



CI Homework 3

👉 M.Amin Ghasvari 97521432

Question1 ?

Part I ⭐

Calculate the weight

Adding the matrices together

Is it stable or not

Result

Part II ⭐

Part III ⭐

Question 2 ?

Part I ⭐

Part II ⭐

Part III ⭐

Question1 ?

Part I ⭐

The patterns have 4 numbers that are 0 or 1. So we need 4 neurons.

We have four pattern in this part. So $k = 4$. In the first step we are going to calculate weight matrix for each one of these patterns.

Calculate the weight

The main equation to create this matrix is $w_{ij}^k = x_i^k \cdot x_j^k$ So we are going to apply all of them to our weights.

$$\bullet P_1 = (1 \quad 1 \quad 1 \quad 1)$$

$$w^1 = \begin{pmatrix} X & 1 & 1 & 1 \\ 1 & X & 1 & 1 \\ 1 & 1 & X & 1 \\ 1 & 1 & 1 & X \end{pmatrix}$$

- $P_2 = \begin{pmatrix} -1 & -1 & -1 & -1 \end{pmatrix}$

$$w^2 = \begin{pmatrix} X & 1 & 1 & 1 \\ 1 & X & 1 & 1 \\ 1 & 1 & X & 1 \\ 1 & 1 & 1 & X \end{pmatrix}$$

- $P_3 = \begin{pmatrix} 1 & 1 & -1 & -1 \end{pmatrix}$

$$w^3 = \begin{pmatrix} X & 1 & -1 & -1 \\ 1 & X & -1 & -1 \\ -1 & -1 & X & 1 \\ -1 & -1 & 1 & X \end{pmatrix}$$

- $P_4 = \begin{pmatrix} -1 & -1 & 1 & 1 \end{pmatrix}$

$$w^4 = \begin{pmatrix} X & 1 & -1 & -1 \\ 1 & X & -1 & -1 \\ -1 & -1 & X & 1 \\ -1 & -1 & 1 & X \end{pmatrix}$$

Adding the matrices together

Then we should sum all of these values $w = \sum w_{ij}^k$

So after adding these matrices together the final weight matrix is created.

$$w = \begin{pmatrix} X & 4 & 0 & 0 \\ 4 & X & 0 & 0 \\ 0 & 0 & X & 4 \\ 0 & 0 & 4 & X \end{pmatrix}$$

And this is the final weight matrix for Part I.



We were able to apply the first and the third one. because the others are inverse!

Is it stable or not

To test this model whether it is stable or not, I will find the energy of each input.

We are using $\text{sign}(\sum w_{ij} \cdot a_i - \text{threshold})$ actually. $\text{threshold} = 0$

- $P_1 = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}$

$$P_{11} = 0 + 1 \times 4 + 0 + 0 = 4 > 0 \rightarrow 1$$

$$P_{12} = 1 \times 4 + 0 + 0 + 0 = 4 > 0 \rightarrow 1$$

$$P_{13} = 0 + 0 + 0 + 1 \times 4 = 4 > 0 \rightarrow 1$$

$$P_{14} = 0 + 0 + 1 \times 4 + 0 = 4 > 0 \rightarrow 1$$

It is stable.

- $P_2 = (-1 \quad -1 \quad -1 \quad -1)$

$$P_{21} = 0 - 1 \times 4 + 0 + 0 = -4 < 0 \rightarrow -1$$

$$P_{22} = -1 \times 4 + 0 + 0 + 0 = -4 < 0 \rightarrow -1$$

$$P_{23} = 0 + 0 + 0 - 1 \times 4 = -4 < 0 \rightarrow -1$$

$$P_{24} = 0 + 0 - 1 \times 4 + 0 = -4 < 0 \rightarrow -1$$

It is stable.

- $P_3 = (1 \quad 1 \quad -1 \quad -1)$

$$P_{31} = 0 + 1 \times 4 + 0 + 0 = 4 > 0 \rightarrow 1$$

$$P_{32} = 1 \times 4 + 0 + 0 + 0 = 4 > 0 \rightarrow 1$$

$$P_{33} = 0 + 0 + 0 - 1 \times 4 = -4 < 0 \rightarrow -1$$

$$P_{34} = 0 + 0 - 1 \times 4 + 0 = -4 < 0 \rightarrow -1$$

It is stable.

- $P_4 = (-1 \quad -1 \quad 1 \quad 1)$

$$P_{41} = 0 - 1 \times 4 + 0 + 0 = -4 < 0 \rightarrow -1$$

$$P_{42} = -1 \times 4 + 0 + 0 + 0 = -4 < 0 \rightarrow -1$$

$$P_{43} = 0 + 0 + 0 + 1 \times 4 = 4 > 0 \rightarrow 1$$

$$P_{44} = 0 + 0 + 1 \times 4 + 0 = 4 > 0 \rightarrow 1$$

It is stable.

Result

Then I want to calculate the energy for each one of the patterns using $E = -\sum w_{ij} \cdot o_i \cdot o_j$, and here is the result:

$$E_1 = -4$$

$$E_2 = -4$$

$$E_3 = -4$$

$$E_4 = -4$$

This is actually the minimum. So as you see we can provide a model for this question.

Part II

For this part, the procedure is not much different. We should calculate the weight matrix first.

There is two pattern so $k = 2$

- $P_1 = (1 \quad -1 \quad 1 \quad -1 \quad 1 \quad -1)$
- $P_2 = (1 \quad 1 \quad 1 \quad -1 \quad -1 \quad -1)$

First, we should apply both of these patterns with $w_{ij}^k = x_i^k \cdot x_j^k$, $w = \sum w_{ij}^k$ to create the final weight matrix. Then, we should check stability for each one of the patterns.

Here is the Hopfield class implementation

```
class Hopfield:

    def __init__(self, patterns):
        """
        :param patterns: patterns of this model
        """
        self.patterns = patterns
        self.n_patterns = len(patterns[0])

        # fields that initialized for later assignment
        self.w = None
        self.epoch = None
```

The constructor has the first given patterns.

```
def calculate_weight(self):
    self.w = np.dot(P.T, P)
    np.fill_diagonal(self.w, 0)
```

Then we calculate the final weight in this class.

```
def calculate_closest(self, a, epoch):
    """
```

```

:param a: input of the function
:param epoch: is the number of iteration
:return: the closest choice, energy, and accuracy
"""
activation = np.array(a)
self.epoch = epoch

# initialize lists for maintaining the choices
choices = [activation]
diffs = [math.inf]

for i in range(self.epoch):
    # calculate the activation
    activation = np.sign(np.dot(activation, self.w))
    choices.append(activation)

    # calculate the difference
    diff = self.n_patterns - np.count_nonzero(np.equal(a, activation))
    diffs.append(diff)

    # check whether it is stable or not
    # only check the last 2 items
    if np.count_nonzero(np.equal(choices[-1], choices[-2])) == self.n_patterns:
        if i == 0:
            # it means that the initial input is stable
            print("A stable input is given!")
            break

# calculate the details about closest one
np_choices = np.array(choices)
np_diffs = np.array(diffs)

idx_closest = np.argmin(np_diffs)

closest = np_choices[idx_closest]
energy = -1 * np.sum(self.w * np.outer(closest, closest))
acc = 100 * ((self.n_patterns - np_diffs[idx_closest]) / self.n_patterns)

return closest, energy, acc

```






The main method is the `calculate_closest` it has 2 inputs the first one is the pattern and the other is the number of iteration. For each one of them we calculate the difference and at the end we will find the minimum index of them based on the `diffs` then we calculate the accuracy and energy.

There are four state in question number 2

- $P_1 = (1 \quad -1 \quad 1 \quad -1 \quad 1 \quad -1)$
- $P_2 = (1 \quad 1 \quad 1 \quad -1 \quad -1 \quad -1)$
- $P_3 = (-1 \quad 1 \quad 1 \quad -1 \quad -1 \quad -1)$

Here is a table in addition to the notebook result.

Part II table

 Given patterns	 Asked pattern	 Energy	 Accuracy	 more
<u>P1,P2</u>	P2	-28	100%	A stable input is given!
<u>P1,P2</u>	P3	-38	100%	A stable input is given!
<u>P1,P2,P3</u>	P2	-28	83.33%	
<u>P1,P2,P3</u>	P3	-38	83.33%	

Part III

```
def create_images(font_size):
    font = ImageFont.truetype("segoepr.ttf", font_size)
    for char in letters:
        im = Image.Image().new(font.getmask(char))
        s = (font_size * 2)
        im = im.resize((s, s))
        im.save(f"{char}_{font_size}.bmp")
```

Creates the images based on the font size and save them for further usage

```
def create_noisy_images(arr, s_vs_p, amount):
    noisy_images = []
    for i in range(10):
        image = arr[i]
        out = image
        num_salt = np.ceil(amount * image.size * s_vs_p)
        coord = [np.random.randint(0, i - 1, int(num_salt))
                  for i in image.shape]
        out[coord] = 255
        num_pepper = np.ceil(amount * image.size * (1. - s_vs_p))
        coord = [np.random.randint(0, i - 1, int(num_pepper))
                  for i in image.shape]
        out[coord] = 0
        noisy_image = np.array(image)
        noisy_images.append(noisy_image)
    return noisy_images
```

Creates some noise points based on the given amount in argument.




```
def convert_image_to_sign_matrix(image):
    sign_matrix = np.ones(image.shape)
    sign_matrix[image < 255 // 2] = -1
    return sign_matrix

def convert_images_to_sign_matrices(arr):
    matrices = []
    for i in range(10):
        sign_matrix = convert_image_to_sign_matrix(arr[i])
        matrices.append(sign_matrix)
    return matrices
```

We consider pixels above $255/2$ as $+1$ and the others as -1 .

The final results are here.

Part III table

 Font size	 Noise	 Accuracy
<u>16</u>	10%	85.05%
<u>16</u>	30%	83.47%
<u>16</u>	60%	83.99%
<u>32</u>	10%	85.41%
<u>32</u>	30%	83.72%
<u>32</u>	60%	82.77%
<u>64</u>	10%	86.04%
<u>64</u>	30%	83.99%
<u>64</u>	60%	83.22%

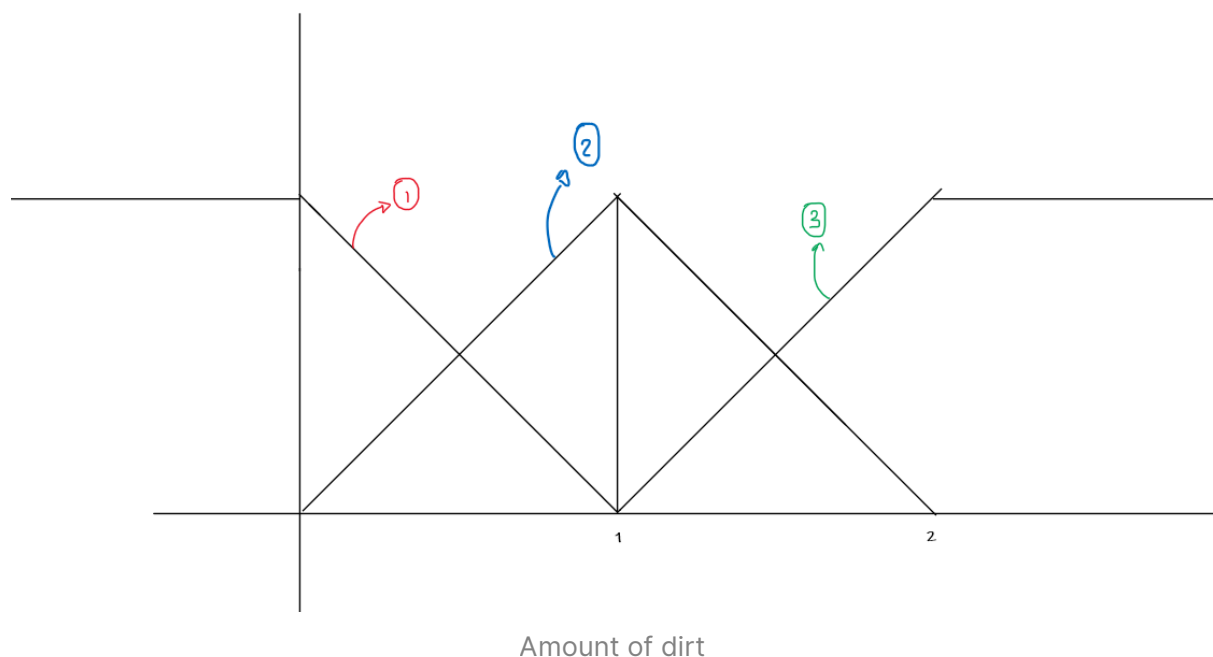
Question 2 ?

Part I

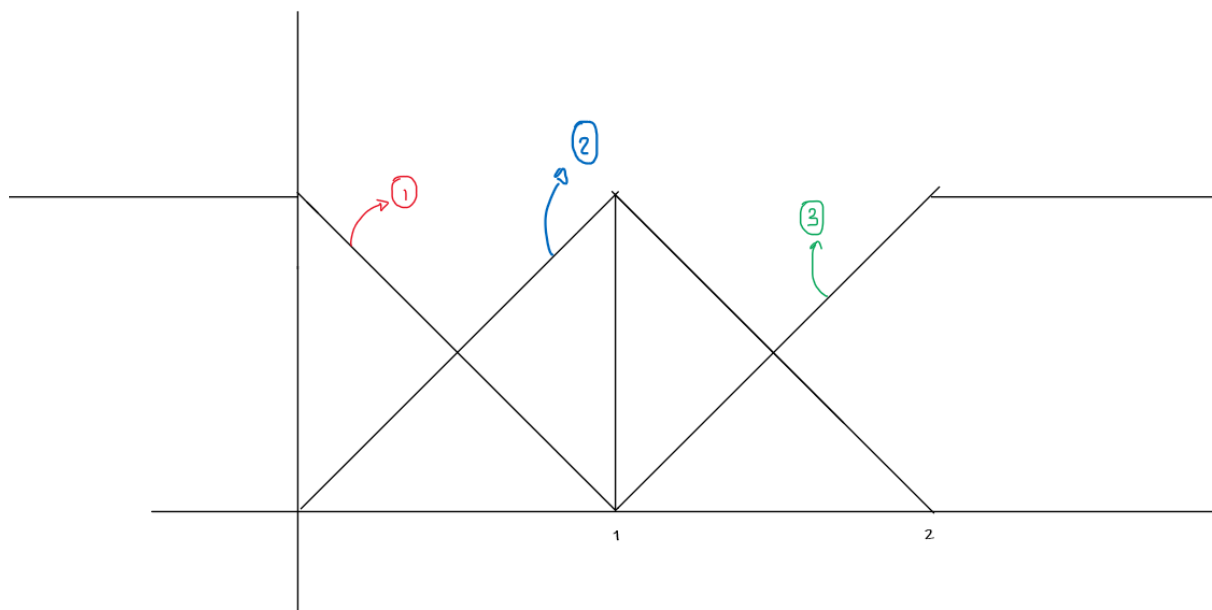
For designing this fuzzy system we use two parameters. The first one is the level and amount of dirt that clothes have. The second one is the type of dirt. For example some times the type could be oil sometimes it could be muddy. Here we just name them dirt one to three. Three levels of dirtiness are also

enough. In the next step, we are going to plot their diagram. First for the amount of dirt.

I choose the low dirt the number 1 or less, the medium dirt 0 to 2, and the high dirt 1 or above. They are all in the bottom diagram numbered from 1 to 3.

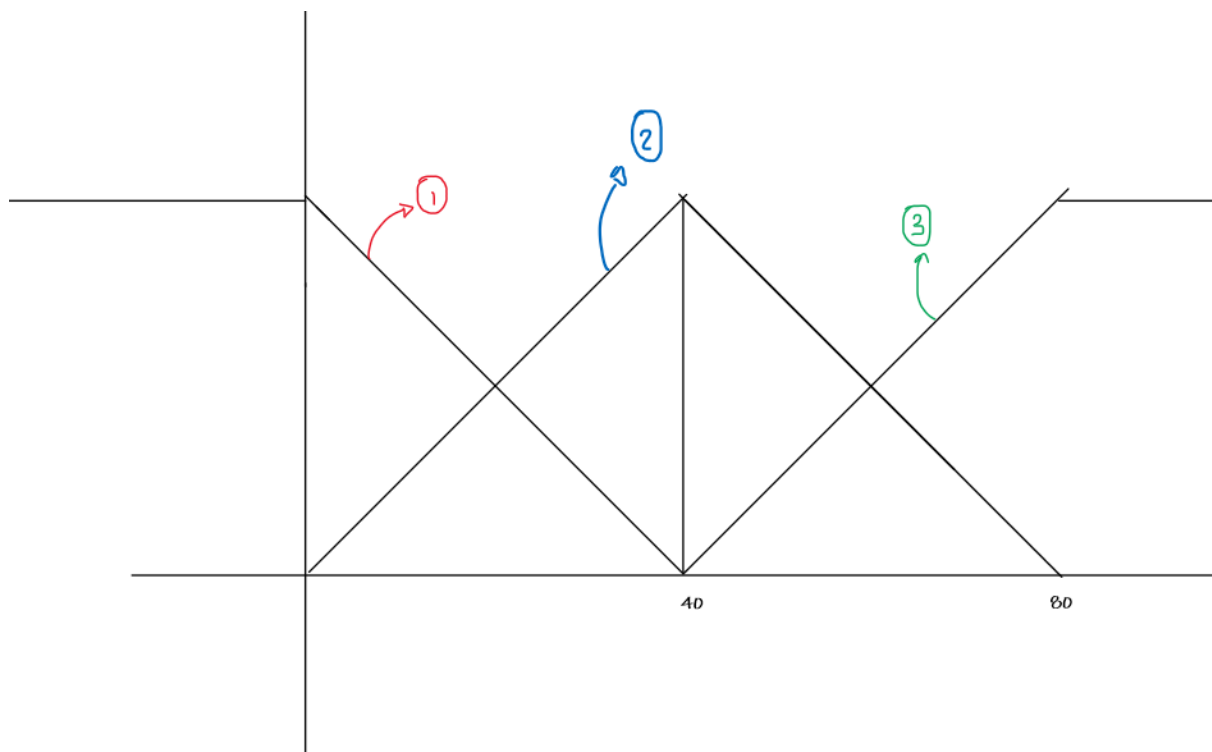


I choose the type of dirt exactly like the previous one. Type 1 of dirt 1 or less, the second dirt that is harder to clean 0 to 2, and the most annoying type of dirt 1 or above. They are all in the bottom diagram numbered from 1 to 3.



Type of dirt

So finally we should have time to wash clothes. It can be little, medium, or much. The little one is 40 minutes or lower, the medium one is between 0 and 80 minutes, and the longest could be 40 or above.



Washing time

So lets define some rules

- If dirt type is 3, it takes much time.
- If dirt type is 2 and amount is medium or above it takes much time to wash.
- If dirt type is 2 and amount is light it takes medium time.
- If dirt type is 1 and amount is light it takes little time to complete its job.
- If dirt type is 1 and amount is medium or above then it takes medium time.

That's all for this system.

Part II

The Larsen method uses the product as a fuzzy implication and max product operator for the composition.

$$B'(-1) = \min(A(3)B(-1), A'(3)) \rightarrow B'(-1) = \min(0.5 \times 0.33, 1) = 0.165$$

$$B'(-2) = \min(A(3)B(-2), A'(3)) \rightarrow B'(-2) = \min(0.5 \times 0.67, 1) = 0.335$$

$$B'(-3) = \min(0.5 \times 1.0, 1) = 0.5$$

$$B'(-4) = \min(0.5 \times 0.5, 1) = 0.25$$

$$B' = KB \text{ that } K = \max(0.165, 0.335, 0.5, 0.25) = 0.5$$

$$B' = 0.5B$$

Part III

We use theta and theta dot as the parameters that we use. We define five terms for theta and seven terms for theta dot and specify the range of them and out they are.

For DEFUZZIFY section we should define some other terms for acting. So here we have 5 terms for right, 5 for left, and a stop term.

At last we write the rules for this system.