Title: Implementation of Hopfield Network Algorithm.

Objective:

1. Train by only defining the weights
2. Predict after calculating in real time, no previous calculation needed
3. Speedy with a high accuracy rate

Methodology:

One of the major contributions to the area of neural networks was made in the early 1980s by John Hopfield, who not only studies an auto associated network that has some similarities with the perceptrons, but also some important differences. Hopfield's contribution was not simply the suggestion of a suitable model, but his extensive analysis and study, which has led to his name being associated with the network. He developed the use of an energy function, and related the networks to other physical systems. The Hopfield net consists of a number of nodes, each connected to every other node, as shown in the figures.

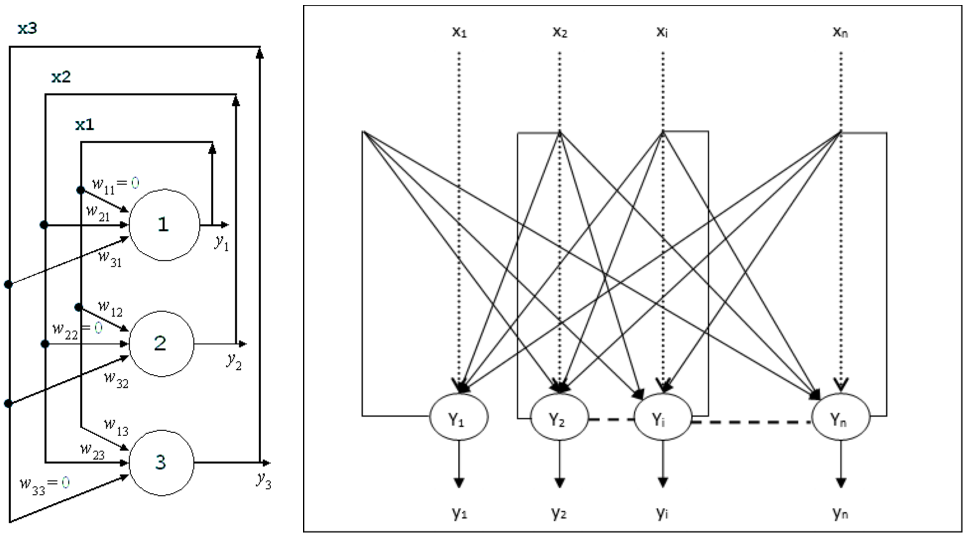


Figure: Two different views of Hopfield Network Model

Here, these two figures are showing the two different looks of the Hopfield Network although both of them are the same at their core. The algorithm is given as follows:

1. **Assign connection weights:**

$$w_{ij} = \begin{cases} \displaystyle\sum_{s=0}^{M-1} x_i^s x_j^s \text{ , } i \neq j \\ 0, i = j, 0 \leq i, j \leq M - 1 \end{cases}$$

   where w_{ij} is the connection weight between node i and node j, and x_i^s is element i of the exemplar pattern for class s, and is either $+1$ or -1. There are M patterns, from 0 to $M-1$, in total. The thresholds of the units are zero.

2. **Initialize with unknown pattern:**

$$\mu_i(0) = x_i, 0 \le i \le N-1$$

where $\mu_i(t)$ is the output node i at time t.

3. **Iterate until convergence:**

$$\mu_i(t+1) = f_h\left[\sum_{j=0}^{N-1} w_{ij}\mu_j(t)\right], 0 \le j \le N-1$$

The function f_h is the hard-limiting non-linearity, the step function. Repeat the iteration until the outputs from the nodes remain unchanged.

An example can be helpful for better understanding the Hopfield network. Two images have been shown below. The first one shows the samples of train set and the second picture shows how the system is predicting the output with extreme precision.

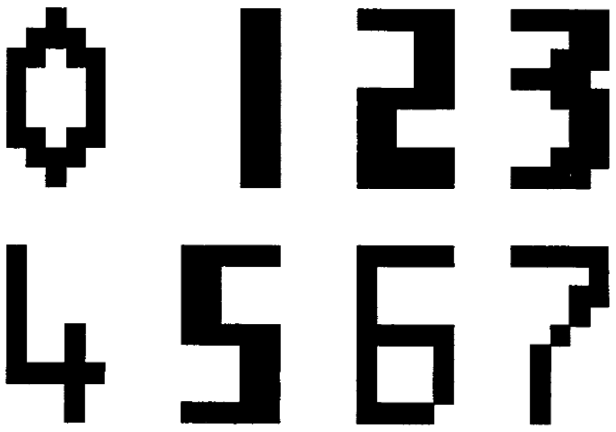


Figure: Samples in the train set

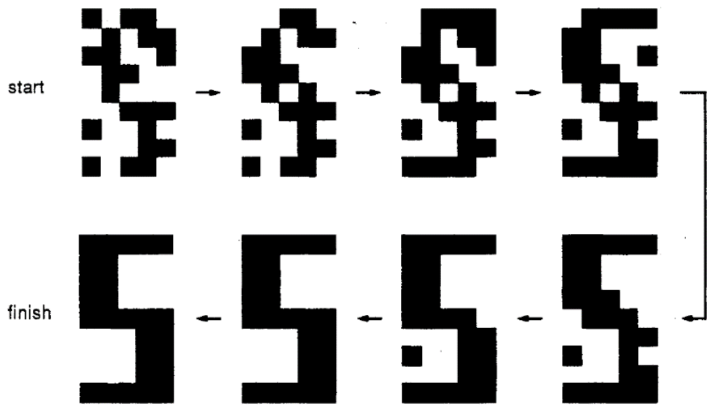
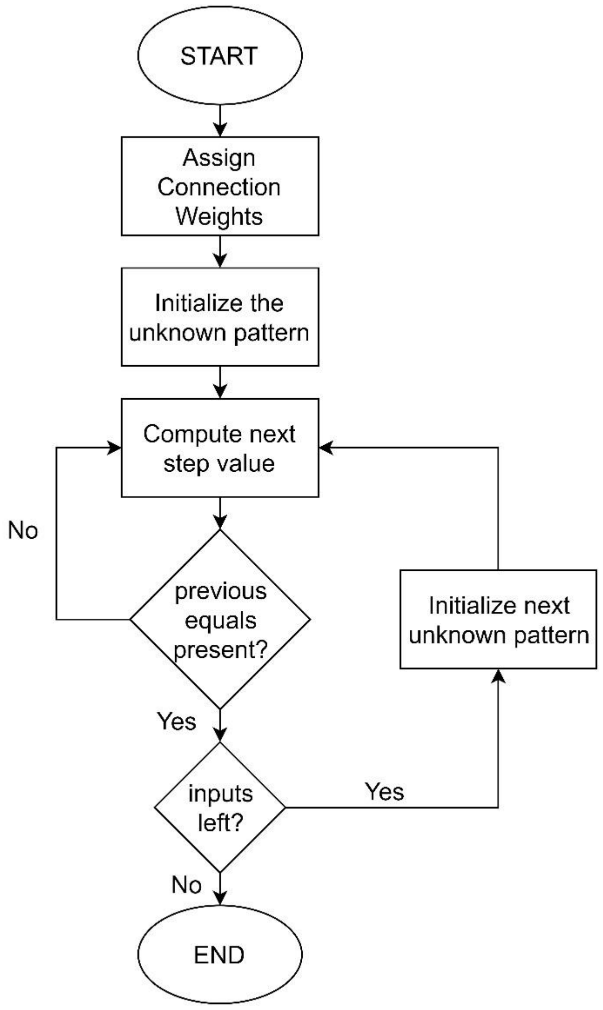


Figure: How the system is matching a sample?

Hence, it can be easily understood, Hopfield's Network Algorithm works great for this kind of datasets. Now, let's look at the flowchart of the algorithm. The flowchart is given as follows:



START
Assign Connection Weights
Initialize the unknown pattern
Compute next step value
No
previous equals present?
Initialize next unknown pattern
Yes
inputs left?
Yes
No
END

Figure: Flowchart for Hopfield Network Algorithm

Implementation:

To implement Hopfield Network Algorithm,

1. Firstly, dataset is generated
2. Dataset is split into train set and test set
3. After training, testing is done to find out best match for each input

The dataset contains 18 samples. As the train percentage is 50%, hence, 9 samples are in train set and 9 samples are in test set. The number of nodes specified for Hopfield Network Algorithm is 60 here. Now, the question is why 60? Why 9 samples in train? The answer remains in the very core of the algorithm.

Researchers have claimed that if there are n number of nodes, then there must be not more than $0.15n$ number of samples in the train set otherwise the values will never converge, the more samples we take, the more error happens. So, if we keep 15 samples in train set, we need 100 nodes. As, I've taken 60 nodes, 9 samples are there in the train class.

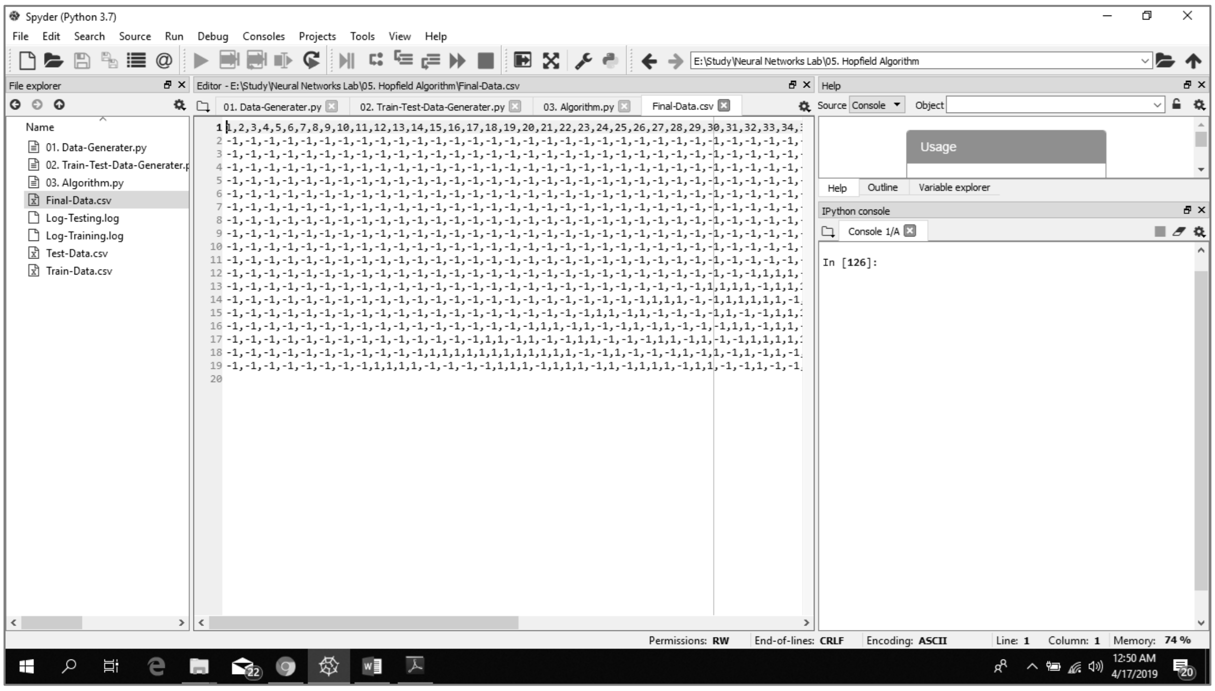
Then the algorithm is applied and the results are shown through files. All these files are stores in GitHub link provided in the abstract section of this report.

Code-1: Dataset Generation

```
"""
Code: Hopfield Network Algorithm
Title: Dataset Generater
Author: Azmain Yakin Srizon
"""

from numpy import binary_repr # Adding binary conversion library
import random

filename = 'Final-Data.csv'
log = open(filename, 'w')  # creating csv file

data = []

input_line = 60 # define input lines

# creating dataset
numbers = []
for i in range(0,52,int(60/18)):
    numbers.append(int(pow(2,i))+random.randint( int(pow(2,i-1)),int(pow(2,i))))


for i in numbers: # generating data as binary form
    tmp = list(binary_repr(i,input_line))
    data.append(tmp) # adding to data

for i in range(0,input_line-1): # generating column names of csv file
    log.write(str(i+1)+',')
log.write(str(i+2)+'\n')

c = 0
for i in data: # adding data to csv file
    for j in i:
        c = c+1
        if int(j)==0:
            j=-1
        log.write(str(j)) # adding value in each column for a row
        if c!=input_line:
            log.write(',')
        else:
            c=0
    log.write('\n') # go to next row

log.close() # closing the csv file
```

Output-1: Dataset Generation

Here is a snapshot of the generated dataset:


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
E:\Study\Neural Networks Lab\05. Hopfield Algorithm
File explorer
Name
01. Data-Generater.py
02. Train-Test-Data-Generater.p
03. Algorithm.py
Final-Data.csv
Log-Testing.log
Log-Training.log
Test-Data.csv
Train-Data.csv
Editor - E:\Study\Neural Networks Lab\05. Hopfield Algorithm\Final-Data.csv
01. Data-Generater.py
02. Train-Test-Data-Generater.py
03. Algorithm.py
Final-Data.csv
Help
Source Console
Object
Usage
Help Outline Variable explorer
IPython console
Console 1/A
Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 1 Column: 1 Memory: 74 %
12:50 AM
4/17/2019

```
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,
-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,-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,-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,-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,-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,-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,-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,-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,-1,1,1,1,1,-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,-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,-1,1,1,-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,-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,-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
```

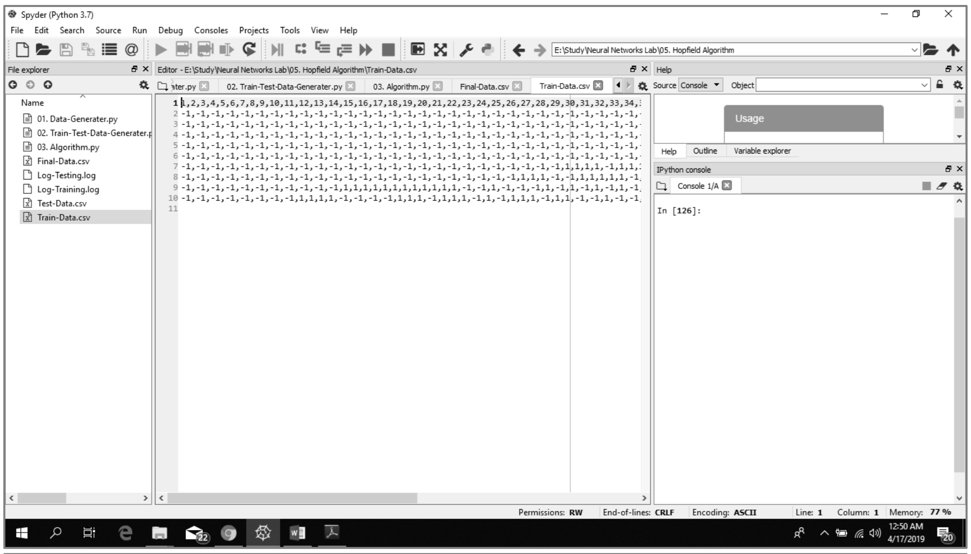
```
In [126]:
```

Code-2: Train-Test Generation

```
"""
Code: Hopfield Network Algorithm
Title: Train-Test Dataset Generater
Author: Azmain Yakin Srizon
"""

# importing necessary libraries
import pandas as pd
import numpy as np

df = pd.read_csv('Final-Data.csv') # reading in dataframe
msk = np.random.rand(len(df)) < 0.49 # 50-50 in train test
train = df[msk] # define train
test = df[~msk] # define test

train.to_csv('Train-Data.csv', index = None, header=True) # write train
test.to_csv('Test-Data.csv', index = None, header=True) # write test
```

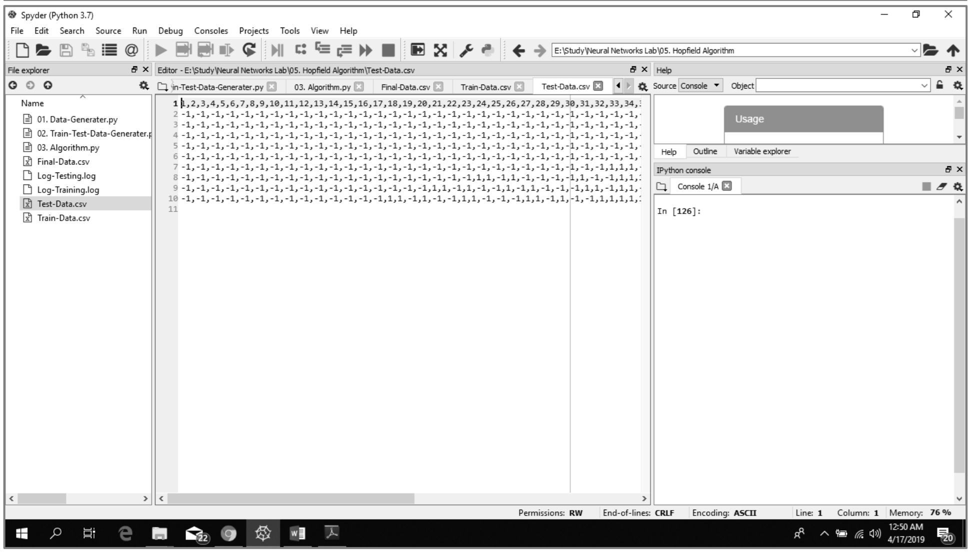
Output-2: Train-Test Generation

Here are 2 snapshots of train and test sets:


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
E:\Study\Neural Networks Lab\05. Hopfield Algorithm
File explorer
Name
01. Data-Generater.py
02. Train-Test-Data-Generater.p
03. Algorithm.py
Final-Data.csv
Log-Testing.log
Log-Training.log
Test-Data.csv
Train-Data.csv
Editor - E:\Study\Neural Networks Lab\05. Hopfield Algorithm\Train-Data.csv
ater.py
02. Train-Test-Data-Generater.py
03. Algorithm.py
Final-Data.csv
Train-Data.csv
Help
Source Console Object
Usage
Help Outline Variable explorer
IPython console
Console 1/A
Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 1 Column: 1 Memory: 77 %
12:50 AM 4/17/2019

```
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,
-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,-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,-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,-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,-1,1,1,1,1,-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,-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,-1,1,1,1,-1,1,1,1,-1,1,-1,1,1,1,-1,1,1,-1,-1,1,-1,-1,
```

```
In [126]:
```


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
E:\Study\Neural Networks Lab\05. Hopfield Algorithm
File explorer
Name
01. Data-Generater.py
02. Train-Test-Data-Generater.p
03. Algorithm.py
Final-Data.csv
Log-Testing.log
Log-Training.log
Test-Data.csv
Train-Data.csv
Editor - E:\Study\Neural Networks Lab\05. Hopfield Algorithm\Test-Data.csv
in-Test-Data-Generater.py
03. Algorithm.py
Final-Data.csv
Train-Data.csv
Test-Data.csv
Help
Source Console Object
Usage
Help Outline Variable explorer
IPython console
Console 1/A
Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 1 Column: 1 Memory: 76 %
12:50 AM 4/17/2019

```
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,
-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,-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,-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,-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,-1,1,1,1,
-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,-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,-1,-1,-1,1,1,-1,1,-1,-1,1,1,1,1,
```

```
In [126]:
```

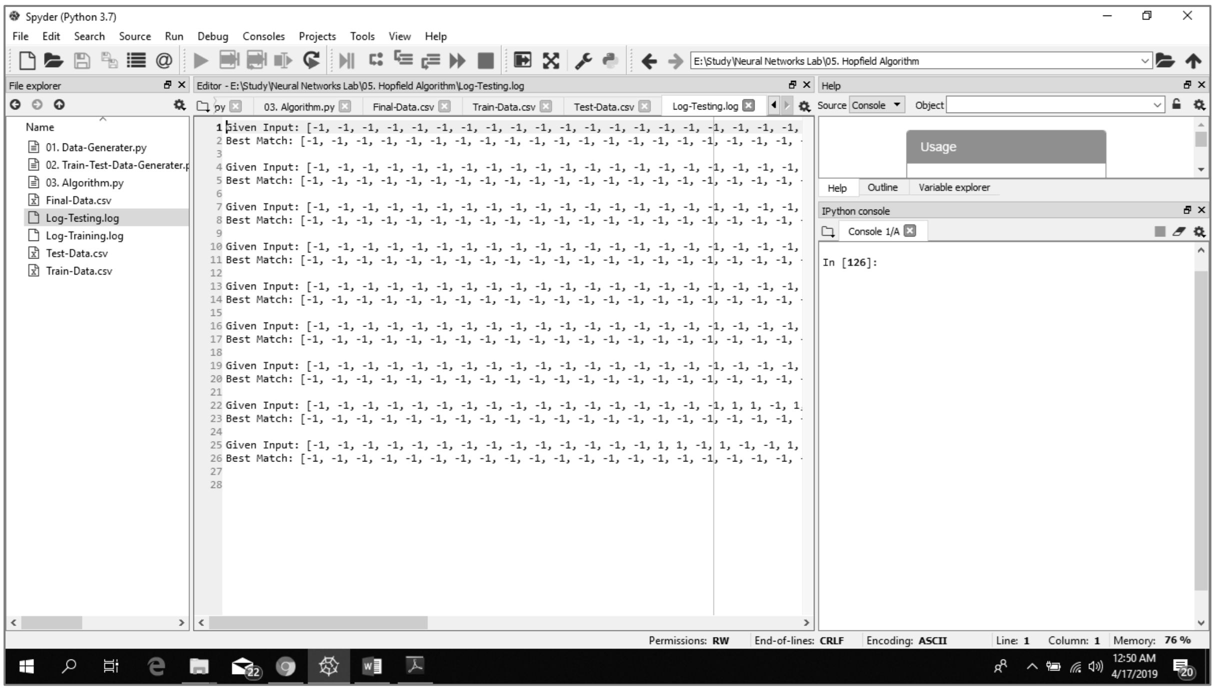
Code-3: Main Algorithm

```
"""
Code: Hopfield Network Algorithm
Title: Algorithm Implementation
Author: Azmain Yakin Srizon
"""

# adding necessary libraries
import pandas

df = pandas.read_csv('Train-Data.csv') # reading the training csv

filename = 'Log-Training.log'
log = open(filename, 'w') #creating training log file

input_nodes = int(df.size/len(df)) # calculating number of inputs
f_h = 0.5 # threshold

weights = [] # initializing weights by 0
for i in range(input_nodes):
    tmp = []
    for j in range(input_nodes):
        tmp.append(0.0)
    weights.append(tmp)

inputs = [] # reading inputs
for j in range(0, len(df)):
    tmp=[]
    for i in range(0,input_nodes):
        tmp.append(df[str(i+1)][j])
    inputs.append(tmp)

for i in range(0,input_nodes): # calculating weights, training phase
    for j in range(0,input_nodes):
        for k in inputs:
            weights[i][j]=weights[i][j]+k[i]*k[j]

log.write(str(weights)) # printing weights in training log
log.close() # closing train log

df = pandas.read_csv('Test-Data.csv') # reading test csv

filename = 'Log-Testing.log'
log = open(filename, 'w') # creating test log

inputs = [] # reading inputs
for j in range(0, len(df)):
    tmp=[]
    for i in range(0,input_nodes):
        tmp.append(df[str(i+1)][j])
    inputs.append(tmp)

for k in inputs: # for each input
    main = k
    log.write('Given Input: ' + str(main) + '\n')
    while(1):
        prev = k
        for i in range(0,len(k)):
            tmp = 0
            for j in range(0,input_nodes):
                tmp = tmp + weights[i][j]*k[j] # calculate summation
            if tmp<f_h: # hard thresholding - no
                k[i]=-1
            else: # hard thresholding - yes
                k[i]=1
        if prev == k: # matches? then break
```

```
            log.write('Best Match: ' + str(prev) + '\n\n') # writing in test log
            break

log.close() # closing test log
```

Output-3: Main Algorithm

Here is a snapshot of the output:



```
Given Input: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
Best Match: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,

Given Input: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
Best Match: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,

Given Input: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
Best Match: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,

Given Input: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
Best Match: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,

Given Input: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
Best Match: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,

Given Input: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
Best Match: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,

Given Input: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
Best Match: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,

Given Input: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1, -1, 1
Best Match: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,

Given Input: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1, -1, 1, -1, -1, 1,
Best Match: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
```

```
In [126]:
```

Performance Analysis:

Not much parameters are present in Hopfield Network Algorithm and for this reason users have not a lot of things to handle themselves.

As described earlier if we're taking 15 samples in the train set, then there has to be 100 nodes. If not, error happens and the values never converges which indicates the fall of the algorithm. However, if we increase the number of samples in the train set, the number of nodes needed also increase. And the weight matrix also increases in size. That means Hopfield Network Algorithm needs more memory if we increase the number of samples in the train set. The other neural network algorithms don't require these much memory at all.

Another parameter is the number of samples in test. Here we've taken 9 samples in test set. However, the number can be increased. The overall accuracy of the algorithm will never decrease. The algorithm will find out the most matching training sample.

The value of threshold is a factor for speed. The value has been kept as 0.5 for average speed. However, if the threshold is very high or very low one of the two things can happen: may get faster, may get slower.

Notice that, the calculation of prediction is done in the test set. In the training phase the weights are being calculated only. In the testing phase, we're actually calculating for prediction. Hence, it does not work like other neural network algorithms where first the kernel is made and then prediction is done. Here, there is no kernel, only weights.

Limitations of this algorithm is that:

1. Need huge memory in terms of more samples. For example: if we need to take English letters in train, then train samples are = 26+26 = 52. Hence, we need a huge number of nodes.
2. This algorithm may not work for all kind of datasets.
3. If the values of samples are not scattered, then the matched patter may be same every time.

Apart from this limitation, Hopfield Network Algorithm was one of the most amazing inventions of all time.