Batch: B2 **Roll No.: 16010121194**

Experiment No. : 03

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

Signature of the Staff In-charge with date

Title: Study and implementation of Perceptron learning rule.

Objective: To write a program to implement Logic gates problem using perceptron network training

Expected Outcome of Experiment:

CO2: Analyze various training algorithms of neural network and its architectures

Books/ Journals/ Websites referred:

Pre Lab/ Prior Concepts:

Learning

Types of learning

- **Supervised Learning:**
  - In supervised learning, the algorithm learns from labeled data, which means it is provided with input-output pairs to make predictions or classifications.
  - Common algorithms used in supervised learning include linear regression, decision trees, and neural networks.

- **Unsupervised Learning:**
  - Unsupervised learning involves finding patterns, structures, or relationships within unlabeled data.
  - Clustering and dimensionality reduction techniques, such as k-means clustering and principal component analysis (PCA), are often used in unsupervised learning.

- **Reinforcement Learning:**
  - Reinforcement learning is a type of machine learning where an agent learns to make sequences of decisions by interacting with an environment.
  - The agent receives rewards or penalties based on its actions, and it aims to maximize its cumulative reward over time. Common algorithms include Q-learning and deep reinforcement learning.

Perceptron learning rule.

The perceptron learning rule is a simple algorithm used for binary classification tasks. It was developed by Frank Rosenblatt in the late 1950s and is a foundational concept in the history of artificial neural networks. The perceptron is a single-layer feedforward neural network with binary (0 or 1) output.

Here is an overview of the perceptron learning rule:

- Input Data: The perceptron takes a set of input features, each with an associated weight.
- Activation: It computes a weighted sum of the input features and applies an activation function (usually a step function) to determine the output. If the weighted sum is above a certain threshold, the output is 1; otherwise, it's 0.
- Learning Rule: The perceptron learning rule is used to adjust the weights of the inputs to minimize classification errors. If the perceptron misclassifies a data point, it updates the weights to bring the decision boundary closer to the correct classification.
- Convergence: The perceptron learning rule guarantees convergence if the data is linearly separable, which means a decision boundary can separate the two classes.

Steps of Perceptron learning algorithm/approach for binary classification

The Perceptron learning algorithm is a simple approach for binary classification tasks. It aims to find a decision boundary that separates two classes in the input space. Below are the steps of the Perceptron learning algorithm:

1. Initialization:

- Initialize the weights (w) and bias (b) to small random values or zeros. These weights represent the importance of each feature in the classification.

2. Input Data:

- Provide the training dataset, which consists of input features (x) and their corresponding target labels (y). Each input feature should be associated with a binary label (0 or 1).

3. Activation Function:

- Calculate the activation function (often represented as z) for a given input using the weighted sum of the input features and the bias term.

4. Update Weights and Bias:

- Compare the predicted class (y_pred) with the actual target class (y) for the current input.
- The learning rate is a hyperparameter that controls the size of weight and bias updates and is typically set beforehand.

5. Repeat:

- Iterate through the training dataset, updating the weights and bias for each data point based on the classification error. This process can be done for a fixed number of epochs or until convergence (when no errors are made or error reduction is minimal).

6. Convergence:

- The Perceptron learning algorithm guarantees convergence if the data is linearly separable. If the data is not linearly separable, the algorithm may not converge, so it's important to preprocess or modify the data if necessary.

7. Testing:

- After training, you can use the trained perceptron to make predictions on new, unseen data by applying the same activation function and classifying based on the sign of the result.

8. Evaluation:

- Evaluate the performance of the perceptron using appropriate metrics like accuracy, precision, recall, or F1 score on a separate test dataset.

9. Fine-tuning:

- If the perceptron's performance is not satisfactory, you may need to fine-tune hyperparameters, such as the learning rate, or consider using more complex models for more challenging classification tasks.

These steps outline the basic procedure for training a perceptron for binary classification. It's important to note that the perceptron is limited to linearly separable problems and is considered a foundational concept in the development of more complex neural networks.

Implementation Details:

Implement AND, OR, NOR NAND logic function (bipolar inputs and bipolar targets) using perceptron training algorithm.
(show the output of each epoch)

Code:

```
def EXP3(expected_output):

#giving inputs for all gates
  inputs = [
      [1, 1],
      [1, -1],
      [-1, 1],
      [-1, -1]
  ]

#initializing all weights to be 0
#weight 1, weight 2, bias weight
  w1 = w2 = wb = 0

#running 3 epochs
  for j in range(3):
    print("Epoch", (j + 1), ": ")

  #there will be 1 iteration for each input, thus there will be 4 epochs for each
input
    for i in range (len(inputs)):
```

```
      #calculating net
      net = w1 * inputs[i][0] + w2 * inputs[i][1] + wb * 1

      #applying activation function to get the final output
      if net >= 1:
        actual = 1
      elif 1 > net > -1:
        actual = 0
      else:
        actual = -1

      #calculating error using the formula for perceptron model
      #error = target output - actual output
      e = expected_output[i] - actual

      #difference in weight = input + error
      w1d = inputs[i][0] * e
      w2d = inputs[i][1] * e
      wbd = 1 * e

      #weight will be updated
      #new weight = difference in weight + old weight
      w1 = w1d + w1
      w2 = w2d + w2
      wb = wbd + wb

      print("Weight 1\t\t", "Weight 2\t\t", "Bias Weight\t\t", "Error")
      print("  ", w1, "\t\t\t\t", w2, "\t\t\t\t", wb, "\t\t\t\t", e)
      print()
    print("\n")

#calling the function for AND gate
expected_output_and = [1, -1, -1, -1]
#calling the function for OR gate
expected_output_or = [1, 1, 1, -1]
#calling the function for NAND gate
expected_output_nand = [-1, 1, 1, 1]
#calling the function for NOR gate
expected_output_nor = [-1, -1, -1, 1]

#printing output
```

```
print("\n_____AND_____\n\n")

EXP3(expected_output_and)

print("\n_____OR_____\n\n")

EXP3(expected_output_or)

print("\n_____NAND_____\n\n")

EXP3(expected_output_nand)

print("\n_____NOR_____\n\n")

EXP3(expected_output_nor)
```

Output:

```
                                    AND


Epoch 1 :
Weight 1        Weight 2        Bias Weight         Error
   1               1               1               1

Weight 1        Weight 2        Bias Weight         Error
   -1              3               -1                  -2

Weight 1        Weight 2        Bias Weight         Error
   1               1               -3                  -2

Weight 1        Weight 2        Bias Weight         Error
   1               1               -3                  0

Epoch 2 :
Weight 1        Weight 2        Bias Weight         Error
   3               3               -1                  2

Weight 1        Weight 2        Bias Weight         Error
   3               3               -1                  0

Weight 1        Weight 2        Bias Weight         Error
   3               3               -1                  0

Weight 1        Weight 2        Bias Weight         Error
   3               3               -1                  0
```

```
_____________________________OR_____________________________


Epoch 1 :
Weight 1        Weight 2        Bias Weight         Error
   1               1               1               1

Weight 1        Weight 2        Bias Weight         Error
   1               1               1               0

Weight 1        Weight 2        Bias Weight         Error
   1               1               1               0

Weight 1        Weight 2        Bias Weight         Error
   1               1               1               0



Epoch 2 :
Weight 1        Weight 2        Bias Weight         Error
   1               1               1               0

Weight 1        Weight 2        Bias Weight         Error
   1               1               1               0

Weight 1        Weight 2        Bias Weight         Error
   1               1               1               0

Weight 1        Weight 2        Bias Weight         Error
   1               1               1               0
```

```
                                    NAND


Epoch 1 :
Weight 1        Weight 2        Bias Weight         Error
   -1               -1                -1                  -1

Weight 1        Weight 2        Bias Weight         Error
   1                -3                1                 2

Weight 1        Weight 2        Bias Weight         Error
   -1               -1                3                 2

Weight 1        Weight 2        Bias Weight         Error
-1              -1                 3                 0



Epoch 2 :
Weight 1        Weight 2        Bias Weight         Error
   -3               -3                1                 -2

Weight 1        Weight 2        Bias Weight         Error
   -3               -3                1                 0

Weight 1        Weight 2        Bias Weight         Error
   -3               -3                1                 0

Weight 1        Weight 2        Bias Weight         Error
   -3               -3                1                 0
```

```
                                        NOR

Epoch 1 :
Weight 1        Weight 2        Bias Weight         Error
   -1               -1                -1                    -1

Weight 1        Weight 2        Bias Weight         Error
-1              -1                   -1                  0

Weight 1        Weight 2        Bias Weight         Error
   -1               -1                -1                    0

Weight 1        Weight 2        Bias Weight         Error
   -1               -1                -1                    0


Epoch 2 :
Weight 1        Weight 2        Bias Weight         Error
   -1               -1                -1                    0

Weight 1        Weight 2        Bias Weight         Error
   -1               -1                -1                    0

Weight 1        Weight 2        Bias Weight         Error
   -1               -1                -1                    0

Weight 1        Weight 2        Bias Weight         Error
-1              -1                   -1                  0
```

Conclusion: Thus, we have successfully implemented perceptron training algorithm for AND function.

Post Lab Descriptive Questions :

1. **How is linear separability implemented using perceptron networking training**

Ans) Linear separability is a fundamental concept in the context of perceptron training. It refers to the property of data points being divided into different classes by a straight line (in 2D), a hyperplane (in higher dimensions), or a linear decision boundary in general.

Initialization: Initialize the weights and bias (threshold) values to small random values or zeros.

Training Data: Provide labeled training examples, where each example consists of input features and their corresponding target outputs.

Activation Function: The perceptron uses a step function as its activation function. The output of the perceptron is calculated as the sum of weighted inputs plus the bias, and then it's passed through the step function.
Output = Step(Weighted Sum + Bias)

Updating Weights: For each training example, compute the weighted sum of inputs, apply the activation function, and compare the result with the target output.

If the predicted output matches the target, no changes are made.

If the predicted output is higher than the target for a positive example (or lower for a negative example), the weights are decreased.

If the predicted output is lower than the target for a positive example (or higher for a negative example), the weights are increased.

The update rule for weights and bias is:
New Weight = Old Weight + Learning Rate * (Target - Predicted) * Input
New Bias = Old Bias + Learning Rate * (Target - Predicted)

Convergence: Repeat the training process for all examples in the training set. If the data is linearly separable, the perceptron will eventually converge, meaning that the weights and bias will be adjusted such that a proper linear decision boundary is found.

Stopping Criterion: You can set a stopping criterion based on the number of epochs (passes through the training data) or when the error becomes small enough.

2. Mention the application of perceptron network.

Ans)
Pattern Recognition: Perceptrons can be used for simple pattern recognition tasks, like distinguishing between classes based on linearly separable features. They were historically used for character recognition and image classification tasks.

Binary Classification: Perceptrons are adept at binary classification problems, such as spam detection in emails, fraud detection in financial transactions, and medical diagnosis.

Feature Extraction: In some cases, perceptrons can be used for feature extraction. They can help identify relevant features from input data, which can then be used as inputs to more advanced models.

Logical Operations: Perceptrons can implement basic logical operations like AND, OR, NAND, and NOR, as discussed earlier. They form the building blocks of more complex neural network architectures used in machine learning.

.

Date: _____ **Signature of faculty in-charge**