Vidyalankar Institute of Technology
Accredited A+ by NAAC

DEPARTMENT OF COMPUTER ENGINEERING

Experiment No. 04

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: Neural Network for Feature Selection and Forward Propagation

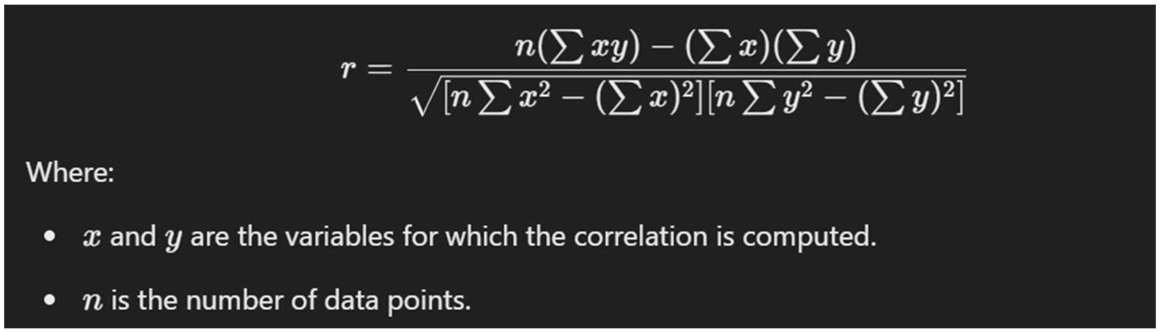
---

Explanation:

n this lab, we explore the foundational concepts of neural networks, focusing on feature selection based on correlation analysis and performing forward propagation with randomly initialized weights.

1. Feature Selection Based on Correlation

Feature selection is a critical preprocessing step in machine learning to improve model efficiency by selecting the most relevant features. Here, we utilize Pearson's correlation coefficient to evaluate the relationship between each feature and the target variable. Pearson correlation measures the linear relationship between two variables, defined as:



$$r = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}}$$

Where:

- x and y are the variables for which the correlation is computed.
- n is the number of data points.

A correlation threshold of 0.5 is chosen in this implementation, meaning features with a correlation magnitude greater than 0.5 with the target variable are selected. These selected features are used as inputs for the neural network, which reduces the dimensionality of the problem and improves computational efficiency.

2. Neural Network Architecture

A neural network consists of layers of interconnected neurons that transform inputs into outputs. In this implementation, the neural network is structured as follows:

- **Input Layer**: The selected features from the dataset are fed into the input layer.
- **Hidden Layers**: The network can have multiple hidden layers, as specified by the user. Each hidden layer contains neurons that compute weighted sums of the inputs, followed by an activation function.
- **Output Layer**: The output of the neural network is the predicted value, calculated by the output neuron(s).

3. Weight Initialization

In neural networks, weights are parameters that control the strength of connections between neurons. Initially, weights are assigned random values to break symmetry and allow the model to learn patterns from the data. In this implementation, weights are initialized using a uniform distribution between -1 and 1.

4. Forward Propagation

Forward propagation is the process through which the neural network makes predictions by passing input data through layers of neurons. Each layer computes the weighted sum of its inputs and applies an activation function (in this case, the sigmoid function) to generate its output. The steps are:

- **Weighted Sum Calculation**: For each neuron in the current layer, the input values are multiplied by their respective weights, and a bias is added.

$$z = \sum(w_i \cdot x_i) + b$$

where w_i represents the weight, x_i is the input, and b is the bias term.

- **Activation Function**: The weighted sum z is passed through the activation function, typically a non-linear function. In this case, the **sigmoid function** is used, which squashes the output between 0 and 1:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

This process is repeated layer by layer, from the input layer to the output layer, generating the final prediction.

5. Sigmoid Activation Function

The sigmoid activation function is commonly used in binary classification problems as it maps the output to a value between 0 and 1. It has the following properties:

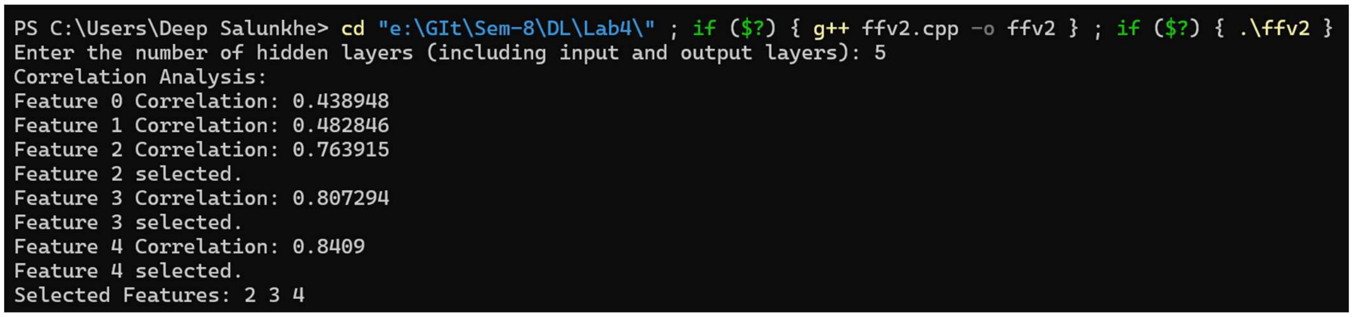
- **Range**: The output is always between 0 and 1.
- **Derivative**: The sigmoid function has a well-defined derivative, making it suitable for backpropagation during training.

Implementation:

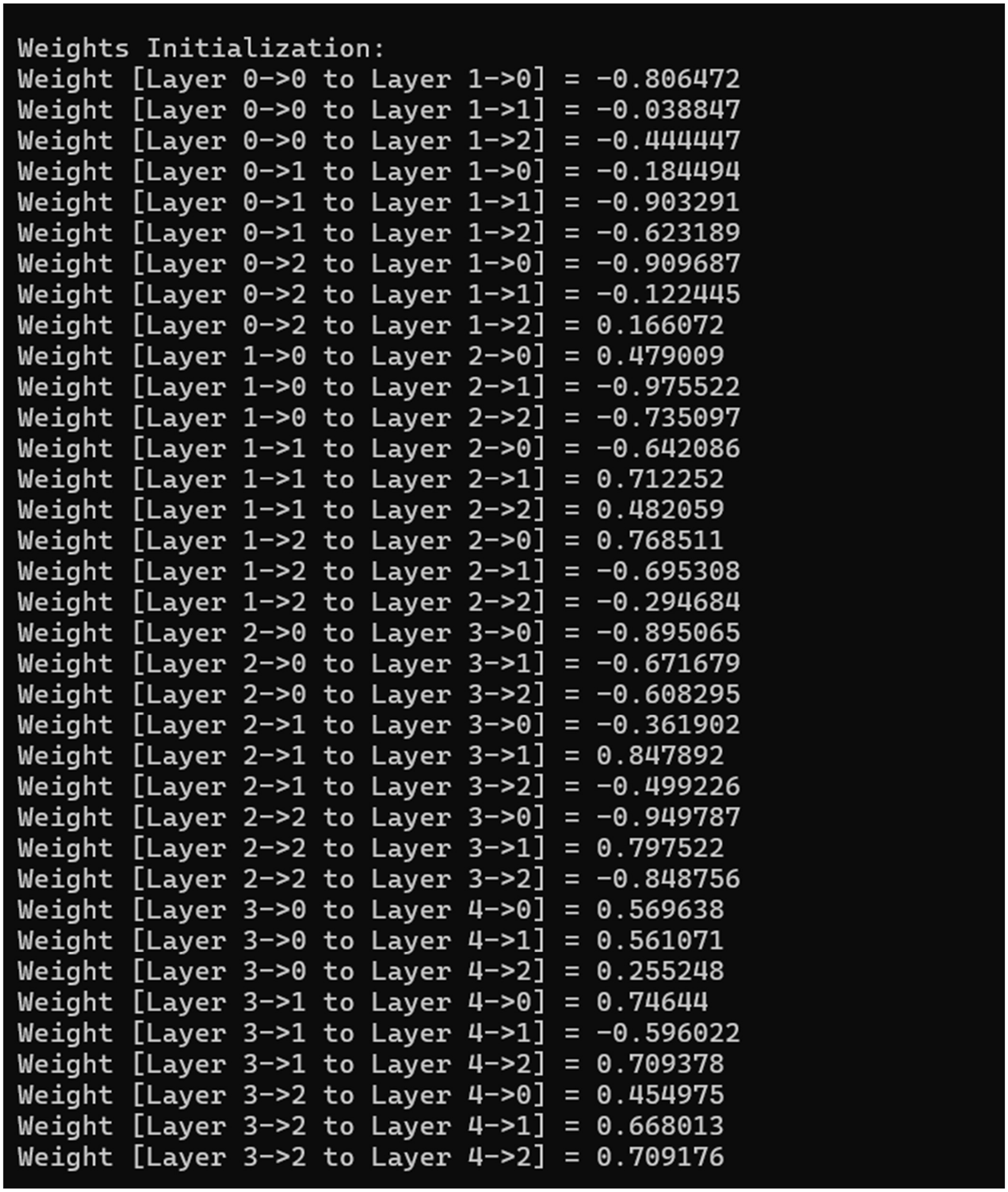
```
#include <iostream>
#include <fstream>
#include <vector>
#include <map>
#include <cmath>
#include <sstream>
#include <random>
#include <algorithm>

using namespace std;

// Function to compute Pearson correlation coefficient
float computeCorrelation(const vector<float>& x, const vector<float>& y) {
    int n = x.size();
    float sum_x = 0, sum_y = 0, sum_xy = 0;
    float sum_x2 = 0, sum_y2 = 0;

    for (int i = 0; i < n; i++) {
        sum_x += x[i];
        sum_y += y[i];
        sum_xy += x[i] * y[i];
        sum_x2 += x[i] * x[i];
        sum_y2 += y[i] * y[i];
    }
    float numerator = (n * sum_xy - sum_x * sum_y);
    float denominator = sqrt((n * sum_x2 - sum_x * sum_x) * (n * sum_y2 - sum_y *
sum_y));

    if (denominator == 0) return 0; // Prevent division by zero
    return numerator / denominator;
}

// Read CSV file and extract data
void readCSV(const string& filename, vector<vector<float>>& features, vector<float>&
target) {
    ifstream file(filename);
    string line;
    getline(file, line); // Skip header line

    while (getline(file, line)) {
        stringstream ss(line);
        vector<float> row;
        string value;

        while (getline(ss, value, ',')) {
            row.push_back(stof(value));
        }

        target.push_back(row.back()); // Last column is 'y'
        row.pop_back(); // Remove 'y' from features
        features.push_back(row);
    }
}

// Generate random weights
void initializeWeights(map<pair<pair<int, int>, pair<int, int>>, float>& weights, int
layerSize, int numLayers) {
    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<float> dist(-1.0, 1.0);

    for (int i = 0; i < numLayers - 1; i++) {
        for (int j = 0; j < layerSize; j++) {
            for (int k = 0; k < layerSize; k++) {
                weights[{{i, j}, {i + 1, k}}] = dist(gen);
            }
        }
    }
}

// Forward propagation
```

```cpp
void forwardPass(vector<vector<float>>& neuron, int layers, map<pair<pair<int, int>,
pair<int, int>>, float>& weights, vector<float>& bias) {
    for (int i = 0; i < layers - 1; i++) {
        for (int j = 0; j < neuron[i+1].size(); j++) {
            float sum = 0;
            for (int k = 0; k < neuron[i].size(); k++) {
                sum += neuron[i][k] * weights[{{i, k}, {i + 1, j}}];
            }
            sum += bias[i + 1];
            neuron[i + 1][j] = 1 / (1 + exp(-sum)); // Sigmoid activation
        }
    }
}

int main() {
    string filename = "data.csv";
    vector<vector<float>> features;
    vector<float> target;

    readCSV(filename, features, target);
    int numFeatures = features[0].size();

    // Prompt user for the number of hidden layers
    int numLayers;
    cout << "Enter the number of hidden layers (including input and output layers): ";
    cin >> numLayers;

    // Compute correlation with y and select features
    vector<int> selectedFeatures;
    cout << "Correlation Analysis: " << endl;
    for (int i = 0; i < numFeatures; i++) {
        vector<float> column;
        for (auto& row : features) column.push_back(row[i]);

        float correlation = computeCorrelation(column, target);
        cout << "Feature " << i << " Correlation: " << correlation << endl;

        if (abs(correlation) > 0.5) {
            selectedFeatures.push_back(i);
            cout << "Feature " << i << " selected." << endl;
        }
    }

    cout << "Selected Features: ";
    for (int idx : selectedFeatures) {
        cout << idx << " ";
    }
    cout << endl;
    int layerSize = selectedFeatures.size();

    // Initialize weights
    map<pair<pair<int, int>, pair<int, int>>, float> weights;
    initializeWeights(weights, layerSize, numLayers);

    // Bias initialization
    vector<float> bias(numLayers, 0.5); // Example bias value

    cout << "\nWeights Initialization: " << endl;
    for (const auto& weight : weights) {
        cout << "Weight [Layer " << weight.first.first.first << "->"
             << weight.first.first.second << " to Layer "
             << weight.first.second.first << "->"
             << weight.first.second.second << "] = "
             << weight.second << endl;
    }

    // Forward propagation for each input
    for (auto& row : features) {
        vector<vector<float>> neuron(numLayers, vector<float>(layerSize, 0));
        for (int i = 0; i < layerSize; i++) {
            neuron[0][i] = row[selectedFeatures[i]]; // Use selected features as input
        }
        forwardPass(neuron, numLayers, weights, bias);

        cout << "\nPrediction for this row: " << neuron.back()[0] << endl; // Output
prediction
    }

    return 0;
}
```

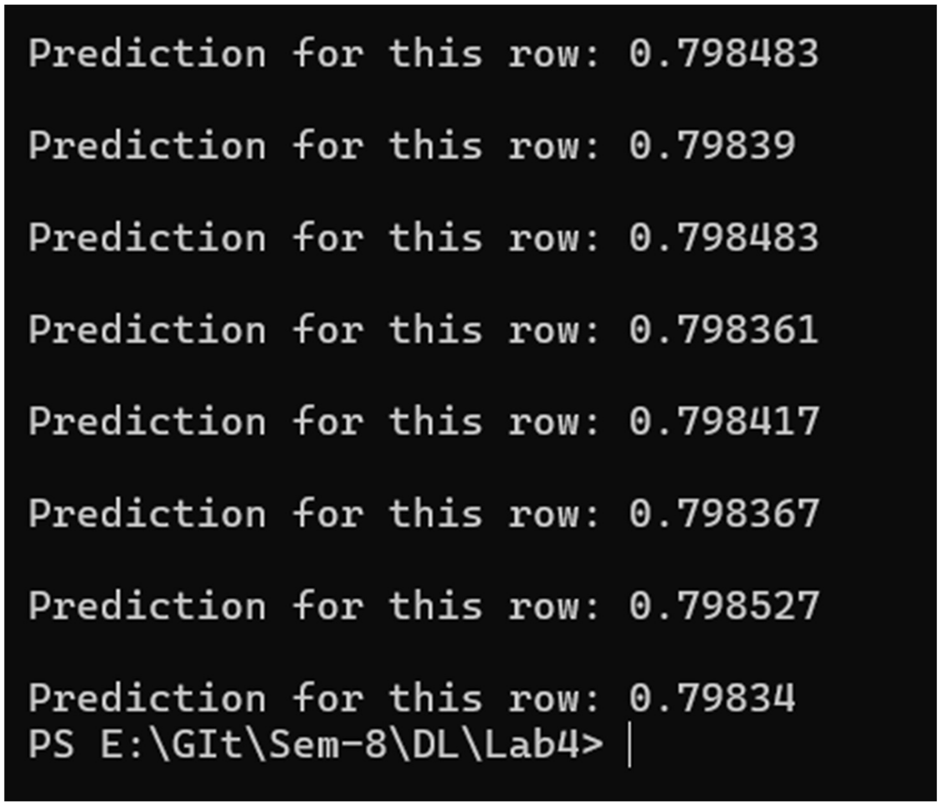
Output:



```
PS C:\Users\Deep Salunkhe> cd "e:\GIt\Sem-8\DL\Lab4\" ; if ($?) { g++ ffv2.cpp -o ffv2 } ; if ($?) { .\ffv2 }
Enter the number of hidden layers (including input and output layers): 5
Correlation Analysis:
Feature 0 Correlation: 0.438948
Feature 1 Correlation: 0.482846
Feature 2 Correlation: 0.763915
Feature 2 selected.
Feature 3 Correlation: 0.807294
Feature 3 selected.
Feature 4 Correlation: 0.8409
Feature 4 selected.
Selected Features: 2 3 4
```



```
Weights Initialization:
Weight [Layer 0->0 to Layer 1->0] = -0.806472
Weight [Layer 0->0 to Layer 1->1] = -0.038847
Weight [Layer 0->0 to Layer 1->2] = -0.444447
Weight [Layer 0->1 to Layer 1->0] = -0.184494
Weight [Layer 0->1 to Layer 1->1] = -0.903291
Weight [Layer 0->1 to Layer 1->2] = -0.623189
Weight [Layer 0->2 to Layer 1->0] = -0.909687
Weight [Layer 0->2 to Layer 1->1] = -0.122445
Weight [Layer 0->2 to Layer 1->2] = 0.166072
Weight [Layer 1->0 to Layer 2->0] = 0.479009
Weight [Layer 1->0 to Layer 2->1] = -0.975522
Weight [Layer 1->0 to Layer 2->2] = -0.735097
Weight [Layer 1->1 to Layer 2->0] = -0.642086
Weight [Layer 1->1 to Layer 2->1] = 0.712252
Weight [Layer 1->1 to Layer 2->2] = 0.482059
Weight [Layer 1->2 to Layer 2->0] = 0.768511
Weight [Layer 1->2 to Layer 2->1] = -0.695308
Weight [Layer 1->2 to Layer 2->2] = -0.294684
Weight [Layer 2->0 to Layer 3->0] = -0.895065
Weight [Layer 2->0 to Layer 3->1] = -0.671679
Weight [Layer 2->0 to Layer 3->2] = -0.608295
Weight [Layer 2->1 to Layer 3->0] = -0.361902
Weight [Layer 2->1 to Layer 3->1] = 0.847892
Weight [Layer 2->1 to Layer 3->2] = -0.499226
Weight [Layer 2->2 to Layer 3->0] = -0.949787
Weight [Layer 2->2 to Layer 3->1] = 0.797522
Weight [Layer 2->2 to Layer 3->2] = -0.848756
Weight [Layer 3->0 to Layer 4->0] = 0.569638
Weight [Layer 3->0 to Layer 4->1] = 0.561071
Weight [Layer 3->0 to Layer 4->2] = 0.255248
Weight [Layer 3->1 to Layer 4->0] = 0.74644
Weight [Layer 3->1 to Layer 4->1] = -0.596022
Weight [Layer 3->1 to Layer 4->2] = 0.709378
Weight [Layer 3->2 to Layer 4->0] = 0.454975
Weight [Layer 3->2 to Layer 4->1] = 0.668013
Weight [Layer 3->2 to Layer 4->2] = 0.709176
```



```
Prediction for this row: 0.798483

Prediction for this row: 0.79839

Prediction for this row: 0.798483

Prediction for this row: 0.798361

Prediction for this row: 0.798417

Prediction for this row: 0.798367

Prediction for this row: 0.798527

Prediction for this row: 0.79834
PS E:\GIt\Sem-8\DL\Lab4> 
```