Experiment 02 ~ Implement Multilayer Perceptron algorithm to simulate XOR gate. Pratham Nagar - 64

npImplementation

```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def sigmoid_derivative(x):
    return x * (1 - x)

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

np.random.seed(7)
input_layer_neurons = X.shape[1]
hidden_neurons = 2
output_neurons = 1

wh = np.random.uniform(size=(input_layer_neurons, hidden_neurons))
bh = np.random.uniform(size=(1, hidden_neurons))
wo = np.random.uniform(size=(hidden_neurons, output_neurons))
bo = np.random.uniform(size=(1, output_neurons))

epochs = 12000
lr = 0.15
for _ in range(epochs):
    hidden_input = np.dot(X, wh) + bh
    hidden_output = sigmoid(hidden_input)
    final_input = np.dot(hidden_output, wo) + bo
    final_output = sigmoid(final_input)
    error = y - final_output
    d_output = error * sigmoid_derivative(final_output)
    error_hidden = d_output.dot(wo.T)
    d_hidden = error_hidden * sigmoid_derivative(hidden_output)
    wo += hidden_output.T.dot(d_output) * lr
    bo += np.sum(d_output, axis=0, keepdims=True) * lr
    wh += X.T.dot(d_hidden) * lr
    bh += np.sum(d_hidden, axis=0, keepdims=True) * lr

print("Final Output after training:")
print(final_output.round(3))
```

```
Final Output after training:
[[0.037]
 [0.967]
 [0.967]
 [0.035]]
```

tfImplementation

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

X = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([[0], [1], [1], [2]])

model = Sequential()
model.add(Dense(2, input_dim=2, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mae'])
model.fit(X, y, epochs=5000, verbose=1)

print("Predictions:")
predictions = model.predict(X)
for i, x in enumerate(X):
    print(f"Input: {x} => Output: {predictions[i][0]:.4f}")
```

Show hidden output