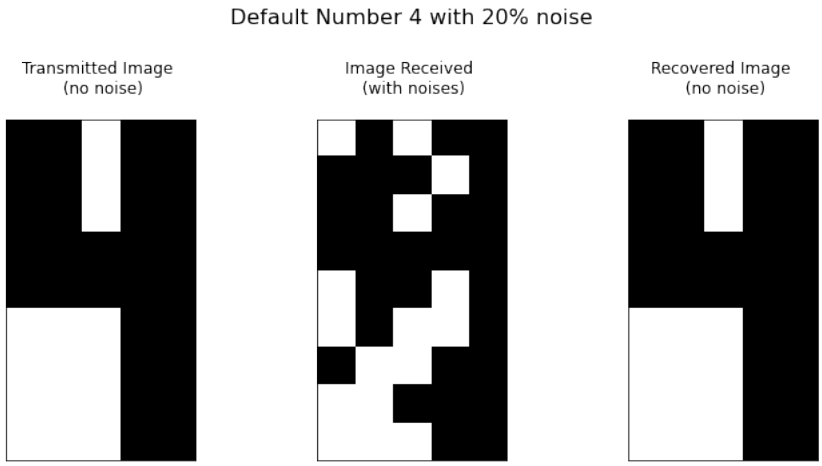

Figure 2: Results 1

```
    recovered = predict(noise,W)
    showPatterns(titlePattern, original, noise, recovered)

import random
for i in range(4):
    for j in range(3):
        test('Default Number {} with 20% noise'.format(i+1), X[i], 20)

for i in range(4):
    p=20
    for j in range(10):
        test('Default Number {} with {}% noise'.format(i+1,p+5), X[i], p)
        p+=5
```

0.6 Results and Performance Analysis

In the code section, I gave four pattern. I used 1, 2, 3, 4 numbers as the patterns. I created a noise function to add some noise in the data and the algorithm will restore the image from the noise data.

We can see in Fig.2 that our code received an image with noise, but our algorithm successfully predicted and recovered the value. In this case, we used only 20 % noise. The change in noise percentage can affect the results.

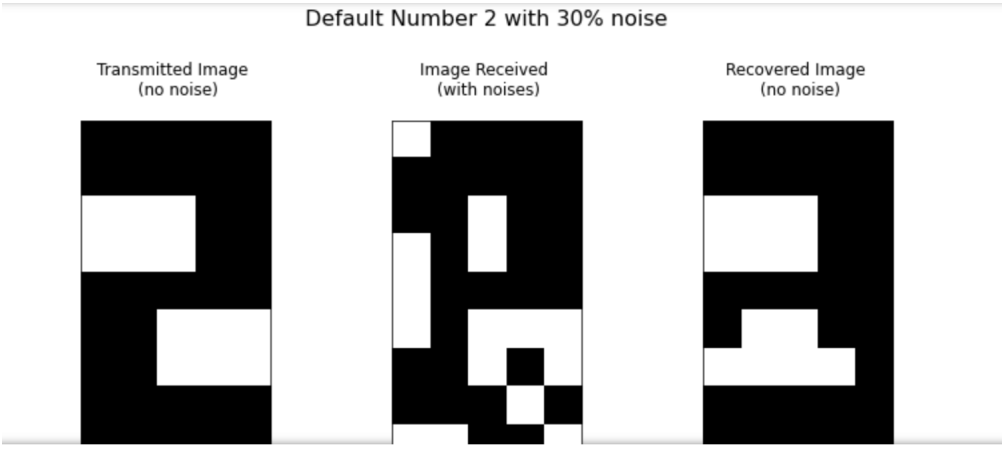

Figure 3: Result 2

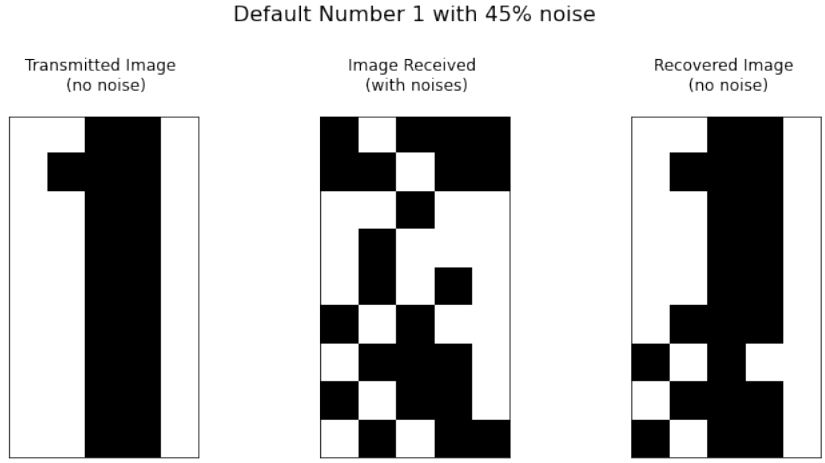

Figure 4: Results 3

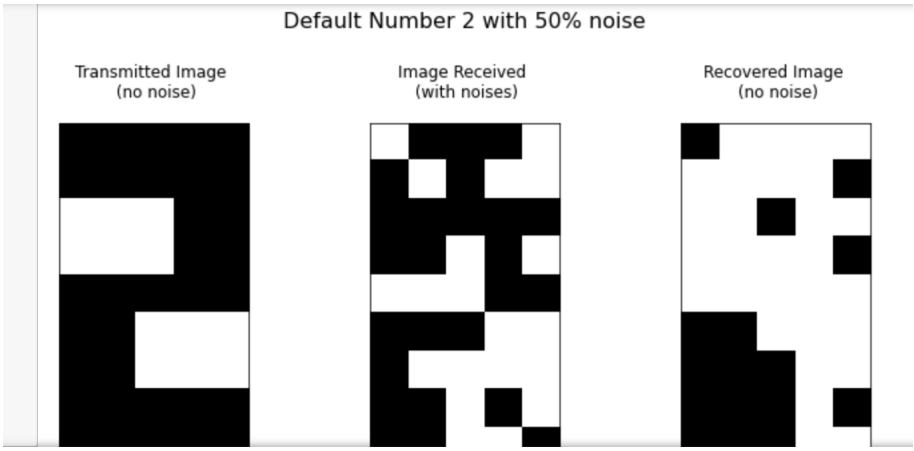

Figure 5: Result 4

To be more specific. I compared the results with 30%, 45%, 50% noises. Our algorithm performed comparatively better with 30% noise.

0.7 Conclusion

By implementing this, we knew about the fundamentals of one of the most basic machine learning algorithms. It has some basic advantages. In this lab, I tried to implement the hopfield neural network algorithm without using any automated library functions All the codes and neccesary files are available in my github profile. The codes will be publicly available after my finals grades. My Github profile: https://github.com/AmitHasanShuvo/