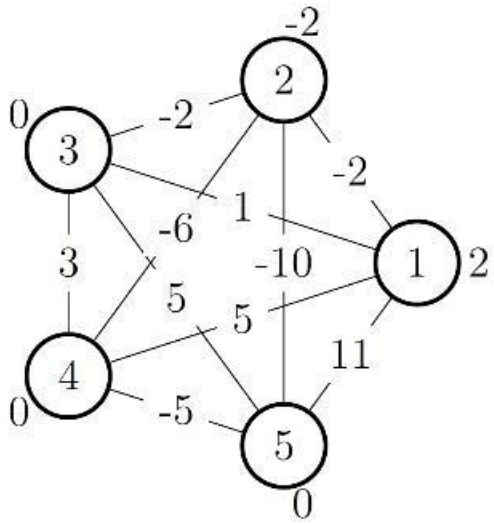
NN EX 4

November 28, 2022

S.M. Erfan Moosavi M.

TABLE OF CONTENTS

1. **Answer the question according to the given Hopfield Network:**



- ***Write down W and Theta:***

W	**1**	**2**	**3**	**4**	**5**
1	0	-2	1	5	11
2	-2	0	-2	-6	-10
3	1	-2	0	3	5
4	5	-6	3	0	-5
5	11	-10	5	-5	0
Theta	***2***	***-2***	***0***	***0***	***0***

- ***Consider the following weight matrix (W), bias vector (theta) and initial state (Z_0):***

- ***Update the state Asynchronously:***

Selecting neuron 1 and 2 and update them results in stable state in the first iteration. (I know in async update, the neuron should be selected randomly, but I asked from Dr. Mozayani and he said its ok to select them in order and also I wanted to study more neurons as initial neuron.

Selecting <u>first</u> Neuron:

z0: [-1 -1 -1 -1 1 1]

z0_1: -1

b1: -5

w1: [0 1 0 0 -4 -2]

Sigma: -7

-7 <= -5 => z0 = -1

z1: [-1 -1 -1 -1 1 1]

Stable state reached.

Selecting <u>second</u> Neuron:

z1: [-1 -1 -1 -1 1 1]

z1_2: -1

b2: 0

w2: [1 0 -4 3 4 -4]

Sigma: 0

0 <= 0 => z1 = -1

z2: [-1 -1 -1 -1 1 1]

Stable state reached.

Now let’s try it on the third neuron:

Selecting third Neuron:

z0: [-1 -1 -1 -1 1 1]

Z0_3: -1

b3: 2

w3: [0 -4 0 -7 -6 -2]

Sigma: 3

3 > 2 => z2 = 1

z1: [-1 -1 1 -1 1 1]

This state is different so let choose the next neuron:

z1: [-1 -1 1 -1 1 1]

z1_4: -1

b4: 1

w4: [0 3 -7 0 -4 3]

Sigma: -11

-11 <= 1 => z3 = -1

z2: [-1 -1 1 -1 1 1]

Stable state reached.

Selecting the third neuron, will result in stable result in two steps (of course if you select the fourth neuron in second step.)

- ***Update the state Synchronously:***

Running the algorithm Synchronously won't end up in a stable sate. (It does not halt). It stuck between these two states:

1,1,-1,-1,-1,-1

1,1,1,1,1,-1

Let's have a look at a list of states after each iteration:

(Recurring states are highlighted)

	N1	N2	N3	N4	N5	N6
0	-1	-1	-1	-1	1	1
1	-1	-1	1	1	1	-1
2	1	1	-1	-1	-1	1
3	1	-1	1	1	1	-1
4	**1**	**1**	**-1**	**-1**	**-1**	**-1**
5	**1**	**1**	**1**	**1**	**1**	**-1**
6	1	1	-1	-1	-1	-1
7	1	1	1	1	1	-1

After iteration 5, we stuck in a loop and states are just repeats.

For sake of completeness, I also added the steps that I took to reach to step 7:

z0: [-1 -1 -1 -1 1 1]	**z0_1: -1**	**b1: -5**	**w1: [0 1 0 0 -4 -2]**	**Sigma: -7**	**-7 <= -5 => z0 = -1**
z0: [-1 -1 -1 -1 1 1]	z0_2: -1	b2: 0	w2: [1 0 -4 3 4 -4]	Sigma: 0	0 <= 0 => z1 = -1
z0: [-1 -1 -1 -1 1 1]	z0_3: -1	b3: 2	w3: [0 -4 0 -7 -6 -2]	Sigma: 3	3 > 2 => z2 = 1
z0: [-1 -1 -1 -1 1 1]	z0_4: -1	b4: 1	w4: [0 3 -7 0 -4 3]	Sigma: 3	3 > 1 => z3 = 1
z0: [-1 -1 -1 -1 1 1]	z0_5: 1	b5: 0	w5: [-4 4 -6 -4 0 -6]	Sigma: 4	4 > 0 => z4 = 1
z0: [-1 -1 -1 -1 1 1]	z0_6: 1	b6: 0	w6: [-2 -4 -2 3 -6 0]	Sigma: -1	-1 <= 0 => z5 = -1
----------(iter 1)----------					
Current z: [-1 -1 -1 -1 1 1]					
New z: [-1 -1 1 1 1 -1]					
z1: [-1 -1 1 1 1 -1]	z1_1: -1	b1: -5	w1: [0 1 0 0 -4 -2]	Sigma: -3	-3 > -5 => z0 = 1
z1: [-1 -1 1 1 1 -1]	z1_2: -1	b2: 0	w2: [1 0 -4 3 4 -4]	Sigma: 6	6 > 0 => z1 = 1
z1: [-1 -1 1 1 1 -1]	z1_3: 1	b3: 2	w3: [0 -4 0 -7 -6 -2]	Sigma: -7	-7 <= 2 => z2 = -1
z1: [-1 -1 1 1 1 -1]	z1_4: 1	b4: 1	w4: [0 3 -7 0 -4 3]	Sigma: -17	-17 <= 1 => z3 = -1
z1: [-1 -1 1 1 1 -1]	z1_5: 1	b5: 0	w5: [-4 4 -6 -4 0 -6]	Sigma: -4	-4 <= 0 => z4 = -1

z1: [-1 -1 1 1 1 -1]	z1_6: -1	b6: 0	w6: [-2 -4 -2 3 -6 0]	Sigma: 1	1 > 0 => z5 = 1
	----------(iter 2)----------				
	Current z: [-1 -1 1 1 1 -1]				
	New z: [1 1 -1 -1 -1 1]				
z2: [1 1 -1 -1 -1 1]	z2_1: 1	b1: -5	w1: [0 1 0 0 -4 -2]	Sigma: 3	3 > -5 => z0 = 1
z2: [1 1 -1 -1 -1 1]	z2_2: 1	b2: 0	w2: [1 0 -4 3 4 -4]	Sigma: -6	-6 <= 0 => z1 = -1
z2: [1 1 -1 -1 -1 1]	z2_3: -1	b3: 2	w3: [0 -4 0 -7 -6 -2]	Sigma: 7	7 > 2 => z2 = 1
z2: [1 1 -1 -1 -1 1]	z2_4: -1	b4: 1	w4: [0 3 -7 0 -4 3]	Sigma: 17	17 > 1 => z3 = 1
z2: [1 1 -1 -1 -1 1]	z2_5: -1	b5: 0	w5: [-4 4 -6 -4 0 -6]	Sigma: 4	4 > 0 => z4 = 1
z2: [1 1 -1 -1 -1 1]	z2_6: 1	b6: 0	w6: [-2 -4 -2 3 -6 0]	Sigma: -1	-1 <= 0 => z5 = -1
	----------(iter 3)----------				
	Current z: [1 1 -1 -1 -1 1]				
	New z: [1 -1 1 1 1 -1]				
z3: [1 -1 1 1 1 -1]	z3_1: 1	b1: -5	w1: [0 1 0 0 -4 -2]	Sigma: -3	-3 > -5 => z0 = 1
z3: [1 -1 1 1 1 -1]	z3_2: -1	b2: 0	w2: [1 0 -4 3 4 -4]	Sigma: 8	8 > 0 => z1 = 1
z3: [1 -1 1 1 1 -1]	z3_3: 1	b3: 2	w3: [0 -4 0 -7 -6 -2]	Sigma: -7	-7 <= 2 => z2 = -1
z3: [1 -1 1 1 1 -1]	z3_4: 1	b4: 1	w4: [0 3 -7 0 -4 3]	Sigma: -17	-17 <= 1 => z3 = -1
z3: [1 -1 1 1 1 -1]	z3_5: 1	b5: 0	w5: [-4 4 -6 -4 0 -6]	Sigma: -12	-12 <= 0 => z4 = -1
z3: [1 -1 1 1 1 -1]	z3_6: -1	b6: 0	w6: [-2 -4 -2 3 -6 0]	Sigma: -3	-3 <= 0 => z5 = -1
	----------(iter 4)----------				
	Current z: [1 -1 1 1 1 -1]				
	New z: [1 1 -1 -1 -1 -1]				
z4: [1 1 -1 -1 -1 -1]	z4_1: 1	b1: -5	w1: [0 1 0 0 -4 -2]	Sigma: 7	7 > -5 => z0 = 1
z4: [1 1 -1 -1 -1 -1]	z4_2: 1	b2: 0	w2: [1 0 -4 3 4 -4]	Sigma: 2	2 > 0 => z1 = 1
z4: [1 1 -1 -1 -1 -1]	z4_3: -1	b3: 2	w3: [0 -4 0 -7 -6 -2]	Sigma: 11	11 > 2 => z2 = 1
z4: [1 1 -1 -1 -1 -1]	z4_4: -1	b4: 1	w4: [0 3 -7 0 -4 3]	Sigma: 11	11 > 1 => z3 = 1
z4: [1 1 -1 -1 -1 -1]	z4_5: -1	b5: 0	w5: [-4 4 -6 -4 0 -6]	Sigma: 16	16 > 0 => z4 = 1
z4: [1 1 -1 -1 -1 -1]	z4_6: -1	b6: 0	w6: [-2 -4 -2 3 -6 0]	Sigma: -1	-1 <= 0 => z5 = -1
	----------(iter 5)----------				
	Current z: [1 1 -1 -1 -1 -1]				
	New z: [1 1 1 1 1 -1]				
z5: [1 1 1 1 1 -1]	z5_1: 1	b1: -5	w1: [0 1 0 0 -4 -2]	Sigma: -1	-1 > -5 => z0 = 1
z5: [1 1 1 1 1 -1]	z5_2: 1	b2: 0	w2: [1 0 -4 3 4 -4]	Sigma: 8	8 > 0 => z1 = 1
z5: [1 1 1 1 1 -1]	z5_3: 1	b3: 2	w3: [0 -4 0 -7 -6 -2]	Sigma: -15	-15 <= 2 => z2 = -1
z5: [1 1 1 1 1 -1]	z5_4: 1	b4: 1	w4: [0 3 -7 0 -4 3]	Sigma: -11	-11 <= 1 => z3 = -1
z5: [1 1 1 1 1 -1]	z5_5: 1	b5: 0	w5: [-4 4 -6 -4 0 -6]	Sigma: -4	-4 <= 0 => z4 = -1

z5: [1 1 1 1 1 -1]	z5_6: -1	b6: 0	w6: [-2 -4 -2 3 -6 0]	Sigma: -11	-11 <= 0 => z5 = -1
----------(iter 6)----------					
Current z: [1 1 1 1 1 -1]					
New z: [1 1 -1 -1 -1 -1]					
z6: [1 1 -1 -1 -1 -1]	z6_1: 1	b1: -5	w1: [0 1 0 0 -4 -2]	Sigma: 7	7 > -5 => z0 = 1
z6: [1 1 -1 -1 -1 -1]	z6_2: 1	b2: 0	w2: [1 0 -4 3 4 -4]	Sigma: 2	2 > 0 => z1 = 1
z6: [1 1 -1 -1 -1 -1]	z6_3: -1	b3: 2	w3: [0 -4 0 -7 -6 -2]	Sigma: 11	11 > 2 => z2 = 1
z6: [1 1 -1 -1 -1 -1]	z6_4: -1	b4: 1	w4: [0 3 -7 0 -4 3]	Sigma: 11	11 > 1 => z3 = 1
z6: [1 1 -1 -1 -1 -1]	z6_5: -1	b5: 0	w5: [-4 4 -6 -4 0 -6]	Sigma: 16	16 > 0 => z4 = 1
z6: [1 1 -1 -1 -1 -1]	z6_6: -1	b6: 0	w6: [-2 -4 -2 3 -6 0]	Sigma: -1	-1 <= 0 => z5 = -1
----------(iter 7)----------					
Current z: [1 1 -1 -1 -1 -1]					
New z: [1 1 1 1 1 -1]					

2. **Calculate a Hopfield weight matrix for the following two inputs:**

$$x1 = (1,-1,1,-1,1,1)$$

$$x2 = (1,1,1,-1,-1,-1)$$

To do so, we have to use the following formula:

$$w_{ij} = \sum_{k=1}^{P} x_i^k x_j^k$$

This means, to calculate the weight between neuron I and J, you should multiply the corresponding indices in input vectors and sum the results of multiplications.

Doing this will result in the following matrix:

0	0	2	-2	0	0
0	0	0	0	-2	-2
2	0	0	-2	0	0
-2	0	-2	0	0	0
0	-2	0	0	0	2
0	-2	0	0	2	0

After this, the question asked us to prove that those inputs are stable states in our network, so we must fed them to the network and if the network state didn't change after one iteration, it proves those are stable states.

For X1:

z0: [1 -1 1 -1 1 1]	**z0_1: 1**	**b1: 0.0**	**w1: [0. 0. 2. -2. 0. 0.]**	**Sigma: 4.0**	**4.0 > 0.0 => z0 = 1**
z0: [1 -1 1 -1 1 1]	z0_2: -1	b2: 0.0	w2: [0. 0. 0. 0. -2. -2.]	Sigma: -4.0	-4.0 <= 0.0 => z1 = -1
z0: [1 -1 1 -1 1 1]	z0_3: 1	b3: 0.0	w3: [2. 0. 0. -2. 0. 0.]	Sigma: 4.0	4.0 > 0.0 => z2 = 1
z0: [1 -1 1 -1 1 1]	z0_4: -1	b4: 0.0	w4: [-2. 0. -2. 0. 0. 0.]	Sigma: -4.0	-4.0 <= 0.0 => z3 = -1
z0: [1 -1 1 -1 1 1]	z0_5: 1	b5: 0.0	w5: [0. -2. 0. 0. 0. 2.]	Sigma: 4.0	4.0 > 0.0 => z4 = 1
z0: [1 -1 1 -1 1 1]	z0_6: 1	b6: 0.0	w6: [0. -2. 0. 0. 2. 0.]	Sigma: 4.0	4.0 > 0.0 => z5 = 1

Current z: [1 -1 1 -1 1 1]
New z: [1 -1 1 -1 1 1]

The state remained the same.

For X2:

z0: [1 1 1 -1 -1 -1]	**z0_1: 1**	**b1: 0.0**	**w1: [0. 0. 2. -2. 0. 0.]**	**Sigma: 4.0**	**4.0 > 0.0 => z0 = 1**
z0: [1 1 1 -1 -1 -1]	z0_2: 1	b2: 0.0	w2: [0. 0. 0. 0. -2. -2.]	Sigma: 4.0	4.0 > 0.0 => z1 = 1
z0: [1 1 1 -1 -1 -1]	z0_3: 1	b3: 0.0	w3: [2. 0. 0. -2. 0. 0.]	Sigma: 4.0	4.0 > 0.0 => z2 = 1
z0: [1 1 1 -1 -1 -1]	z0_4: -1	b4: 0.0	w4: [-2. 0. -2. 0. 0. 0.]	Sigma: -4.0	-4.0 <= 0.0 => z3 = -1
z0: [1 1 1 -1 -1 -1]	z0_5: -1	b5: 0.0	w5: [0. -2. 0. 0. 0. 2.]	Sigma: -4.0	-4.0 <= 0.0 => z4 = -1
z0: [1 1 1 -1 -1 -1]	z0_6: -1	b6: 0.0	w6: [0. -2. 0. 0. 2. 0.]	Sigma: -4.0	-4.0 <= 0.0 => z5 = -1

Current z: [1 1 1 -1 -1 -1]
New z: [1 1 1 -1 -1 -1]

The state remained the same.

3. Finish the provided notebook:

mat2vec:

This function gets a matrix in its input and convert it to a vector, to do so, we used flatten() function of numpy which is for the same purpose.

create_W:

This function is used to create (initiate) weights of the Hopfield network, using the following formula:

$$w_{ij} = \sum_{k=1}^{P} x_i^k x_j^k$$

This formula can be implemented using two simple loops and a condition to control i==j.

readImg2array:

This function is to read an image from a file and convert it to a binary image, to do so, we need to define a threshold and set values below the threshold to minimum pixel value (here -1) and values above the threshold to the maximum pixel value (here 1).

To do this we used numpy masking and array comparison to find the values below and above the threshold and update them accordingly.

array2img:

This function just converts a numpy array to an image and then store it to a file.

Update:

This is the asynchronous update function which updates the states of the network, in each iteration, it randomly selects a neuron and update its state, then apply the new state to state vector of the network and start over by selecting a new neuron.

It does so, for a number of iterations because if we set the algorithm stop criteria to be founding the same state, it might not finish the job due to big number of neurons.

We have to run this algorithm for about 30,000 iterations to end up with a seemingly good result.

update_sync:

this is the synchronous version of the above algorithm, which first calculate the new states of every neuron and then update the whole state vector all together.

This algorithm reaches a good result with only 100 steps and with 1000 steps it reaches nearly the exact image.

Hopfield:

This is the driver function of the whole algorithm which we will use for first reading the test and train image, the converting them to binary image vectors (I've stored the binary images for sake of comparison), after that it calculates the weight matrix and run the algorithm, here we can call sync or async version of the algorithm.

The final results are as follows:

Figure 1: ASYNC Result image, Figure 2: SYNC Result image

Figure 3: Test Image, Figure 4: Test image binary

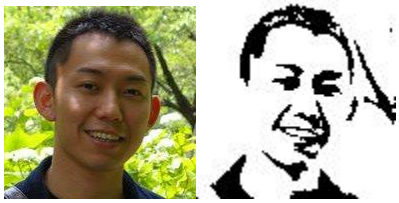


Figure 5: Train Image, Figure 6: Train Image binary

4. Run LVQ notebook.

In the first cell, I imported the required libraries: NumPy, Matplotlib

In the next cell, with genfromtxt function of numpy, we read and parsed the provided dataset CSV file.

After that using numpy indexing and slicing, we considered the first two samples as prototypes and rest of the samples as training data.

In the "show Plot data in 3D befor train" cell, by using Matplotlib scatter plot, we plotted all data (train and prototype) in 3D plane, we used color coding to separate classes from each other.

In "LVQ1 Model Implementation From Scratch" cell, we implemented a LVQ model from scratch.

LVQ class:

For specified number of epochs, we iterate over training data and in each iteration, we select the corresponding sample data, for this sample data, we calculate the winner label and compare it to the current label (Actual) of the sample, after that we update the weights accordingly; which means calculating an update value and add this to weight if winner label predicted correctly or subtract it from weights otherwise.

After this, we initiate an object of this class and train it using our training data.

In the next cell, we plotted the W to see the changes.

And in the final cell, we called winner_label function of LVQ model to determine the label of the given sample:

[[0.304 1.488 1.408]] => 1