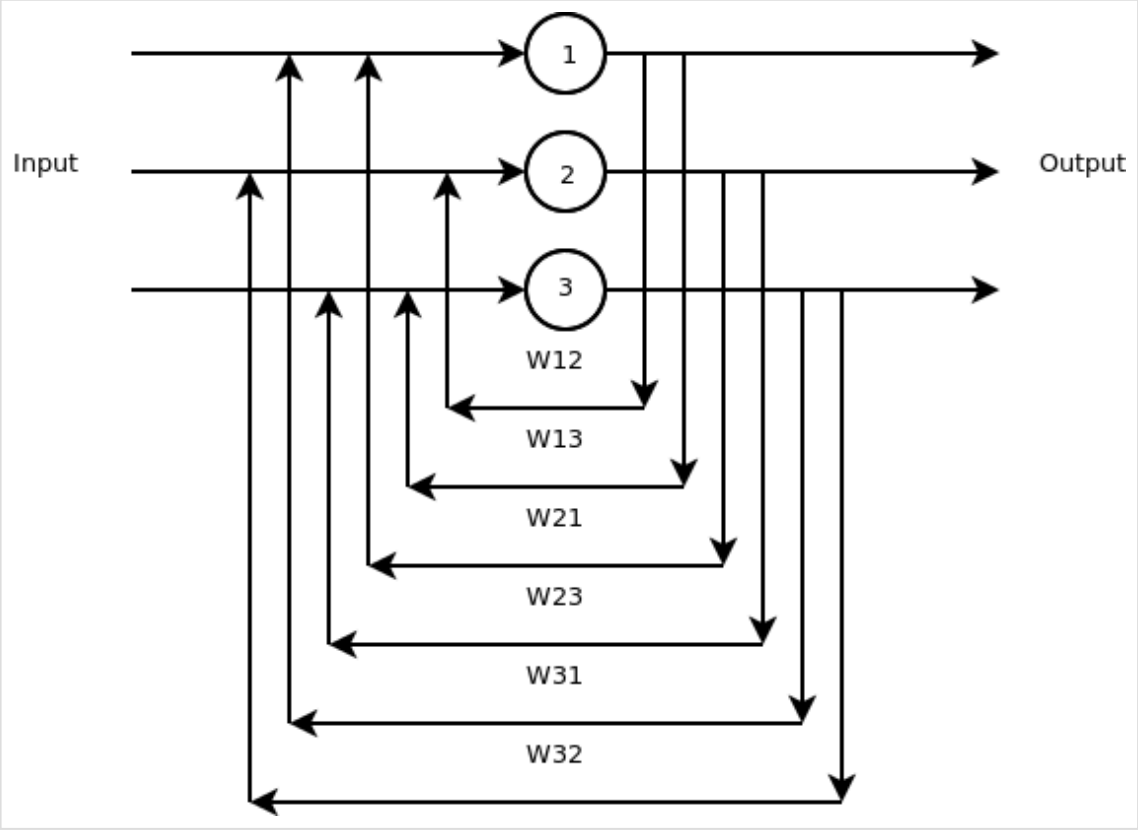
Panagiotis Matigakis
Programming in Python

Character recognition using Hopfield networks

Posted on **January 18, 2014**

The Hopfield network is a single layer artificial neural network that can be used to recall patterns that have been stored in it. The Hopfield network can serve as a content-addressable associative memory because when it is given a noisy input pattern it will converge to one of the patterns it has been trained with that best matches the input pattern. This neural network consists of a number of neurons that is equal to the number of inputs. Every neuron is connected to every other neuron and a value is assigned to that connection which is called the weight of the connection.



The inputs to the neural network can either be -1 or 1 and the outputs will also be either one of those two values. the rule that is used to calculate the output of a neuron is the following.

$$s_i = \begin{cases} 1 & \text{if } \sum_{j=1}^{k} w_{ij} s_j > \theta_i \\ -1 & \text{otherwise} \end{cases}$$

Where

- w_{ij} is the weight from neuron i to neuron j
- s_j is the state of neuron j
- θ_i is a threshhold value
- k is the number of bits in the input pattern

The rules that apply for the weights are the following.

- $w_{ij} = 0$ if i = j
- $w_{ij} = w_{ji}$

One easy way to train a Hopfield network is to use the Hebbian learning rule. The equation that is used to calculate the values of the weight matrix is the following.

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

Where ϵ_i^μ is bit i from the input patternμ.

One important thing that should be taken into account is the number of patterns a network can store. This number is approximately 0.14 times the number of neurons in the network. For example a network with 100 neurons can store 14 patterns.

Hopfield network example

In this example I will use aHopfield network libraryI wrote in Python in order to train a neural network that can recognize the letters A, B and C.

```
1  from random import randint
2
3  import numpy as np
4  from matplotlib import pyplot as plt
5
6  from hopfieldnet.net import  HopfieldNetwork
7  from hopfieldnet.trainers import hebbian_training
8
9  #Create the training patterns
10 a_pattern = np.array([[0, 0, 1, 0, 0],
11                        [0, 1, 0, 1, 0],
12                        [1, 0, 0, 0, 1],
13                        [1, 1, 1, 1, 1],
14                        [1, 0, 0, 0, 1],
15                        [1, 0, 0, 0, 1],
```

```
                        [1, 0, 0, 0, 1]])

b_pattern = np.array([[1, 1, 1, 1, 0],
                      [1, 0, 0, 0, 1],
                      [1, 0, 0, 0, 1],
                      [1, 1, 1, 1, 0],
                      [1, 0, 0, 0, 1],
                      [1, 0, 0, 0, 1],
                      [1, 1, 1, 1, 0]])

c_pattern = np.array([[1, 1, 1, 1, 1],
                      [1, 0, 0, 0, 0],
                      [1, 0, 0, 0, 0],
                      [1, 0, 0, 0, 0],
                      [1, 0, 0, 0, 0],
                      [1, 0, 0, 0, 0],
                      [1, 1, 1, 1, 1]])

a_pattern *= 2
a_pattern -= 1

b_pattern *= 2
b_pattern -= 1

c_pattern *= 2
c_pattern -= 1

input_patterns = np.array([a_pattern.flatten(), b_pattern.flatten(

#Create the neural network and train it using the training patter
network = HopfieldNetwork(35)

hebbian_training(network, input_patterns)

#Create the test patterns by using the training patterns and addi
#and use the neural network to denoise them
a_test =  a_pattern.flatten()

for i in range(4):
    p = randint(0, 34)
    a_test[p] *= -1

a_result = network.run(a_test)

a_result.shape = (7, 5)
a_test.shape = (7, 5)

b_test =  b_pattern.flatten()

for i in range(4):
    p = randint(0, 34)
    b_test[p] *= -1

b_result = network.run(b_test)

b_result.shape = (7, 5)
b_test.shape = (7, 5)

c_test =  c_pattern.flatten()

for i in range(4):
    p = randint(0, 34)
    c_test[p] *= -1

c_result = network.run(c_test)

c_result.shape = (7, 5)
c_test.shape = (7, 5)
```

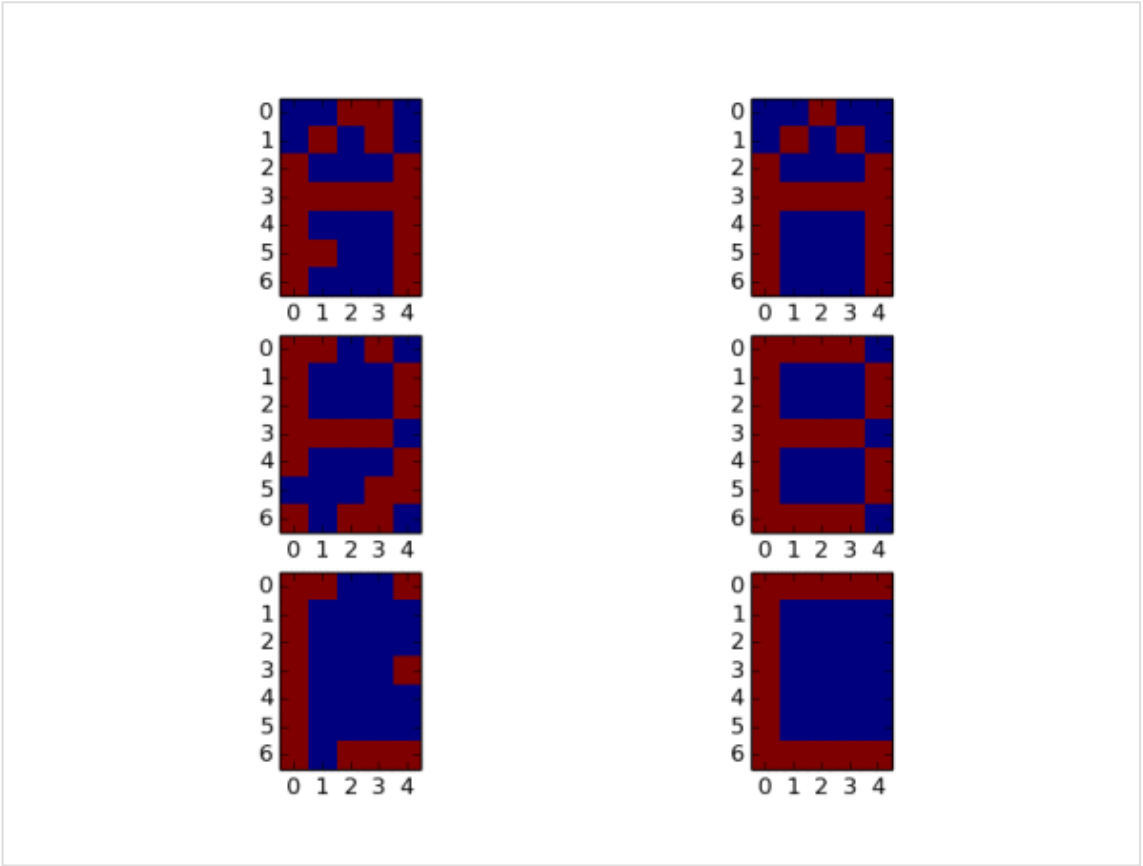
```
#Show the results
plt.subplot(3, 2, 1)
plt.imshow(a_test, interpolation="nearest")
plt.subplot(3, 2, 2)
plt.imshow(a_result, interpolation="nearest")

plt.subplot(3, 2, 3)
plt.imshow(b_test, interpolation="nearest")
plt.subplot(3, 2, 4)
plt.imshow(b_result, interpolation="nearest")

plt.subplot(3, 2, 5)
plt.imshow(c_test, interpolation="nearest")
plt.subplot(3, 2, 6)
plt.imshow(c_result, interpolation="nearest")

plt.show()
```

After we run the above script the result we get is an image with 3 rows and 2 columns. On the first column we see the noisy input data while on the second column we see the result returned by the neural network. As we can see it managed to recognize the letters A, B and C from the corrupted input.



Github: hopfieldnet