Experiment No. 02

Semester	B.E. Semester VIII – Computer Engineering
Subject	Deep Learning Lab
Subject Professor In-charge	Dr. Kavita Shirsat
Academic Year	2024-25

Student Name	Deep Salunkhe
Roll Number	21102A0014

Title: Implementation of Delta Rule for updating weights and bias

---

Explanation:

- **Delta Rule to Update Weights and Bias:** The **Delta Rule** is a supervised learning method used in neural networks to adjust weights and bias to minimize the difference between predicted and actual outputs. It is primarily used in single layer perceptrons for tasks involving linearly separable data.

- **Concept:** The Delta Rule relies on the principle of gradient descent, where the goal is to iteratively reduce the error by adjusting the weights and bias in the direction that minimizes the error. This adjustment process ensures the network learns from its mistakes and improves its predictions over time.

- **How It Works:**

  1. **Error Measurement:** The difference between the desired output (target) and the actual output is calculated. This difference is referred to as the error.

  2. **Weight Adjustment:** Each weight in the network is updated based on its contribution to the error. If a weight contributes significantly to the error, it is adjusted more substantially. The adjustment magnitude is influenced by a learning rate, which controls how quickly the model adapts.

3. **Bias Adjustment:** The bias is also updated to account for systematic errors in the predictions. This adjustment allows the model to fine-tune its predictions more effectively.

4. **Iterative Updates:** The process is repeated for all training examples. Over time, the weights and bias converge to values that minimize the error for the given dataset.

- **Applications:**
  - Used in single-layer perceptrons for binary classification tasks.
  - Forms the foundation for more complex algorithms like backpropagation in multi-layer networks.

- **Strengths**
  - Simple to implement and computationally efficient for small datasets.
  - Guarantees convergence to the optimal weights for problems that are linearly separable.

- **Limitations**
  - Restricted to single-layer networks and linearly separable problems.
  - Performance is sensitive to the choice of learning rate.
  - Ineffective for non-linear problems, requiring more advanced techniques like backpropagation.

- **Importance:** The Delta Rule is a cornerstone of machine learning and neural networks, laying the groundwork for modern algorithms. It provides insights into how models learn and adapt by systematically reducing prediction errors.

Implementation:

```java
import java.util.Scanner;

public class DR {

    public static double activationFunction(double x) {
        return 1 / (1 + Math.exp(-x));
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // User Inputs
        System.out.println("Enter Number of tuples: ");
        int n = sc.nextInt();

        System.out.println("Enter the values of x:");
        double[] x = new double[n];
        for (int i = 0; i < n; i++) {
            x[i] = sc.nextDouble();
        }

        System.out.println("Enter the value of y:");
        double y = sc.nextDouble();

        System.out.println("Enter the values of weights:");
        double[] weights = new double[n];
        for (int i = 0; i < n; i++) {
            weights[i] = sc.nextDouble();
        }

        System.out.println("Enter the learning rate: ");
        double c = sc.nextDouble();

        System.out.println("Enter the Bias: ");
        double b = sc.nextDouble();

        System.out.println("Enter the number of iterations: ");
        int iter = sc.nextInt();

        sc.close();

        // Training Loop
        for (int it = 0; it < iter; it++) {
            double z = 0;
```

```
            for (int i = 0; i < n; i++) {
                z += weights[i] * x[i];
            }

            double yPred = activationFunction(z + b);

            for (int i = 0; i < n; i++) {
                weights[i] += c * (y - yPred) * x[i];
            }

            b += c * (y - yPred);
        }

        // Output Results
        double zFinal = 0;
        for (int i = 0; i < n; i++) {
            zFinal += weights[i] * x[i];
        }
        double yPredFinal = activationFunction(zFinal + b);

        System.out.printf("Original Value: %.2f, Predicted Value: %.2f\n", y,
yPredFinal);
    }
}
```

Output:

```
PS E:\GIt\Sem-8\DL\Lab2> java DR
Enter Number of tuples:
2
Enter the values of x:
1 2
Enter the value of y:
0.8
Enter the values of weights:
0.5 -0.5
Enter the learning rate:
0.1
Enter the Bias:
0.1
Enter the number of iterations:
4
Original Value: 0.80, Predicted Value: 0.59
PS E:\GIt\Sem-8\DL\Lab2> java DR
Enter Number of tuples:
2
Enter the values of x:
1 2
Enter the value of y:
0.8
Enter the values of weights:
0.5 -0.5
Enter the learning rate:
0.1
Enter the Bias:
0.1
Enter the number of iterations:
6
Original Value: 0.80, Predicted Value: 0.65
```