**Batch: C-2 Roll No.:16010122323**

**Experiment No. 06**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title: To implement XOR LOGIC using perceptron network.** . |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Objective:** Implement multilayer perceptron for XOR function with binary inputs and target.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

CO2: Analyze various neural network architectures **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

* .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
**Pre Lab/ Prior Concepts:**

**Perceptron Model**

A Perceptron is a type of linear classifier, which means it can only solve problems where the data points are linearly separable. It takes input features, applies weights, and outputs a decision based on the sign of the sum of the weighted inputs.

**Linear Separability**

A dataset is linearly separable if there exists a straight line (or hyperplane in higher dimensions) that can perfectly separate the two classes. Perceptron’s can only solve problems that are linearly separable. The XOR logic is an example of a problem that is not linearly separable, so it requires a more complex model, like a Multi-Layer Perceptron (MLP), to be solved.

**Design of Classifier using Multi-layer Perceptron model for XOR logic of 2 inputs**

To design a classifier that can solve the **XOR logic** problem using a **Multi-layer Perceptron (MLP)**, we must consider a model that can handle non-linear decision boundaries. The XOR problem is not linearly separable, so a simple single-layer Perceptron is insufficient. Instead, we will use an MLP with at least one hidden layer.

**Structure of the MLP:**

1. **Input Layer**: Two input neurons, corresponding to the two inputs of the XOR problem.

2. **Hidden Layer**: A small hidden layer with a few neurons (typically 2 or more) to allow the network to capture the non-linear patterns in the data. We use an activation function like **sigmoid** or **ReLU** to introduce non-linearity.

3. **Output Layer**: One output neuron, corresponding to the binary result of the XOR operation. The output should be either 0 or 1, so we use a **sigmoid activation** in the output layer.

**Truth table for XOR logic**

The XOR (exclusive OR) logic gate outputs True when the inputs are different, i.e., either one input is True but not both.

|  |  |  |
| --- | --- | --- |
| **Input A** | **Input B** | **Output** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

This problem cannot be solved using a single-layer Perceptron, as it is not linearly separable. Hence, we use a Multi-layer Perceptron (MLP).

**Code:**import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

print("Script is running...")

# XOR inputs (features) and targets (labels)

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

y = np.array([[0], [1], [1], [0]])

# Build the MLP model

model = Sequential()

# Input layer with 2 neurons, hidden layer with 2 neurons (ReLU activation)

model.add(Dense(2, input\_dim=2, activation="relu"))

# Output layer with 1 neuron (Sigmoid activation for binary output)

model.add(Dense(1, activation="sigmoid"))

# Compile the model with binary crossentropy loss and Adam optimizer

model.compile(loss="binary\_crossentropy", optimizer="adam", metrics=["accuracy"])

# Train the model (verbose=0 to suppress output)

model.fit(X, y, epochs=1000, verbose=0)

# Evaluate the model accuracy on the XOR inputs

\_, accuracy = model.evaluate(X, y)

print(f"Model Accuracy: {accuracy \* 100:.2f}%")

# Predict the outputs for the XOR inputs

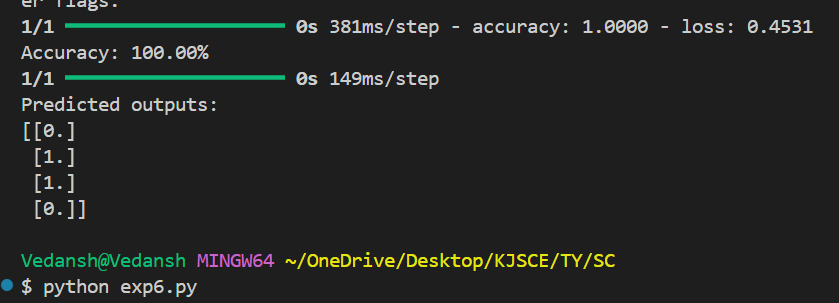
predictions = model.predict(X)

predictions = np.round(predictions)  # Round predictions for binary classification

# Print the predicted outputs

print(f"Predicted Outputs:\n{predictions}")

**Output:**



**Conclusion:** Thus, we have successfully implemented classifier using MLP for linearly non-separable XOR functions of 2 inputs

**Post Lab Descriptive Questions :**

1. **Why is XOR-logic function being Linearly non separable?**

* To be linearly separable, it should be possible to draw a straight line (or hyperplane) that separates the two classes of output (0 and 1).
* In the case of XOR, no single straight line can separate the points representing output 0 from the points representing output 1. You would need at least two lines or a non-linear decision boundary to separate the classes.

1. **Design a classifier using MLP perceptron model for implementing NOT XOR logic**

import numpy as np

from keras.models import Sequential

from keras.layers import Dense

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

y = np.array([[1], [0], [0], [1]])

model = Sequential()

model.add(Dense(2, input\_dim=2, activation='sigmoid'))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(X, y, epochs=10000, verbose=0)

\_, accuracy = model.evaluate(X, y)

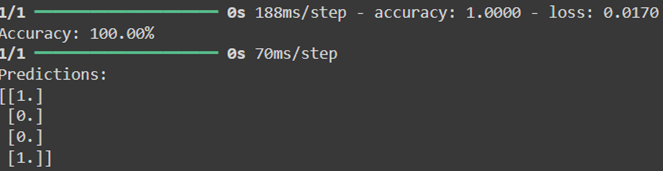
print(f'Accuracy: {accuracy \* 100:.2f}%')

predictions = model.predict(X)

print("Predictions:")

print(np.round(predictions))

.



.

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**