

廈門大學



软件学院

《人工智能导论》实验报告

题 目 反向传播算法

姓 名 陈澄

学 号 32420212202930

班 级 软工三班

实验时间 2024/05/02

2024 年 05 月 02 日

1 实验目的

编程实现 BP 神经网络算法；理解算法原理

2 实验步骤

将 Iris（鸢尾花）数据集分为训练集（Iris-train.txt）和测试集（Iris-test.txt），分别含 75 个样本，每个集合中每种花各有 25 个样本。为了方便训练，将 3 类花分别编号为 1，2，3。使用这些数据训练一个 4 输入（分别对应 4 个特征）、隐含层（10 个神经元）、3 输出（分别对应该样本属于某一品种的可能性大小）的神经网络（4*10*3）。

使用训练集对网络进行训练，再预测测试集中每个样本的标签，并输出预测准确率（独立运行 10 次，列出 10 次的准确率，并输出平均准确率和标准差）。

3 实验步骤

1. 定义神经网络结构

```
// 定义神经网络结构
const int INPUT_NODES = 4;
const int HIDDEN_NODES = 10;
const int OUTPUT_NODES = 3;
```

2. 定义学习率和迭代次数

```
// 定义学习率和迭代次数
const double LEARNING_RATE = 0.1;
const int EPOCHS = 1000;
```

3. 定义神经网络类

```
// 定义神经网络类
class NeuralNetwork {
private:
    vector<vector<double>> inputHiddenWeights;
    vector<vector<double>> hiddenOutputWeights;
    vector<double> hiddenBiases;
    vector<double> outputBiases;
```

4. 使用随机值初始化权重和偏差

```
NeuralNetwork() {
    // 初始化权重和偏差
    random_device rd;
    mt19937 gen(rd());
    normal_distribution<double> dist(0.0, 1.0);

    for (int i = 0; i < INPUT_NODES; ++i) {
        vector<double> weights;
        for (int j = 0; j < HIDDEN_NODES; ++j) {
            weights.push_back(dist(gen));
        }
        inputHiddenWeights.push_back(weights);
    }

    for (int i = 0; i < HIDDEN_NODES; ++i) {
        vector<double> weights;
        for (int j = 0; j < OUTPUT_NODES; ++j) {
            weights.push_back(dist(gen));
        }
        hiddenOutputWeights.push_back(weights);
        hiddenBiases.push_back(dist(gen));
    }

    for (int i = 0; i < OUTPUT_NODES; ++i) {
        outputBiases.push_back(dist(gen));
    }
}
```

5. sigmoid 函数（激活函数）

```
// sigmoid函数
double sigmoid(double x) {
    return 1 / (1 + exp(-x));
}
```

6. 神经网络的前向传播：

hiddenOutputs 用于存储隐含层的输出，outputs 用于存储最终的网路输出。通过循环计算隐含层的输出。对于每个隐含层节点，计算其输入加权和，并通过激活函数得到节点的输出值。

接下来，通过类似的方式计算输出层的输出。返回存储了输出层节点输出值的 `outputs` 向量，这些值即为神经网络对给定输入的预测输出。

```
// 计算神经网络输出
vector<double> feedForward(const vector<double>& inputs) {
    vector<double> hiddenOutputs(HIDDEN_NODES);
    vector<double> outputs(OUTPUT_NODES);

    // 计算隐含层输出
    for (int i = 0; i < HIDDEN_NODES; ++i) {
        double sum = 0.0;
        for (int j = 0; j < INPUT_NODES; ++j) {
            sum += inputs[j] * inputHiddenWeights[j][i];
        }
        sum += hiddenBiases[i];
        hiddenOutputs[i] = sigmoid(sum);
    }

    // 计算输出层输出
    for (int i = 0; i < OUTPUT_NODES; ++i) {
        double sum = 0.0;
        for (int j = 0; j < HIDDEN_NODES; ++j) {
            sum += hiddenOutputs[j] * hiddenOutputWeights[j][i];
        }
        sum += outputBiases[i];
        outputs[i] = sigmoid(sum);
    }

    return outputs;
}
```

7. 权重更新:

在前向传播阶段，根据输入计算了隐含层和输出层的输出。然后，在反向传播阶段，根据输出误差和隐藏层误差，利用误差逆传播算法来更新神经网络的权重和偏差。

```

// 更新权重和偏差
void updateWeights(const vector<double>& inputs, const vector<double>& targets) {
    vector<double> hiddenOutputs(HIDDEN_NODES);
    vector<double> outputs(OUTPUT_NODES);

    // 前向传播
    for (int i = 0; i < HIDDEN_NODES; ++i) {
        double sum = 0.0;
        for (int j = 0; j < INPUT_NODES; ++j) {
            sum += inputs[j] * inputHiddenWeights[j][i];
        }
        sum += hiddenBiases[i];
        hiddenOutputs[i] = sigmoid(sum);
    }

    for (int i = 0; i < OUTPUT_NODES; ++i) {
        double sum = 0.0;
        for (int j = 0; j < HIDDEN_NODES; ++j) {
            sum += hiddenOutputs[j] * hiddenOutputWeights[j][i];
        }
        sum += outputBiases[i];
        outputs[i] = sigmoid(sum);
    }

    // 反向传播
    vector<double> outputErrors(OUTPUT_NODES);
    for (int i = 0; i < OUTPUT_NODES; ++i) {
        outputErrors[i] = (targets[i] - outputs[i]) * outputs[i] * (1 - outputs[i]);
    }

    vector<double> hiddenErrors(HIDDEN_NODES);
    for (int i = 0; i < HIDDEN_NODES; ++i) {
        double error = 0.0;
        for (int j = 0; j < OUTPUT_NODES; ++j) {
            error += outputErrors[j] * hiddenOutputWeights[i][j];
        }
        hiddenErrors[i] = error * hiddenOutputs[i] * (1 - hiddenOutputs[i]);
    }

    // 更新输出层权重和偏差
    for (int i = 0; i < OUTPUT_NODES; ++i) {
        for (int j = 0; j < HIDDEN_NODES; ++j) {
            hiddenOutputWeights[j][i] += LEARNING_RATE * outputErrors[i] * hiddenOutputs[j];
        }
        outputBiases[i] += LEARNING_RATE * outputErrors[i];
    }

    // 更新隐含层权重和偏差
    for (int i = 0; i < HIDDEN_NODES; ++i) {
        for (int j = 0; j < INPUT_NODES; ++j) {
            inputHiddenWeights[j][i] += LEARNING_RATE * hiddenErrors[i] * inputs[j];
        }
        hiddenBiases[i] += LEARNING_RATE * hiddenErrors[i];
    }
}

```

8. Main()方法

读取训练集和数据集文件，创建神经网络进行训练


```

int main() {
    // 读取训练集和测试集数据
    ifstream trainFile("D:\\learn\\大三下\\人工智能导论\\实验五决策树算法\\traindata.txt");
    ifstream testFile("D:\\learn\\大三下\\人工智能导论\\实验五决策树算法\\testdata.txt");
    if (!trainFile.is_open() || !testFile.is_open()) {
        cout << "Failed to open files." << endl;
        return 1;
    }

    // 读取数据并转换为神经网络可用的格式
    vector<vector<double>> trainInputs;
    vector<vector<double>> trainTargets;
    vector<vector<double>> testInputs;
    vector<vector<double>> testTargets;

    string line;
    while (getline(trainFile, line)) {
        stringstream ss(line);
        vector<double> values;
        double val;
        while (ss >> val) {
            values.push_back(val);
        }
        trainInputs.push_back({ values[0], values[1], values[2], values[3] });
        trainTargets.push_back({ values[4] == 0.0 ? 1.0 : 0.0,
                                values[4] == 1.0 ? 1.0 : 0.0,
                                values[4] == 2.0 ? 1.0 : 0.0 });
    }

    while (getline(testFile, line)) {
        stringstream ss(line);
        vector<double> values;
        double val;
        while (ss >> val) {
            values.push_back(val);
        }
        testInputs.push_back({ values[0], values[1], values[2], values[3] });
        testTargets.push_back({ values[4] == 0.0 ? 1.0 : 0.0,
                                values[4] == 1.0 ? 1.0 : 0.0,
                                values[4] == 2.0 ? 1.0 : 0.0 });
    }

    // 创建神经网络
    NeuralNetwork nn;

    // 训练神经网络
    for (int epoch = 0; epoch < EPOCHS; ++epoch) {
        double totalError = 0.0;
        for (size_t i = 0; i < trainInputs.size(); ++i) {
            auto outputs = nn.feedForward(trainInputs[i]);
            vector<double> targets(trainTargets[i].begin(), trainTargets[i].end());
            totalError += 0.5 * ((targets[0] - outputs[0]) * (targets[0] - outputs[0]) +
                                (targets[1] - outputs[1]) * (targets[1] - outputs[1]) +
                                (targets[2] - outputs[2]) * (targets[2] - outputs[2]));
            nn.updateWeights(trainInputs[i], targets);
        }
        cout << "Epoch: " << epoch << ", Error: " << totalError << endl;
    }

    // 在测试集上测试神经网络
    int correct = 0;
    for (size_t i = 0; i < testInputs.size(); ++i) {
        auto outputs = nn.feedForward(testInputs[i]);
        vector<double> targets(testTargets[i].begin(), testTargets[i].end());

        // 将输出转换为类别
        int predictedClass = distance(outputs.begin(), max_element(outputs.begin(), outputs.end()));
        int trueClass = distance(targets.begin(), max_element(targets.begin(), targets.end())) + 1;

        if (predictedClass == trueClass) {
            correct++;
        }
    }

    double accuracy = static_cast<double>(correct) / testInputs.size() * 100.0;
    cout << "Test Accuracy: " << accuracy << "%" << endl;

    return 0;
}

```

4 实验结果

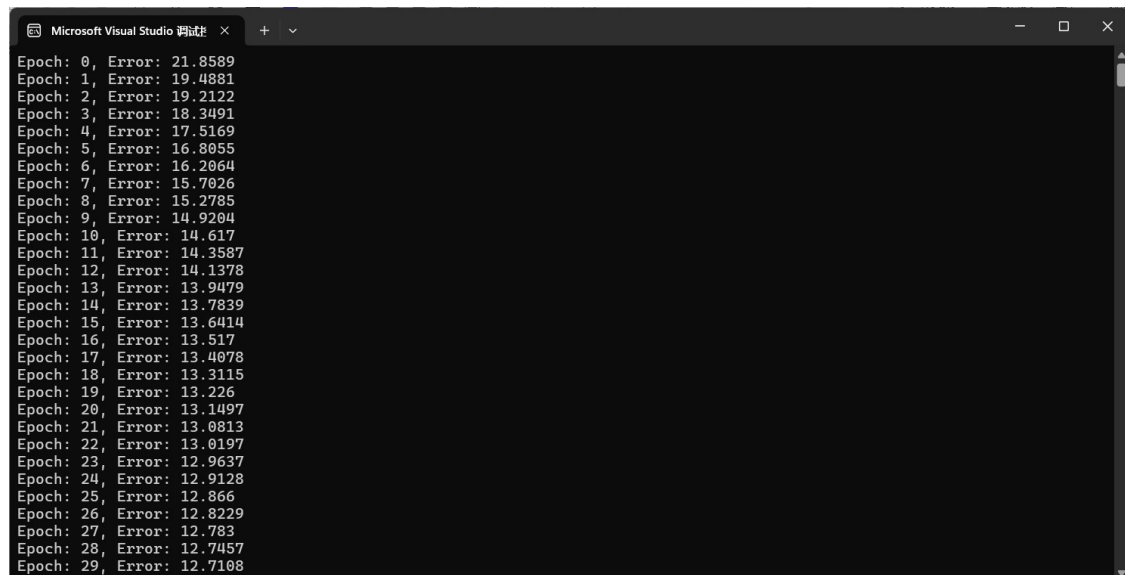
Epoch 为迭代次数，Error 为误差，计算方式如下：

Targets[i] 为第 i 种花的目标值（是第 i 种花为 1，其余为 0）

Outputs[i] 为第 i 种花的预测值（可以理解为是 i 种花的概率）

```
totalError += 0.5 * ((targets[0] - outputs[0]) * (targets[0] - outputs[0]) +  
    (targets[1] - outputs[1]) * (targets[1] - outputs[1]) +  
    (targets[2] - outputs[2]) * (targets[2] - outputs[2]));
```

经过 1000 次迭代误差值逐渐趋近于 0



```
Microsoft Visual Studio 调试  x + -  
Epoch: 0, Error: 21.8589  
Epoch: 1, Error: 19.4881  
Epoch: 2, Error: 19.2122  
Epoch: 3, Error: 18.3491  
Epoch: 4, Error: 17.5169  
Epoch: 5, Error: 16.8055  
Epoch: 6, Error: 16.2064  
Epoch: 7, Error: 15.7026  
Epoch: 8, Error: 15.2785  
Epoch: 9, Error: 14.9204  
Epoch: 10, Error: 14.617  
Epoch: 11, Error: 14.3587  
Epoch: 12, Error: 14.1378  
Epoch: 13, Error: 13.9479  
Epoch: 14, Error: 13.7839  
Epoch: 15, Error: 13.6414  
Epoch: 16, Error: 13.517  
Epoch: 17, Error: 13.4078  
Epoch: 18, Error: 13.3115  
Epoch: 19, Error: 13.226  
Epoch: 20, Error: 13.1497  
Epoch: 21, Error: 13.0813  
Epoch: 22, Error: 13.0197  
Epoch: 23, Error: 12.9637  
Epoch: 24, Error: 12.9128  
Epoch: 25, Error: 12.866  
Epoch: 26, Error: 12.8229  
Epoch: 27, Error: 12.783  
Epoch: 28, Error: 12.7457  
Epoch: 29, Error: 12.7108
```

```
Microsoft Visual Studio 调试
Epoch: 974, Error: 0.790762
Epoch: 975, Error: 1.34879
Epoch: 976, Error: 0.989609
Epoch: 977, Error: 0.801992
Epoch: 978, Error: 1.4088
Epoch: 979, Error: 0.999017
Epoch: 980, Error: 0.688195
Epoch: 981, Error: 1.45279
Epoch: 982, Error: 0.984684
Epoch: 983, Error: 0.831442
Epoch: 984, Error: 1.17641
Epoch: 985, Error: 0.744674
Epoch: 986, Error: 1.283
Epoch: 987, Error: 1.13477
Epoch: 988, Error: 1.28618
Epoch: 989, Error: 1.15963
Epoch: 990, Error: 0.991713
Epoch: 991, Error: 1.29178
Epoch: 992, Error: 0.992109
Epoch: 993, Error: 0.726306
Epoch: 994, Error: 1.33028
Epoch: 995, Error: 1.15312
Epoch: 996, Error: 1.27181
Epoch: 997, Error: 1.06689
Epoch: 998, Error: 1.29108
Epoch: 999, Error: 0.875983
Test Accuracy: 97.3333%

C:\Users\CC507\source\repos\人工智能导论\Project6\x64\Debug\Project6.exe (进程 37136)已退出, 代码为 0。
按任意键关闭此窗口。 . . .
```

最终预测精确率为 97.33%

5 我的体会

在这次实验中，我尝试了使用神经网络进行训练和预测的过程。通过这个实验，我对神经网络的工作原理和训练过程有了更深入的了解，学会了神经网络的前向传播和反向传播的原理和过程，也进行了相应的代码编写。我相信神经网络在未来的应用中有着巨大的潜力，期待继续深入学习和探索这个领域的知识。