Vidyalankar Institute of Technology Accredited A+ by NAAC	**DEPARTMENT OF COMPUTER ENGINEERING**

Experiment No. 06

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: Batch Gradient Descent

- **Overview**

This program implements **Batch Gradient Descent (BGD)** for linear regression using a dataset provided in a CSV file. The goal is to find optimal weights (theta values) that minimize the Mean Squared Error (MSE).

- **Key Components of the Code**

- **Function: printThetas(vector<float>& thetas)**
  - Prints the current theta values (weights).
- **Function: computeMSE(vector<float>& y_cap, vector<float>& target)**
  - Computes Mean Squared Error (MSE).
  - Formula:

MSE = (1/N) * Σ (y_actual - y_predicted)^2

where N is the number of data points.

- **Function: gradientDescent(vector<float>& thetas, vector<vector<float> >& data, vector<float>& target, int tc, int tl, float**

lr, int epochs)

- Performs batch gradient descent:

1. **Prediction Step:**

y_cap[i] = theta_0 + theta_1 * x1 + theta_2 * x2 + ... + theta_n * xn

2. **Gradient Update Rule:**

theta_j = theta_j - (lr * (1/N) * Σ (y_actual - y_predicted) * x_j)

- lr = Learning Rate
- N = Number of training samples

- **Function: readCSV(string filename, vector<float>& target)**
  - Reads the CSV file, extracts feature values and target labels.

- **Main Execution (main())**

- Prompts user for CSV file path.
- Reads data and initializes weights (theta values) to zero.
- Accepts **learning rate (lr)** from the user.
- Runs **gradient descent for 10 epochs** and prints updated weights and MSE after each epoch.

- **Expected Behavior**

- If learning rate is **too high**, weights may diverge, leading to inf values.
- If learning rate is **too low**, convergence may be too slow.
- A properly tuned learning rate leads to decreasing MSE.

Implementation:

```
#include <iostream>
#include <vector>
#include <fstream>
```

```cpp
#include <sstream>
#include <cmath>

using namespace std;

void printThetas(vector<float>& thetas) {
    for (float theta : thetas) {
        cout << theta << " ";
    }
    cout << endl;
}

float computeMSE(vector<float>& y_cap, vector<float>& target) {
    float mse = 0.0;
    int n = target.size();
    for (int i = 0; i < n; i++) {
        mse += pow(target[i] - y_cap[i], 2);
    }
    return mse / n;
}

void gradientDescent(vector<float>& thetas, vector<vector<float>>& data, vector<float>&
target, int tc, int tl, float lr, int epochs) {
    for (int epoch = 1; epoch <= epochs; epoch++) {
        vector<float> y_cap(tl);

        for (int i = 0; i < tl; i++) {
            float c_y_cap = thetas[0];
            for (int j = 0; j < tc - 1; j++) {
                c_y_cap += data[i][j] * thetas[j + 1];
            }
            y_cap[i] = c_y_cap;
        }

        for (int i = 0; i < tc; i++) {
            float sum = 0.0;
            for (int j = 0; j < tl; j++) {
                if (i == 0) {
                    sum += target[j] - y_cap[j];
                } else {
                    sum += (target[j] - y_cap[j]) * data[j][i - 1];
                }
            }
            thetas[i] -= (lr * (-sum / tl));
        }

        float mse = computeMSE(y_cap, target);
        cout << "Epoch " << epoch << " MSE: " << mse << endl;
```

```cpp
        cout << "Weights: ";
        printThetas(thetas);
    }
}

vector<vector<float>> readCSV(string filename, vector<float>& target) {
    vector<vector<float>> data;
    ifstream file(filename);
    string line;

    // Read and discard the first line (feature names)
    getline(file, 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());
        row.pop_back();
        data.push_back(row);
    }
    return data;
}

int main() {
    string filename;
    cout << "Enter CSV file path: ";
    cin >> filename;

    vector<float> target;
    vector<vector<float>> data = readCSV(filename, target);
    int tl = data.size();
    int tc = data[0].size() + 1;

    vector<float> thetas(tc, 0.0);
    float lr;
    int epochs = 10;

    cout << "Enter learning rate: ";
    cin >> lr;

    gradientDescent(thetas, data, target, tc, tl, lr, epochs);

    cout << "Final Thetas: ";
    printThetas(thetas);
```

```
    return 0;
}
```

Output:

```
PS E:\GIt\Sem-8\DL\Lab6> cd "e:\GIt\Sem-8\DL\Lab6\" ; if ($?) { g++ gdv2.cpp -o gdv2 } ; if ($?) { .\gdv2 }
Enter CSV file path: data.csv
Enter learning rate: 0.0001
Epoch 1 MSE: 3418.61
Weights: 0.00552248 0.0294322 0.414004 0.0362218 0.0255495
Epoch 2 MSE: 841.176
Weights: 0.00811927 0.0442597 0.61238 0.0533194 0.037652
Epoch 3 MSE: 249.211
Weights: 0.00931405 0.052088 0.707426 0.0612526 0.0433107
Epoch 4 MSE: 113.24
Weights: 0.00983697 0.0565617 0.752956 0.0647942 0.0458816
Epoch 5 MSE: 81.9809
Weights: 0.0100379 0.0594274 0.774758 0.0662314 0.0469729
Epoch 6 MSE: 74.7679
Weights: 0.0100846 0.061522 0.785189 0.0666601 0.0473552
Epoch 7 MSE: 73.077
Weights: 0.0100573 0.0632466 0.790171 0.0666057 0.0473978
Epoch 8 MSE: 72.654
Weights: 0.00999457 0.0647933 0.792543 0.0663197 0.0472777
Epoch 9 MSE: 72.5222
Weights: 0.00991485 0.0662543 0.793663 0.0659227 0.0470798
Epoch 10 MSE: 72.4576
Weights: 0.00982699 0.0676737 0.794183 0.0654726 0.0468446
Final Thetas: 0.00982699 0.0676737 0.794183 0.0654726 0.0468446
PS E:\GIt\Sem-8\DL\Lab6>
```

Conclusion from Output of Batch Gradient Descent

1. **Initial MSE and Rapid Decrease:**
    - The initial **Mean Squared Error (MSE)** starts at **3418.61**, indicating a high error at the beginning.
    - As training progresses, MSE drops significantly, reaching **72.4576** by epoch 10.
    - This suggests that the model is **learning effectively** and adjusting the weights to reduce prediction error.
2. **Weight (Theta) Convergence:**

- The theta values (**weights**) start from **0.0** and gradually update.
- The weights **increase sharply** in the initial epochs but **stabilize** towards the end.
- The final theta values:

0.00982699, 0.0676737, 0.794183, 0.0654726, 0.0468446

- The small updates in later epochs suggest the model is nearing **convergence**.

3. **Learning Rate Impact:**
   - The chosen learning rate (**0.0001**) is **slow but stable**, leading to a smooth decrease in MSE.
   - A **higher learning rate** might have resulted in faster convergence but could also risk divergence.
   - A **lower learning rate** would slow convergence further.
4. **Final Observations:**
   - The model **successfully reduced error** over 10 epochs.
   - However, **MSE is still 72.4576**, which may indicate that:
     - More training epochs might further reduce error.
     - Feature scaling (normalization) might improve efficiency.
     - A slightly higher learning rate could speed up convergence.