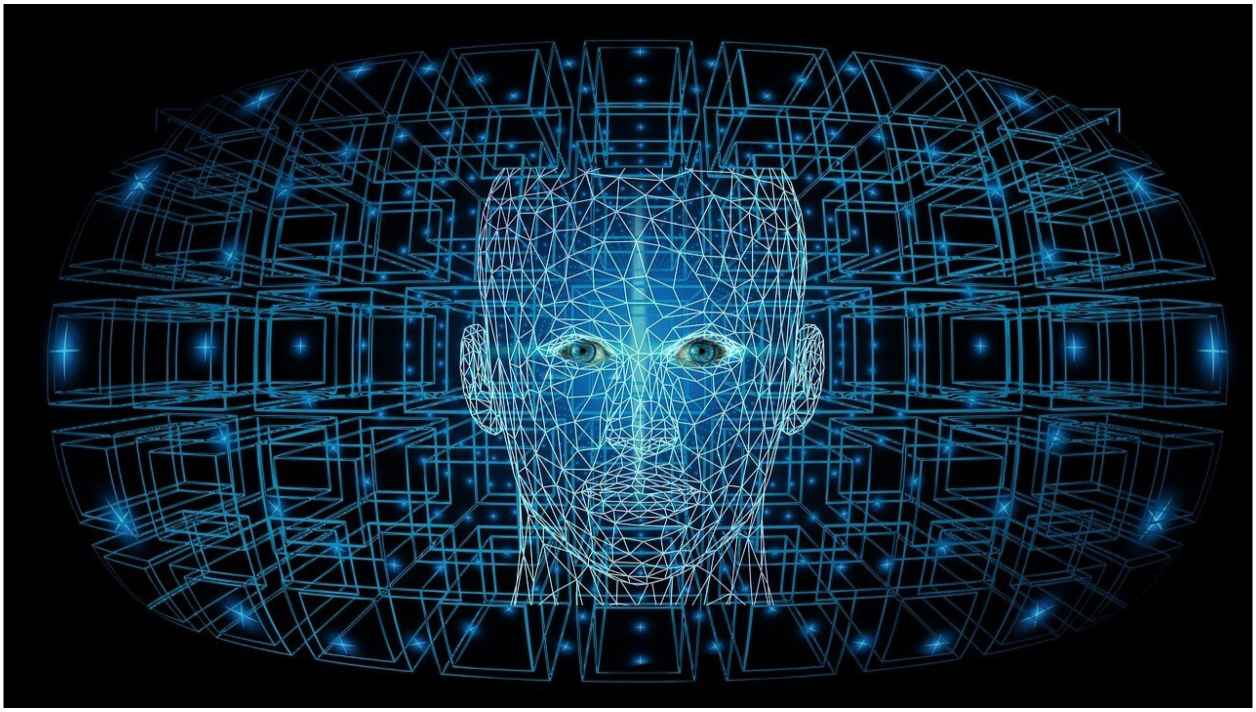


CI HW3

Sohrab Namazinia
97522085

A1.1)

Subject :
Year . Month . Date . ( )

A1.1) we compute the weights, then compute the energy so that we check if they are local minimum or not. then it shows if they can be stored or not.

$P_0 = (1,1,1,1)$, $P_1 = (-1,-1,-1,-1)$, $P_2 = (1,1,-1,-1)$, $P_3 = (-1,-1,1,1)$

$$\left.\begin{aligned}
w_{01} &= 1\times1 + (-1)\times(-1) + 1\times1 + (-1)\times(-1) = 4\\
w_{02} &= 1\times1 + (-1)\times(-1) + 1\times(-1) + (-1)\times(1) = 0\\
w_{03} &= 1\times1 + (-1)\times(-1) + 1\times(-1) + (-1)\times1 = 0\\
w_{12} &= 1\times1 + (-1)\times(-1) + 1\times(-1) + (-1)\times1 = 0\\
w_{13} &= 1\times1 + (-1)\times(-1) + 1\times(-1) + (-1)\times1 = 0\\
w_{23} &= 1\times1 + (-1)\times(-1) + (-1)\times(-1) + 1\times1 = 4
\end{aligned}\right\} \Rightarrow
\begin{array}{c|cccc}
 & 0 & 1 & 2 & 3\\
\hline
0 & 0 & 4 & 0 & 0\\
1 & 4 & 0 & 0 & 0\\
2 & 0 & 0 & 0 & 4\\
3 & 0 & 0 & 4 & 0
\end{array}$$

now, we compute the network energy based on each input from 4 patterns: $E = -\sum_{i,j} w_{i,j}\, o_i\, o_j$

$E([1,1,1,1]) = -(0\times\underbrace{1}_{1\times1} + 4\times\underbrace{1}_{1\times1} + 0\times\underbrace{1}_{1\times1} + 0\times1 + 4\times\underbrace{1}_{1\times1} + \cdots + 4\times1 + 4\times1) = -16$

$E([-1,-1,-1,-1]) = -(0\times\underbrace{1}_{(-1)\times(-1)} + 4\times\underbrace{1}_{(-1)\times(-1)} + \cdots) = -16$

$E([1,1,-1,-1]) = -(1\times1\times0 + 1\times1\times4 + 1\times(-1)\times0 + 1\times(-1)\times0 + \ldots) = -16$

$E([-1,-1,1,1]) = -((-1)\times(-1)\times0 + (-1)\times(-1)\times4 + (-1)\times1\times0 + \ldots) = -16$

for All these 4 patterns, if we flip just 1 bit to get neighbours of them, this flip Action will cause multiplication $(1\times(-1)$ or $(-1)\times1)\times4$ in Energy formula inside the Paranthesis $(-(\,))$, so instead of having $-(16)$, we will have $-(16-\underset{\alpha>0}{\alpha}) \Rightarrow -(\text{smaller number}) \Rightarrow$ bigger number

$\Rightarrow$ bigger energy $\Rightarrow$ All these 4 states are local Minimum $\Rightarrow$ these 4 Patterns can be stored $\Rightarrow$ ✓

PaPCO

A1.2)

My code consists of 4 cells and has been commented on, but I do explain it here too.

Note: for this question, you must upload the "Arial.ttf" font to run it on google colab.

CELL #1:

The first cell is import sections.

CELL #2:

I have constructed the Hopfield network. The Hopfield has a sign function according to what the question wants. It takes the necessary parameters (neuron count, stable states, etc), it has a compute_weight function which computes the weights of the network according to Hopfield Hebbian learning. It also has a run method that has a while loop. In the while loop, by the Hopfield formula, each time we convert our existing pattern to a new one. The new pattern is compared to the old pattern. The conversion is async. If the old & new are the same, it is done and we have gotten a stable state, else, we update the existing pattern and iterate again in the loop.

CELL #3:

In the third cell, I have built the network object, set the parameters as mentioned exactly in the question, and first proved that the pattern (1, 1, 1, -1, -1, -1) is stable by giving it as the initial state and getting to that again.

CELL #4:

I have changed the initial state to (-1, 1, 1, -1, -1, -1) to see the final pattern which will be (1, 1, 1, -1, -1, -1).

A1.3)

My code for this part consists of many cells that have been commented on, but I do explain it here too:

Cell #1: import cell

Cell #2: it contains all necessary methods for solving this question:

build _data(): creates the BMP files with the given font size.

convert_image_to_array(): converts images to grayscale mode so that they can be stored as arrays.

convert_2darr_to_1darr(): it flattens the resulting array that was converted

convert_array_to_bin_array(): converts an array to a binary array with the given limitation that acts as a border. If elements are greater than the border, their value is 1, else 0.

convert_image_to_binarray(): it does all the above together.

add_noise_to_binarray(): it flips N bits of the binary array. N depends on the given parameter noise percentage.

Compare_lists(): a debugging purpose method to see the differences between 2 lists.

Compute_patterns_diff_count(): computes the number of differences between two lists.

compare_patterns(): a debugging purpose method to compare patterns.

find_closest_matching _pattern(): it gets the result and compares it by the patterns, returns the most similar pattern.

run_test(): it does all together. It takes the number of tests as a parameter. It just prints the times when the code diagnoses the result successfully. Finally, it prints the accuracy percentage.

Cell #3:

It contains some parameters that are common through all 6 testing cases. The next cells are just building data and running test with different noises and font sizes:

Cell #4: noise = 60%, font size = 16

Cell #5: noise = 30%, font size = 16

Cell #6: noise = 10%, font size = 16

Cell #7: noise = 60%, font size = 32

Cell #8: noise = 30%, font size = 32

Cell #9: noise = 10%, font size = 32

Cell #10: noise = 60%, font size = 64

Cell #11: noise = 30%, font size = 64

Cell #12: noise = 10%, font size = 64

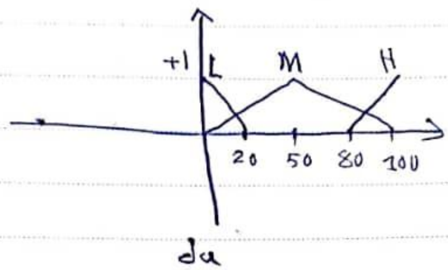
The result of each cell is not fixed, because the noisy cells are given randomly, but here is the result of an attempt to all 9 different cases:

Font size \ Noise	10%	30%	60%
16	70.0%	60.0%	10.0%
32	3.0%	32.0%	42.0%
64	2.0%	37.0%	41.0%

A2.1)

A2.1) first, we must refer to some experts to define our inputs for us:

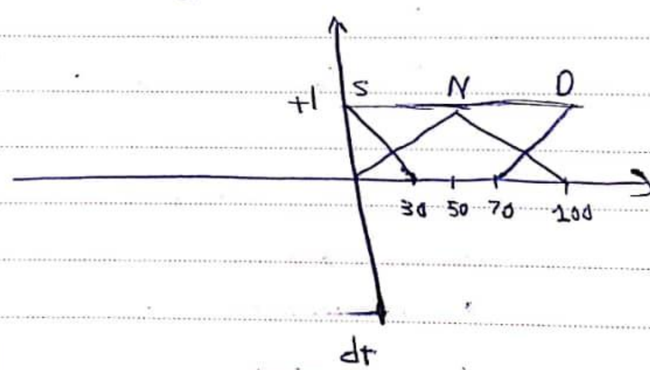
1) dirtiness amount (da): we suppose the experts have categorized it into 3 types: [low, medium, high] with the following scales;



$da \in [0, 100]$

the da unit is %, e.g. 30% of the whole cloth is dirty.

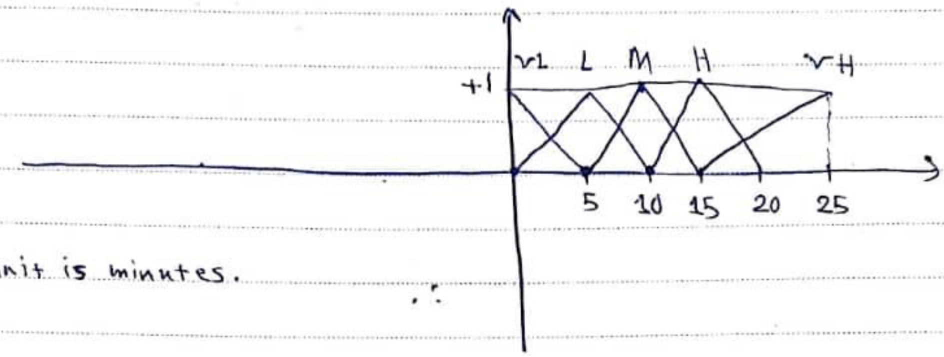
2) dirtiness type: the experts have categorized it into 3 labels that are: sparse, normal, dense. dense dirtiness is much harder to be removed by washing. here are the scales:



$dt \in [0, 100]$

the unit of dt is also percentage, e.g. type 100% is totally dense dirtiness.

now, we need to define our ouput form which is washing time (wt), experts have categorized it into 5 types:
[very low, low, Medium, high, very high], with the following scales:



wt unit is minutes.

$wt \in [0, 25]$

so we have defined the input and output completely, so:
- our fuzzification function is triangular.
- we did fuzzification
- we need to specify a method for fuzzy inference: we chose Mamdani method ⇒ max-min.
- so now we now we know how to slove problems with this fuzzy system with input tuples = (da, dt) ⟶ fuzzification ⟶ Mamdani inference ⟶ Aggregation of outputs of different rules (we need to define rule Base) ⟶ defuzzification by «MOM» method.
- our rules are of the form: if $\underset{da}{A}$ and $\underset{dt}{B}$, then $\underset{wt}{C}$

matching degree is the Min of them.

now we just need to define rule Base table . that is done with the help of experts:

dt \ dq	Low	Medium	High
Sparse	Very Low	Low	Medium
Normal	Low	Medium	High
Dense	Medium	High	very high

output: washing time

So, totally I defined an appropriate fuzzy control for washig machine system.

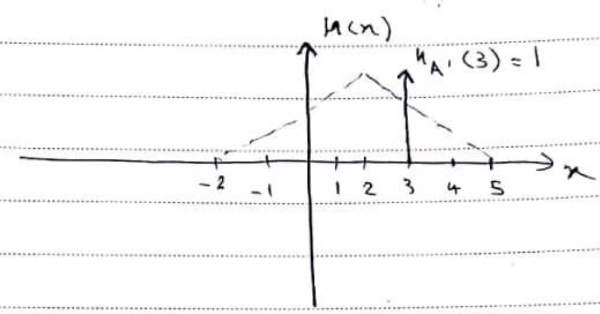
A2.2)

Since the input is a fuzzy number which is 1 in a single point and 0 in others, it can be treated as a normal number. Also if we don't, we still get the same answer.

A2.2) if $x = \tilde{A} \Rightarrow y = \tilde{B}$ we should do the inference by larsen method.

Larsen method: $\mu_{B'}(w) = \bigvee_{i=1}^{n} [a_i \cdot \mu_{B_i}(w)] = \bigvee_{i=1}^{n} \mu_{B'_i}(w)$

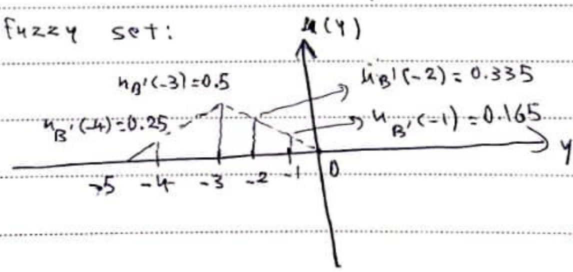
So, now we try to find $\tilde{B}'$



$$\mu_{A'}(x) = \begin{cases} 1 & x = 3 \\ 0 & \text{o.w} \end{cases} \Rightarrow A' = 3$$

$\mu_A(3) = 0.5$ $\Rightarrow$ matching degree $= a = 0.5$

with Larsen method, we multiply a by $\tilde{B}$ fuzzy set to get $\tilde{B}'$ fuzzy set, here is B' fuzzy set:



A2.3)

I studied FCL and so on so that I can be able to write the FCL language file. I defined my fuzzy inputs, which are theta and theta_dot. I check that these two parameters are enough to fix the pendulum. My output is F (changed from f to F so that it works).

Then, I defined their fuzzy subsets. The name of the fuzzy subsets is so clear that there is no need to explain. I got their intervals by trial and error and observing the plotting outputs.

I defined 10 rules, I had defined more rules but some of them were not needed so I removed them. I got the rule from the basis of Physics science that can be gotten by simple observation (no need for formulas).

I tested it and it works properly. In the worst case, it takes about 10 seconds so that it becomes stable. (also sometimes in the first second)

I have uploaded my FCL file too.

"THANKS"