BE(E&C)	Data Science & Visualization Lab	
Experiment No.:10	**Implementation of OR/NOR gate using feed forward Neural Network.**	**Page:** /6

- **Aim:** Implementation of OR/NOR gate using feed forward Neural Network.
- **Software Used:** Python 3.12, Jupyter Notebook
- **Learning Objective:**

By the end of this student should be able to:

1. Understand the principles of neural networks and their structure.
2. Learn how to represent binary logic gates using neural networks.
3. Implement a Feed Forward Neural Network to model OR and NOR gates.
4. Train the model using binary input data and evaluate its performance.
5. Visualize the training process and results.

- **Learning Outcomes:**

After performing the experiment students will be able to-

1. Ability to build and train a neural network using TensorFlow/Keras.
2. Knowledge of how neural networks can be applied to logic functions.
3. Skills in evaluating neural network performance and visualizing results.

- **Theory:**

- **Neural Networks**:
  - A neural network is a computational model inspired by the way biological neural networks in the human brain work. It consists of interconnected nodes (neurons) organized in layers: input layer, hidden layers, and output layer.
  - Each connection has a weight, and neurons apply an activation function to the weighted sum of their inputs.

- **Feed Forward Neural Network**:
  - A type of neural network where connections between nodes do not form cycles. Information flows in one direction—from input to output.
  - Activation functions such as Sigmoid or ReLU are commonly used to introduce non-linearity.

BE(E&C)	Data Science & Visualization Lab	
Experiment No.:10	**Implementation of OR/NOR gate using feed forward Neural Network.**	**Page:** /6

- **Logic Gates**:

  - **OR Gate**: Outputs true (1) if at least one input is true (1).
  - **NOR Gate**: Outputs true (1) only if both inputs are false (0). This is the negation of the OR gate.

- **Training Neural Networks**:

  - Neural networks are trained using a dataset consisting of input-output pairs. The training process involves adjusting the weights through backpropagation based on the error between predicted and actual outputs.

- **Activation Functions**:

  - **Sigmoid**: f(x)=11+e−xf(x) = \frac{1}{1 + e^{-x}}f(x)=1+e−x1 maps input values to a range between 0 and 1, making it suitable for binary classification.
  - **Binary Cross-Entropy Loss**: Used as a loss function for binary classification problems.

Code:

```
import numpy as np

import matplotlib.pyplot as plt

from tensorflow import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense


# Prepare the dataset for OR gate

X_or = np.array([[0, 0],

        [0, 1],

        [1, 0],

        [1, 1]])

y_or = np.array([[0], [1], [1], [1]])  # OR gate outputs


# Prepare the dataset for NOR gate
```

```
X_nor = np.array([[0, 0],
                  [0, 1],
                  [1, 0],
                  [1, 1]])
y_nor = np.array([[1], [0], [0], [0]])  # NOR gate outputs

# Function to create and train the model
def create_and_train_model(X, y):
    # Create the model
    model = Sequential()
    model.add(Dense(2, input_dim=2, activation='sigmoid'))  # Hidden layer
    model.add(Dense(1, activation='sigmoid'))  # Output layer

    # Compile the model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

    # Train the model
    model.fit(X, y, epochs=5000, verbose=0)

    return model

# Train the OR gate model
or_model = create_and_train_model(X_or, y_or)
print("OR Gate Predictions:")
print(or_model.predict(X_or))
```

```
# Train the NOR gate model
nor_model = create_and_train_model(X_nor, y_nor)
print("\nNOR Gate Predictions:")
print(nor_model.predict(X_nor))


# Visualize predictions
def plot_predictions(model, X, title):
    predictions = model.predict(X)
    plt.figure()
    plt.scatter(X[:, 0], X[:, 1], c=predictions.flatten(), cmap='coolwarm', s=100)
    plt.title(title)
    plt.xlabel('Input 1')
    plt.ylabel('Input 2')
    plt.colorbar(label='Output')
    plt.xlim(-0.5, 1.5)
    plt.ylim(-0.5, 1.5)
    plt.axhline(0.5, color='grey', lw=0.5, ls='--')
    plt.axvline(0.5, color='grey', lw=0.5, ls='--')
    plt.show()


# Plotting the predictions for both gates
plot_predictions(or_model, X_or, "OR Gate Predictions")
plot_predictions(nor_model, X_nor, "NOR Gate Predictions")
```

BE(E&C)	Data Science & Visualization Lab	
Experiment No.:10	**Implementation of OR/NOR gate using feed forward Neural Network.**	**Page:** /6

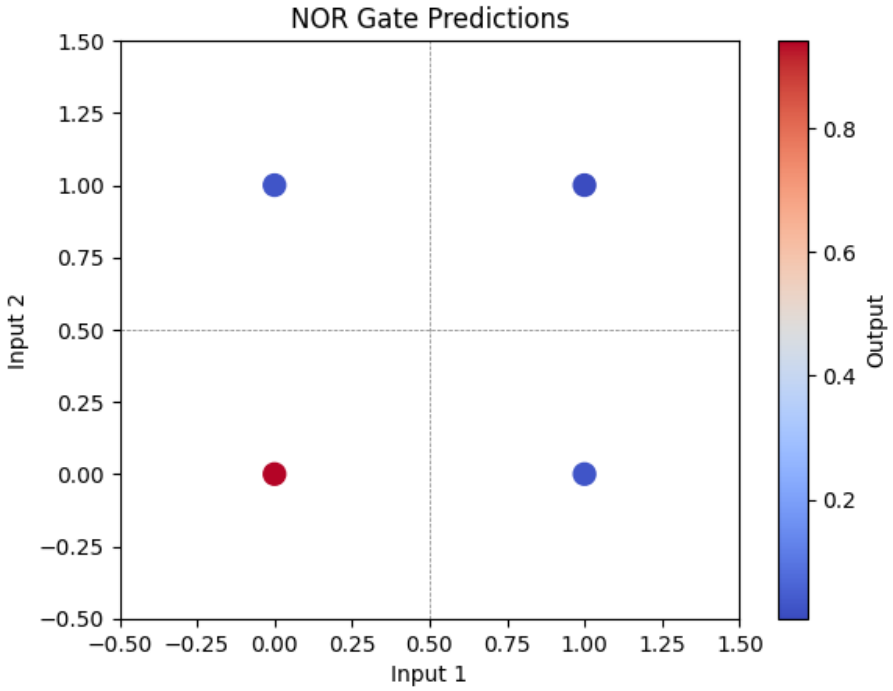
❖ Output

The output will include:

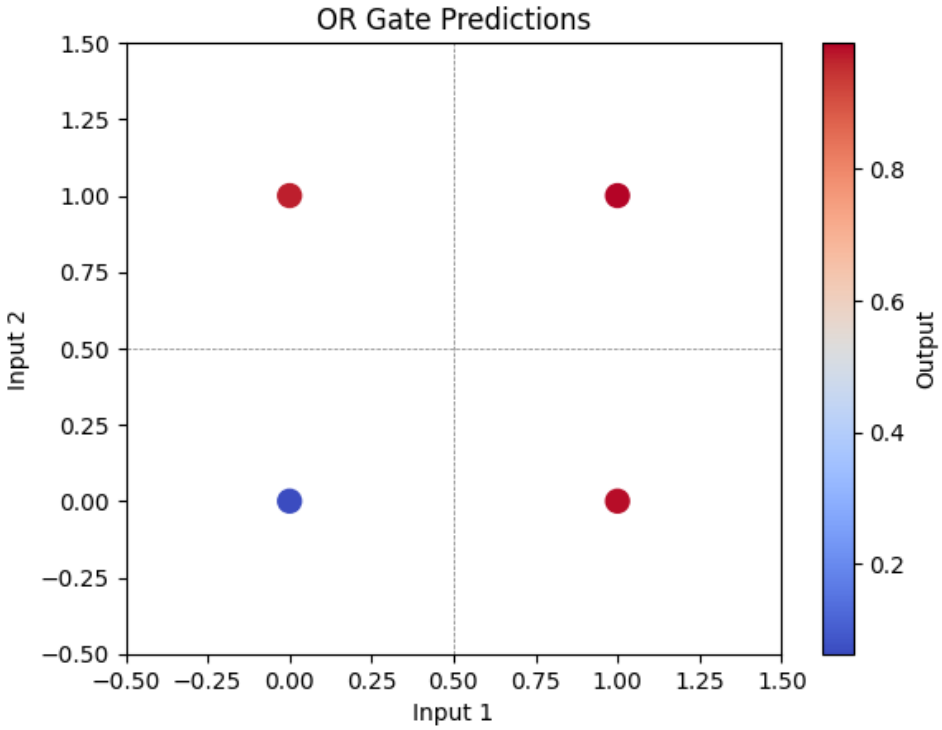
- Predictions of the OR gate for all input combinations.
- Predictions of the NOR gate for all input combinations.
- Visual representations of the outputs for both gates.

```
OR Gate Predictions:
 [[0.06310298]
 [0.9667483 ]
 [0.98086256]
 [0.99098796]]

NOR Gate Predictions:
 [[0.94112545]
 [0.02576258]
 [0.02587234]
 [0.00746044]]
```



BE(E&C)	Data Science & Visualization Lab	
Experiment No.:10	**Implementation of OR/NOR gate using feed forward Neural Network.**	**Page:** /6



❖ **Conclusion:**

_____

_____

_____

_____

❖ **Questions:**

1. Explain how a Feed Forward Neural Network is structured and its role in classification tasks.
2. What are the differences between the OR and NOR logic gates in terms of their truth tables and output behaviour?
3. How does the choice of activation function (e.g., sigmoid) influence the performance of a neural network?
4. What are common loss functions used in binary classification tasks, and why is binary cross-entropy preferred in this implementation?
5. Discuss potential challenges when training neural networks, such as overfitting and underfitting. How can these issues be mitigated?