|  |  |
| --- | --- |
|  | **DEPARTMENT OF COMPUTER ENGINEERING** |

Experiment No. 03

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

|  |  |
| --- | --- |
| Student Name | Deep Salunkhe |
| Roll Number | 21102A0014 |

**Title:** Implementation of different Activation functions.

**Explanation:**

**1. Sigmoid (Logistic Function)**

* Converts inputs into values between 0 and 1, making it suitable for binary classification problems.
* It compresses large input ranges but suffers from the **vanishing gradient problem**, making it less effective for deep networks.

**2. ReLU (Rectified Linear Unit)**

* Outputs the input if it's positive; otherwise, it outputs 0.
* Commonly used in hidden layers of deep neural networks due to its simplicity and efficiency.
* Avoids the vanishing gradient issue but can face the **dying ReLU problem**, where neurons become inactive.

**3. Leaky ReLU**

* A variation of ReLU that introduces a small slope for negative inputs, preventing neurons from becoming inactive.
* Solves the dying ReLU problem and ensures small gradients for all inputs.
* Requires tuning of the slope parameter for optimal performance.

**4. Tanh (Hyperbolic Tangent)**

* Similar to the sigmoid function but outputs values between -1 and 1, which are zero-centered, aiding optimization.
* Works well with normalized data but suffers from the vanishing gradient problem for extreme inputs.

**5. Softmax**

* Converts input scores into a probability distribution over multiple classes, with the sum of probabilities equal to 1.
* Commonly used in the output layer of multi-class classification models.
* Sensitive to large input values, so normalization is often needed.

**Implementation:**

**import** **java.util.\***;

**public** **class** AF {

**public** **static** **double** sigmoid(**double** x) {

**return** 1 **/** (1 **+** Math.exp(**-**x));

    }

**public** **static** **double** relu(**double** x) {

**return** Math.max(0, x);

    }

**public** **static** **double** tanh(**double** x) {

**return** (Math.exp(x) **-** Math.exp(**-**x)) **/** (Math.exp(x) **+** Math.exp(**-**x));

    }

**public** **static** **double** linear(**double** x) {

**return** x;

    }

**public** **static** **double** leakyRelu(**double** x) {

**return** x **<=** 0 **?** 0.01 **\*** x **:** x;

    }

**public** **static** **double**[] softmax(**double**[] x) {

**double**[] fx **=** **new** **double**[x.length];

**double** sumExp **=** 0;

**for** (**double** val **:** x) {

            sumExp **+=** Math.exp(val);

        }

**for** (**int** i **=** 0; i **<** x.length; i**++**) {

            fx[i] **=** Math.exp(x[i]) **/** sumExp;

        }

**return** fx;

    }

**public** **static** **void** main(**String**[] args) {

**Scanner** sc **=** **new** Scanner(System.in);

**int** n**=**4;

**double**[] x1 **=** {2104, 1600, 2400, 1416};

**double**[] x2 **=** {3, 3, 3, 2};

**double**[] y **=** {400, 330, 369, 232};

**double**[] weights **=** **new** **double**[n];

**Random** rand **=** **new** Random();

**for** (**int** i **=** 0; i **<** n; i**++**) {

            weights[i] **=** **-**1 **+** (1 **-** (**-**1)) **\*** rand.nextDouble(); // Random weights between -1 and 1

        }

        sc.close();

        System.out.println("Weights: " **+** Arrays.toString(weights));

        System.out.println("Features: x1=" **+** Arrays.toString(x1) **+** ", x2=" **+** Arrays.toString(x2));

        System.out.println("Target Values: y=" **+** Arrays.toString(y));

**List**<**java**.**util**.**function**.**Function**<**Double**, **Double**>> activationFunctions **=** Arrays.asList(

            AF**::**sigmoid,

            AF**::**relu,

            AF**::**tanh,

            AF**::**linear,

            AF**::**leakyRelu

        );

**String**[] functionNames **=** {"Sigmoid", "ReLU", "Tanh", "Linear", "Leaky ReLU"};

**for** (**int** idx **=** 0; idx **<** activationFunctions.size(); idx**++**) {

            System.out.println("Activation Function: " **+** functionNames[idx]);

**double** mse **=** 0;

**for** (**int** i **=** 0; i **<** 4; i**++**) {

**double** z **=** weights[0] **\*** x1[i] **+** weights[1] **\*** x2[i];

**double** yPred **=** activationFunctions.get(idx).apply(z);

                mse **+=** Math.pow(yPred **-** y[i], 2);

            }

            mse **/=** 4;

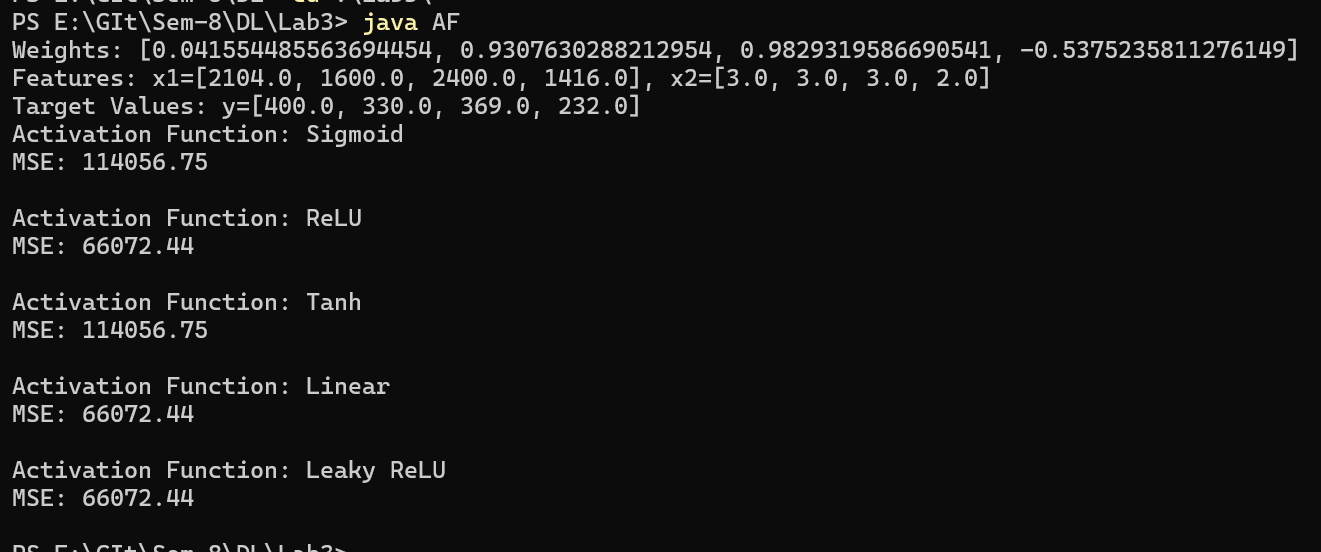
            System.out.printf("MSE: %.2f%n%n", mse);

        }

    }

}

**Output:**

****