Subject: 23CSE301

Lab Session: 08

Notes:

1. Please read the assignment notes carefully and comply to the guidelines provided.
2. Code should be checked into the GitHub. These details shall be provided in the Lab.
3. If you have not completed the prerequisite assignments, please complete them before the next lab session.

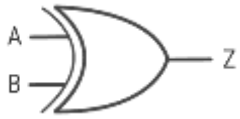
Coding Instructions:

1. The code should be modularized; The asked functionality should be available as a function. Please create multiple functions if needed. However, all functions should be present within a single code block, if you are using Jupyter or Colab notebooks.
2. There should be no print statement within the function. All print statements should be in the main program.
3. Please use proper naming of variables.
4. For lists, strings and matrices, you may use your input values as appropriate.
5. Please make inline documentation / comments as needed within the code blocks.

Main Section (Mandatory):

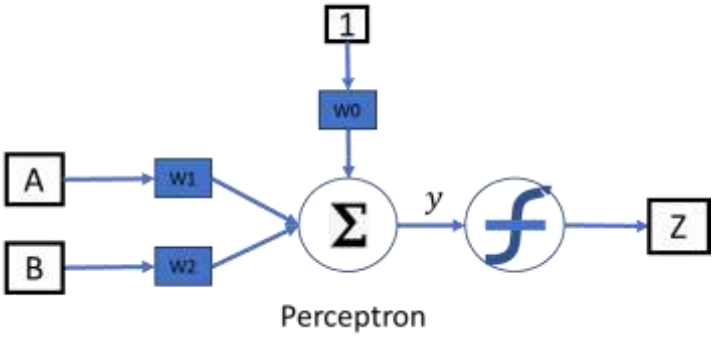
Please use the data associated with your own project.

Ref: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html



AND GATE		
A	**B**	**Z**
0	0	0
0	1	0
1	0	0
1	1	1

XOR GATE		
A	**B**	**Z**
0	0	0
0	1	1
1	0	1
1	1	0



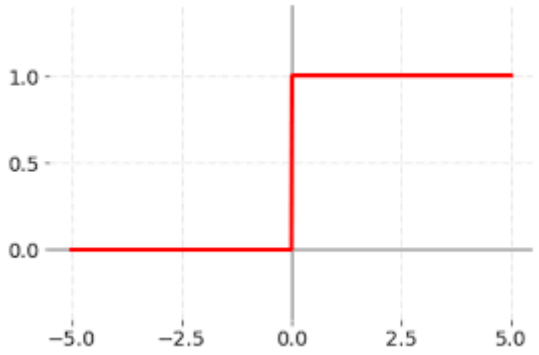
A1. Write your own functions for the following modules:

a) Summation unit
b) Activation Unit – Step, Bipolar Step, Sigmoid, TanH, ReLU and Leaky ReLU functions
c) Comparator unit for Error calculation

A2. Develop the above perceptron in your own code (don't use the perceptron model available from package). Use the initial weights as provided below.

W0 = 10, W1 = 0.2, w2 = -0.75, learning rate (α) = 0.05

Write a function for Activation function. Develop & Use the code for Step activation function to learn the weights of the network to implement above provided AND gate logic. The activation function is demonstrated below.

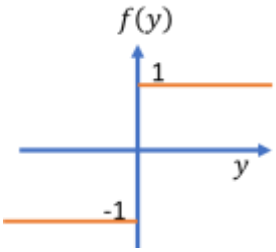


Identify the number of epochs needed for the weights to converge in the learning process. Make a plot of the epochs against the error values calculated (after each epoch, calculate the sum-square-error against all training samples).

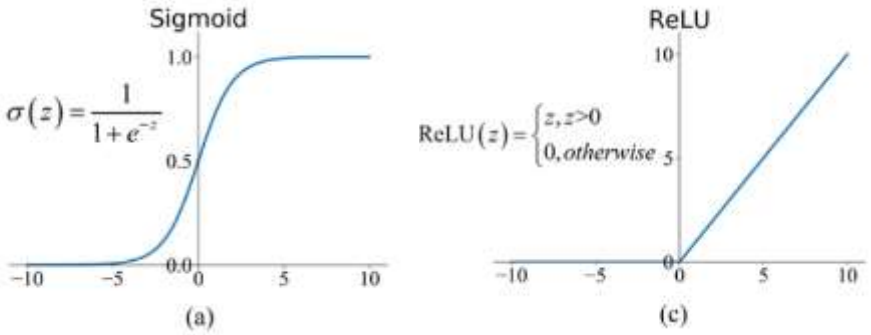
(Note: Learning is said to be converged if the error is less than or equal to 0.002. Stop the learning after 1000 iterations if the convergence error condition is not met.)

A3. Repeat the above A1 experiment with following activation functions (write your own code for activation functions). Compare the iterations taken to converge against each of the activation functions. Keep the learning rate same as A1.

- Bi-Polar Step function
- Sigmoid function
- ReLU function



$$f(y) = \begin{cases} 1 & if\ y > 0 \\ 0 & if\ y = 0 \\ -1 & if\ y < 0 \end{cases}$$



A4. Repeat exercise A1 with varying the learning rate, keeping the initial weights same. Take learning rate = {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1}. Make a plot of the number of iterations taken for learning to converge against the learning rates.

A5. Repeat the above exercises, A1 to A3, for XOR gate logic.

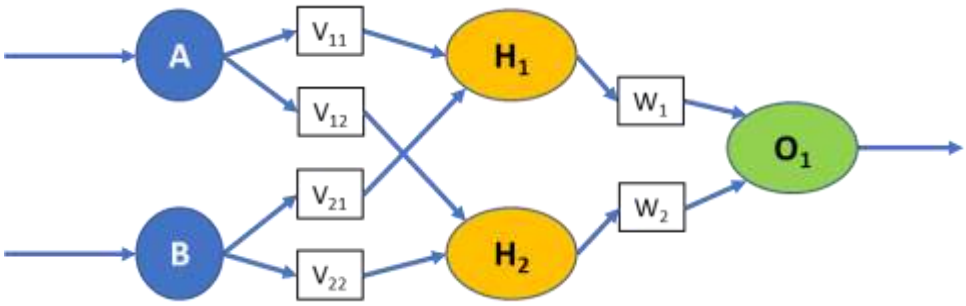
A6. Use customer data provided below. Build a perceptron & learn to classify the transactions as high or low value as provided in the below table. Use sigmoid as the activation function. Initialize the weights & learning rate with your choice.

Customer	Candies (#)	Mangoes (Kg)	Milk Packets (#)	Payment (Rs)	High Value Tx?
C_1	20	6	2	386	Yes
C_2	16	3	6	289	Yes
C_3	27	6	2	393	Yes
C_4	19	1	2	110	No
C_5	24	4	2	280	Yes
C_6	22	1	5	167	No
C_7	15	4	2	271	Yes
C_8	18	4	2	274	Yes
C_9	21	1	4	148	No
C_10	16	2	4	198	No

A7. Compare the results obtained from above perceptron learning to the ones obtained with matrix pseudo-inverse.

A8. Develop the below Neural Network. Use learning rate (α) = 0.05 with a Sigmoid activation function. Learn the weights of the network using back-propagation algorithm to implement above provided AND gate logic.

(Note: Learning is said to be converged if the error is less than or equal to 0.002. Stop the learning after 1000 iterations if the convergence error condition is not met. Logic for back-propagation is provided below.)



A9. Repeat the above A1 experiment for XOR Gate logic. Keep the learning rate & activation function same as A1.

BACKPROPAGATION(*training_examples*, η, n_{in}, n_{out}, n_{hidden})

Each training example is a pair of the form $\langle \vec{x}, \vec{t} \rangle$, *where* $\vec{x}$ *is the vector of network input values, and* $\vec{t}$ *is the vector of target network output values.*

η *is the learning rate (e.g., .05).* n_{in} *is the number of network inputs,* n_{hidden} *the number of units in the hidden layer, and* n_{out} *the number of output units.*

The input from unit i into unit j is denoted x_{ji}, *and the weight from unit i to unit j is denoted* w_{ji}.

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers (e.g., between −.05 and .05).
- Until the termination condition is met, Do
  - For each $\langle \vec{x}, \vec{t} \rangle$ in *training_examples*, Do

Propagate the input forward through the network:

1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \qquad \text{(T4.3)}$$

3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k \qquad \text{(T4.4)}$$

4. Update each network weight w_{ji}

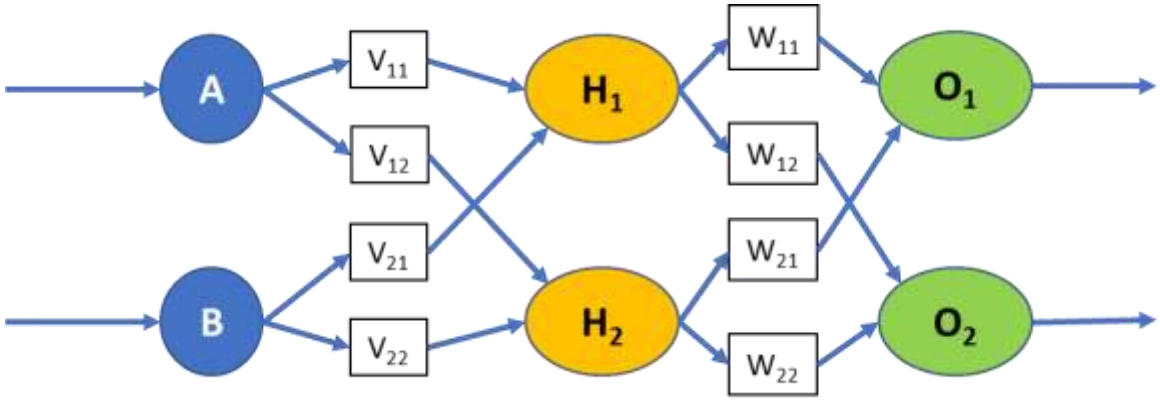
$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = \eta\, \delta_j\, x_{ji} \qquad \text{(T4.5)}$$

(Content obtained from Tom Mitchell book.)

A10. Repeat exercise A1 & A2 with 2 output nodes (as shown below). A zero output of logic gate maps to [O_1 O_2] = [1 0] from output layer while a one output from logic gate maps to [0 1].



A11. Learn using a MLP network from Sci-Kit manual available at https://scikit-learn.org/stable/modules/neural_networks_supervised.html. Repeat the AND Gate and XOR Gate exercises using MLPClassifier() function.

A12. Use the MLPClassifier() function on your project dataset.

Optional Section:

O1. Vary the learning rate for Sigmoid and ReLU activations. Observe the number of iterations taken to converge. Make a plot of iterations to converge against the learning rate for different activations. Study and interpret the graph.

O2. Try the other activation functions and repeat exercise O1.

O3. Vary the learning rate & Activation functions in A7 & A8 and test for their convergence.

Report Assignment:

Please update your last week's report in IEEE format. Expand the methodology and results sections with outcomes of this experiments & results obtained. Please discuss your observations, inferences in results & discussion section. Please conclude the report appropriately with these experiments. Consider following points for observation analysis & inferences. Follow the following instructions:

1. Please tabulate the results (don't take screenshots and add to report)
2. Each table and figure should be captioned and cited in text
3. Refer to the "WritingPaper.pptx" file for more instructions and guidelines