Neural Networks
Lab 4

Juan Jose Mendez Torrero (s3542416)
Sharif Hamed (s2562677)

May 23, 2018

1 Implementing a Hopfield network in MATLAB

1. In order to compute the weight matrix W, first, we initialize to a matrix of zeros, then, we use a for loop in order to get the data from each example given. Furthermore, we have used another for loop to create the matrix. The code looks like following:

```
weights = zeros(size_examples);
for p=1:n_examples
  for i=1:size_examples
    for j= 1:size_examples
      weights(i, j) = weights(i,j)+vector_data(p,i)*vector_data(p,j);
     end
    end
end
```

Listing 1: Computing the weight matrix

To delete the connection with each node with itself, we have to write the following code:

```
for i = 1:size(weights,1)
  for j = 1:size(weights,2)
    if i==j
      weights(i,j)=0;
    end
  end
end
```

Listing 2: Itself connections to 0

The resulting matrix has a size of 25x25, and it looks like this:

```
0 -1 -1 -3 -1 1 1 -1 3 1 3 -1 1 -1 1 3 -1 -1 1 -1 1 -3 -3 -1 1
-1 0 3 1 -1 1 -3 -1 -1 -3 -1 3 1 3 -3 -1 -1 -1 1 -1 1 1 1 3 -3
-1 3 0 1 -1 1 -3 -1 -1 -3 -1 3 1 3 -3 -1 -1 -1 1 -1 1 1 1 3 -3
-3 1 1 0 1 -1 -1 1 -3 -1 -3 1 -1 1 -1 -3 1 1 -1 1 -1 3 3 1 -1
-1 -1 -1 1 0 1 1 -1 -1 1 -1 -1 1 -1 1 -1 -1 -1 -3 3 1 1 1 -1 1
1 1 1 -1 1 0 -1 -3 1 -1 1 1 3 1 -1 1 -3 -3 -1 1 3 -1 -1 1 -1
1 -3 -3 -1 1 -1 0 1 1 3 1 -3 -1 -3 3 1 1 1 -1 1 -1 -1 -1 -3 3
-1 -1 -1 1 -1 -3 1 0 -1 1 -1 -1 -3 -1 1 -1 3 3 1 -1 -3 1 1 -1 1
3 -1 -1 -3 -1 1 1 -1 0 1 3 -1 1 -1 1 3 -1 -1 1 -1 1 -3 -3 -1 1
1 -3 -3 -1 1 -1 3 1 1 0 1 -3 -1 -3 3 1 1 1 -1 1 -1 -1 -1 -3 3
3 -1 -1 -3 -1 1 1 -1 3 1 0 -1 1 -1 1 3 -1 -1 1 -1 1 -3 -3 -1 1
```

```
-1 3 3 1 -1 1 -3 -1 -1 -3 -1 0 1 3 -3 -1 -1 -1 1 -1 1 1 1 3 -3
1 1 1 -1 1 3 -1 -3 1 -1 1 1 0 1 -1 1 -3 -3 -1 1 3 -1 -1 1 -1
-1 3 3 1 -1 1 -3 -1 -1 -3 -1 3 1 0 -3 -1 -1 -1 1 -1 1 1 1 3 -3
1 -3 -3 -1 1 -1 3 1 1 3 1 -3 -1 -3 0 1 1 1 -1 1 -1 -1 -1 -3 3
3 -1 -1 -3 -1 1 1 -1 3 1 3 -1 1 -1 1 0 -1 -1 1 -1 1 -3 -3 -1 1
-1 -1 -1 1 -1 -3 1 3 -1 1 -1 -1 -3 -1 1 -1 0 3 1 -1 -3 1 1 -1 1
-1 -1 -1 1 -1 -3 1 3 -1 1 -1 -1 -3 -1 1 -1 3 0 1 -1 -3 1 1 -1 1
1 1 1 -1 -3 -1 -1 1 1 -1 1 1 -1 1 -1 1 1 1 0 -3 -1 -1 -1 1 -1
-1 -1 -1 1 3 1 1 -1 -1 1 -1 -1 1 -1 1 -1 -1 -1 -3 0 1 1 1 -1 1
1 1 1 -1 1 3 -1 -3 1 -1 1 1 3 1 -1 1 -3 -3 -1 1 0 -1 -1 1 -1
-3 1 1 3 1 -1 -1 1 -3 -1 -3 1 -1 1 -1 -3 1 1 -1 1 -1 0 3 1 -1
-3 1 1 3 1 -1 -1 1 -3 -1 -3 1 -1 1 -1 -3 1 1 -1 1 -1 3 0 1 -1
-1 3 3 1 -1 1 -3 -1 -1 -3 -1 3 1 3 -3 -1 -1 -1 1 -1 1 1 1 0 -3
1 -3 -3 -1 1 -1 3 1 1 3 1 -3 -1 -3 3 1 1 1 -1 1 -1 -1 -1 -3 0
```

2. In order to get the value of the activations, we have to know that $a_i = \Sigma w_{ij} \cdot x_i$, hence, we can compute the activation vector as follows:

```
% Compute the new activation
activation =  weights * activation;
```

Listing 3: Activation calculation

After this, we have to apply all the updates. As we know, in synchronous updates, first we have to calculate all the activation in all nodes of a specific layer, and then, we have to update each node, having into account the value of the activation. In order to code that, we have done this:

```
% Apply the activation function
if activation(epoch) >=0
  activation(epoch)=1;
else
  activation(epoch)=0;
end
```

Listing 4: Applying updates

2 Questions on the Hopfield implementation

1. We can capture the Hebb's learning rule repetitively with this formula because, when both term have the same sign (positive or negative), they always will have a positive value, nevertheless, if they have opposite sign, then, they will have a negative value. This holds with Hebb's learning rule: *Cells that fire together, wire together. When cells fire apart, their wires depart.*

2. If we omit the normalization, we would have more errors. We have proved it, and we have seen that the error have increased twice times more than before. This is due to the values that the activation gets, between 1 or -1. We can solve this problem just applying the normalization to the weights, they become more realistic predictors.

3. As we know, we can obtain the number of patterns (p) that a Hopfield network can store by given the number of neuron (N) as following:

$$m < \frac{N}{2 * ln(N)}$$

Hence, we can substitute the factor **N** with 25, being the result 3.88 patterns.

4. After a lot of attempts, we have seen that the networks is not able to store 4 patterns. This is because, theoretically 4 patterns is impossible, the network almost approaches the solution, but it does not reach it perfectly. Hence, we have decided that the apropppiate number of patterns should be 3.

3 Noise and spurious states

1. The network is able to recover the pattern from the noisy input because, the weight are based on the ratio between the pixels in every training image. With these weight, the network adjusts the noised input with it weights, making the network to evolve toward the lowest possible energy. Furthermore, with the number of epoch that we have used, therefore it does not always reach the solution.
2. As we can see in Figure 1, there is a pattern (in letters M, S and T) that the network returns which does not correspond to the desired output. This would not happened with an asynchronous update, this is because, when we use thus, the network update the weights one by one instead of all at the same time.

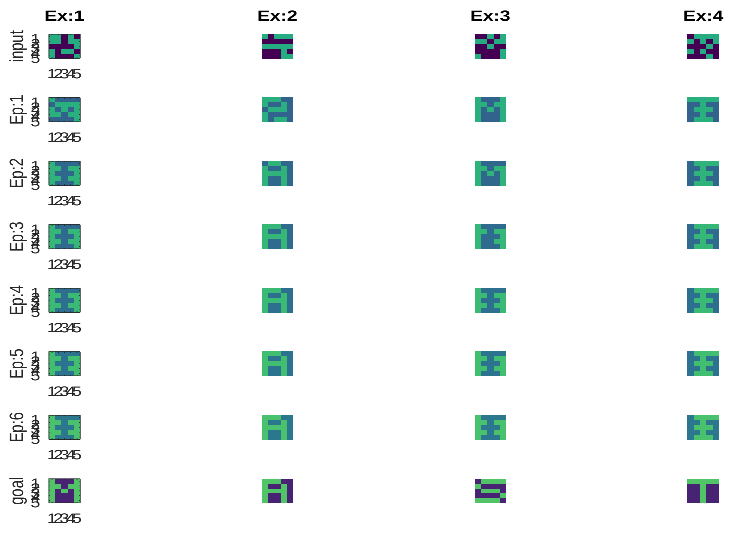

Figure 1: Results of the Hopfield network

3. What happens with the input is, that the input is inverted after the noise is added. As we could observe, with higher noise the network seems to make a better classification with the trained result.

4. The inverted pattern can be also stored in the network because of the negative values in the weights matrix. This means that if we have been already working with negative values by considering the input as a positive input.