南京信息工程大学

《程序设计实训》报告

题　　目　神经网络 BP 算法的实现

院　　系　应用技术学院

年级班级　2017 计科

学生姓名　万宇轩

学　　号　201733050050

学　　期　2019-2020（二）

任课教师　黄　群

二〇二0年 六 月 十 日

1. 引言

1.1. 人工神经网络的由来

在生物神经网络中，每个神经元与其他神经元相连，当它“兴奋”时，就会向相连的神经元发送化学物质，从而改变这些神经元内的电位；如果某神经元的电位超过了一个“阈值”，那么它就会被激活，即“兴奋”起来，同时向其他神经元发送化学物质。受此启发，人工神经网络便诞生了，虽然不同的神经网络算法有所不同，但都依赖于这种神经元之间的信息传递方式。

1.2. 人工神经网络

人工神经网络（Artificial Neural Networks，简写为 ANNs）也简称为神经网络（NNs）或称作连接模型（Connection Model），它是一种模仿动物神经网络行为特征，进行分布式并行信息处理的算法数学模型。这种网络依靠系统的复杂程度，通过调整内部大量节点之间相互连接的关系，从而达到处理信息的目的。

常见的神经网络所使用的大都是层级结构，每层神经元与下一神经元全互连，神经元之间不存在同层连接，也不存在跨层连接。这样的神经网络通常称为”多层前馈神经网络“（multi-layer feedforward neural networks）。其中输入层神经元接收外界输入，隐层与输出层神经元对信号进行加工，最终结果由输出层神经元输出；换言之，输入层神经元仅是接受输入，不进行函数处理，隐层与输出层包含功能神经元。神经网络的学习过程，就是根据训练数据来调整神经元之间的“连接权”以及每个功能神经元的阈值；换言之，神经网络“学”到的东西，蕴涵在连接权与阈值中。

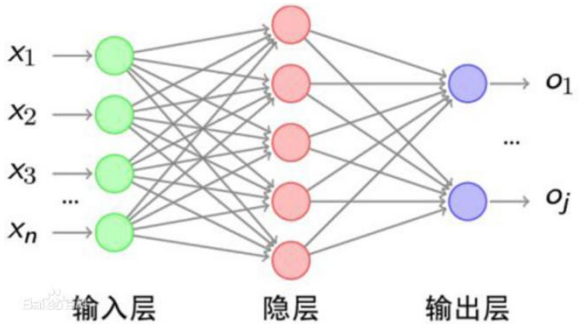

图 1 人工神经网络结构

1.3. 人工神经网络发展历程

人工神经网络最早的研究是 40 年代心理学家 Mcculloch 和数学家 Pitts 合作提出的，他们提出的 MP 模型拉开了神经网络研究的序幕。

神经网络的发展大致经过 3 个阶段：1947～1969 年为初期，在这期间科学家们提出了许多神经元模型和学习规则，如 MP 模型、HEBB 学习规则和感知器等；60 年代末期至 80 年代中期，神经网络控制与整个神经网络研究一样，处于低潮。在此期间，科学家们做了大量的工作，如 Hopfield 教授对网络引入能量函数的概念，给出了网络的稳定性判据，提出了用于联想记忆和优化计算的途径。1984 年，Hiton 教授提出 Bol tzman 机模型；1986 年 Kumelhart 等人提出误差反向传播神经网络，简称 BP 网络。目前，BP 网络已成为广泛使用的网络。1987 年至今为发展期，在此期间，神经网络受到国际重视，各个国家都展开研究，形成神经网络发展的另一个高潮。

2. 需求分析

本课程设计主要设计并实现了一个神经网络，并基于此神经网络，构建了一个手写数字识别系统，包括用户手写数据的输入与采集，神经网络的底层实现与搭建，和使用 MNIST 数据集对神经网络进行训练等。

具体的功能上，用户可以通过鼠标在画板上写 0~9，系统获取到用户的输入后，将其输入到已提前训练好的神经网络模型中，经过具体的运算后来判断用户的输入并返回其预测结果。

3. 总体设计方案

3.1. 总体功能结构

总体可分为三个部分，第一部分是用户手写数据的输入与采集部分，第二部分为神经网络的底层实现与搭建，包括 BP 神经网络中前向传播与反向传播的实现，具体神经元层与权值连接层的实现，使用批梯度下降来对神经网络中权值误差进行调整，神经网络模型的保存等，最后一部分是使用 MNIST 数据集对神经网络进行训练，包括 MNIST 数据集的读取与解析，MNIST 神经网络的搭建、训练、测试与预测等。

图 2 功能结构图（1）

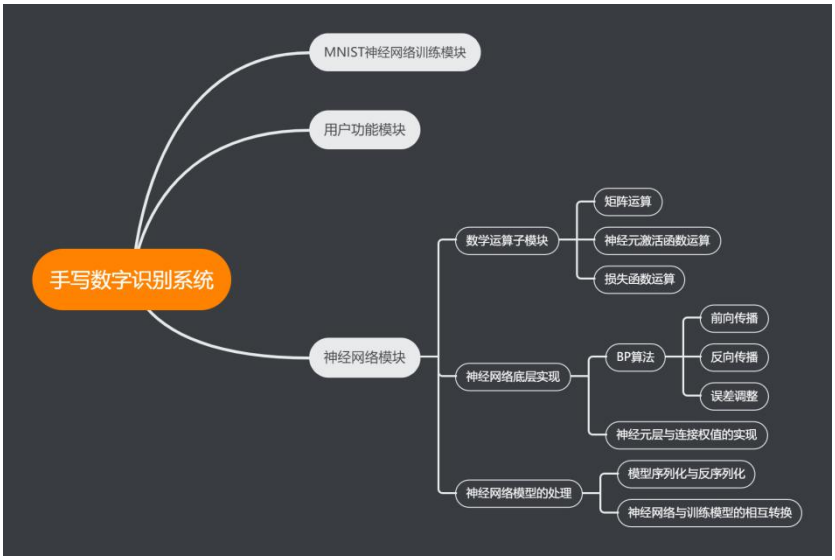

图 3 功能结构图（2）

3.2. 各功能模块算法设计及流程

3.2.1. 用户功能模块

用户模块功能较为简单，仅包含用户数据的采集读取，以及预测结果的展示。

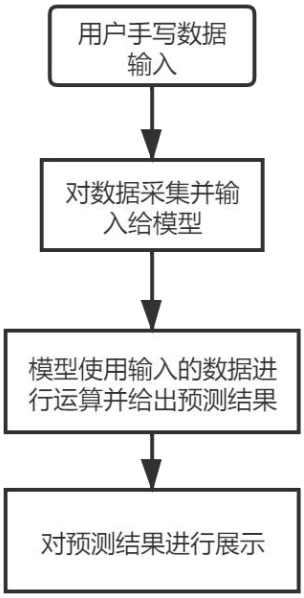

图 4 用户功能模块流程图

3.2.2. MNIST 神经网络训练模块

该模块主要是使用 MNIST 数据集（手写数字数据集）对神经网络进行训练，得出一个可以识别手写数字的模型，从而用于用户数据输入后的识别功能。

该部分主要包含训练数据的读取，神经网络的搭建，模型的训练，训练模型的保存，训练模型的测试和使用训练得到的模型进行预测。

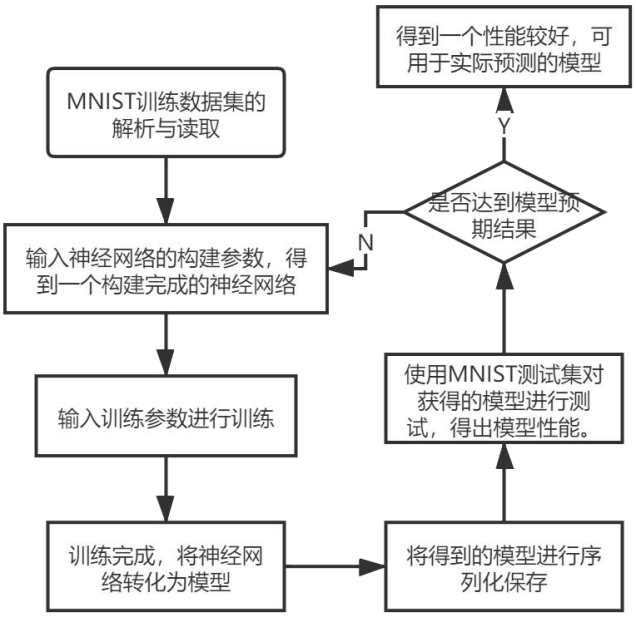

图 5 模型训练流程图

3.2.3. 神经网络模块

神经网络模块为本课程设计的核心模块，包含三个子模块，分别为数学运算的实现，神经网络的底层实现以及神经网络模型的保存等。

1) 数学运算

该部分包含对矩阵的一些运算和处理，例如矩阵的加减乘运算，以及矩阵转置、矩阵标量乘法和矩阵的 Hadamard 乘积运算等。同时还有用于对神经元进行激活操作的激活函数 Sigmoid 函数，和用于计算损失的均方误差 MSE 函数。

2) 神经网络底层

该部分主要设计了一个简易神经网络框架，可根据传入的一些网络参数自动构建一个神经网络，同时使用 BP 算法对神经网络的权值进行更新，也可以从已经训练好的模型中恢复原来的神经网络。

该框架自动构建神经网络的流程主要是：首先输入神经网络的构建所需的参数，如输入层、输出层与各个隐藏层的神经元个数，除第一层输入层外，每一层神经网络所使用的激活函数等，接着对这些输入的参数进行一些预处理，主要是使用默认值对一些缺失参数进行补全，然后根据这些参数从输入层开始逐层创建神经元层，并将创建的神经元层添加到当前神经网络中，最后对所有的神经元层进行连接操作。

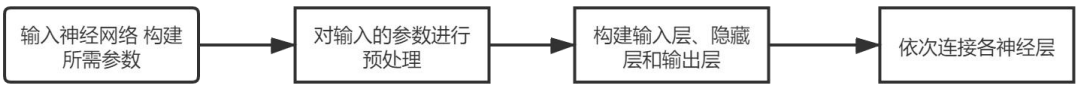

图 6 神经网络自动构建流程

BP 算法是整个神经网络的核心，他主要用于对神经网络中的连接权值进行更新，主要分为以下几个部分：前向传播、误差计算，误差反向传播与权值更新。

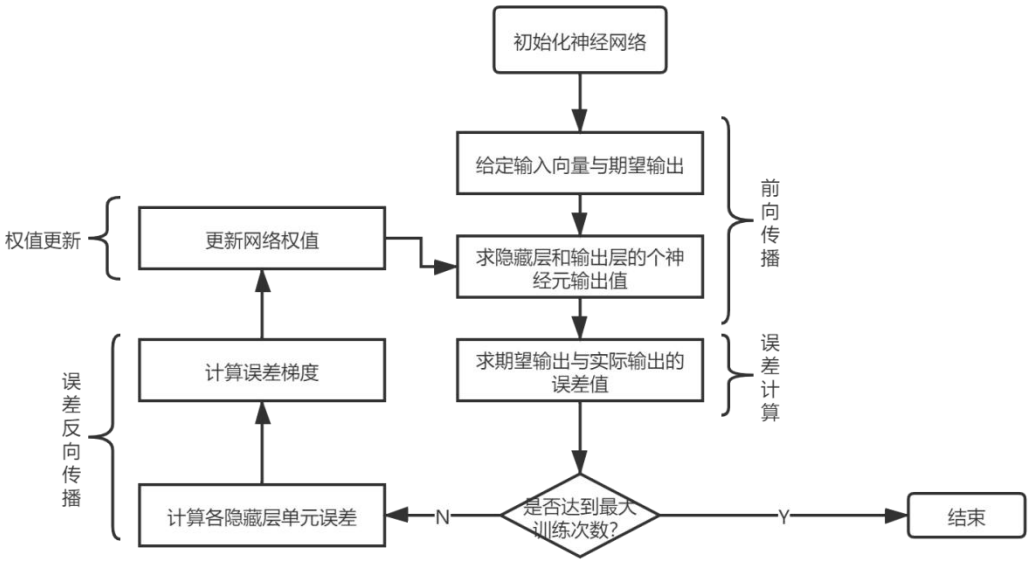

图 7 BP 算法大体流程图

下面简要介绍一下前向传播、误差反向传播与权值更新的具体内容。

BP 神经网络的前向传播比较简单，以下图为例，说明一下前向传播的过程。

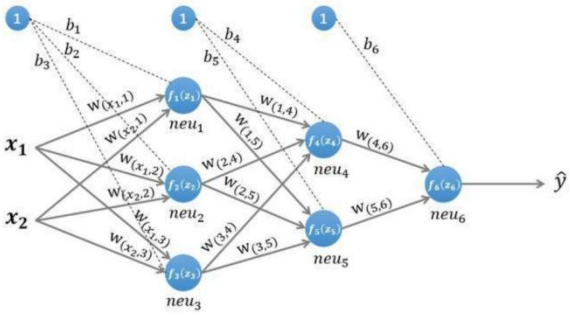

图 8 前向传播图例

如上图所示，现设有一个输入样本：

$$\vec{a} = (x_1, x_2)$$

图 9 输入样本

图中第一层网络的参数为：

$$W^{(1)} = \begin{bmatrix} W_{(x_1,1)}, W_{(x_2,1)} \\ W_{(x_1,2)}, W_{(x_2,2)} \\ W_{(x_1,3)}, W_{(x_2,3)} \end{bmatrix}, \quad b^{(1)} = [b_1, b_2, b_3]$$

图 10 第一层网络参数

图中第二层网络的参数为：

$$W^{(2)} = \begin{bmatrix} W_{(1,4)}, W_{(2,4)}, W_{(3,4)} \\ W_{(1,5)}, W_{(2,5)}, w_{(3,5)} \end{bmatrix}, \quad b^{(2)} = [b_4, b_5]$$

图 11 第二层网络参数

图中第三层网络的参数为：

$$W^3 = \left[w_{(4,6)}, w_{(5,6)}\right], \quad b^{(3)} = [b_6]$$

图 12 第三层网络参数

上述的参数中，W 矩阵为各神经层之间神经元的连接权值，b 为偏置单元的连接权值。这里以第一层隐藏层计算为例，介绍一下前向传播在隐藏层中是如何计算的。

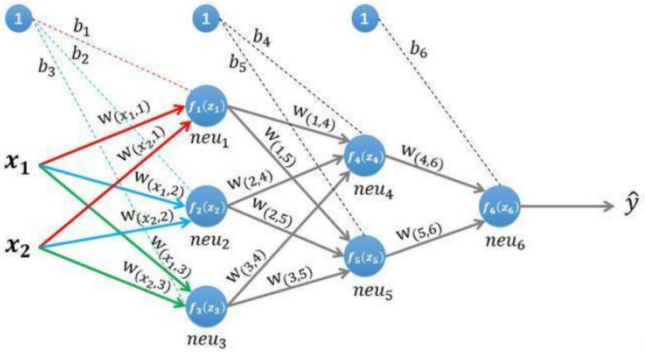

图 13 第一层隐藏层

该层的输入是输入样例经过权值网络计算后的结果

即：

$$Z^{(1)} = W^{(1)} * (\vec{a})^T + \left(b^{(1)}\right)^T$$

图 14 第一层输入

这里以 neu1 神经元为例，其输入值为：

$$z_1 = w_{(x_1,1)} * x_1 + w_{(x_2,1)} * x_2 + b_1$$

图 15 neu1 神经元输入值

其中 W（x1,1）为输入样例中 x1 到该神经元的权重，W（x2,1）为输入样例中 x2 到该神经元的权重，b1 为偏置单元（一般设为常量 1）到达该神经元的权重。

现在我们有了输入，那么接下去就是对每一个输入进行激活操作，也就是使用一个激活函数对输入值进行计算操作。一般而言，同一层的激活函数是一样的，常用的激活函数有：Sigmoid、ReLu、Tanh 等，这里我们使用 Sigmoid 作为我们的激活函数。

Sigmoid 公式为：

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$$

图 16 sigmoid 公式

Sigmoid 图像为：

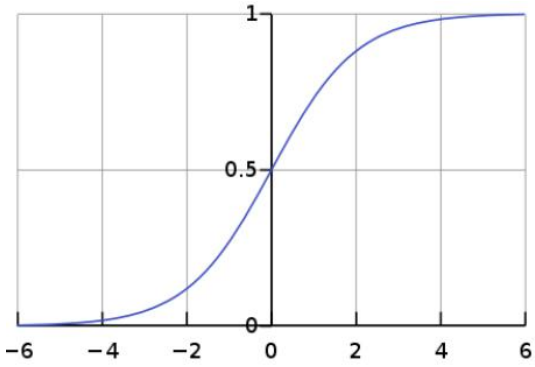

图 17 sigmoid 函数图像

因此，第一层在经过激活函数激活之后的输出结果为：

$$O^{(1)} = Sigmoid(Z^{(1)}) = Sigmoid(W^{(1)} * (\vec{a})^T + (b^{(1)})^T)$$

图 18 第一层神经元输出

后面两层以此类推，最后我们得到输出层的结果为：

$$Z^{(3)} = W^{(3)} * (O^{(2)})^T + (b^{(3)})^T$$

$$O^{(3)} = \text{Sigmoid}\left(Z^{(3)}\right)$$

图 19 输出层结果

综上，前向传播的公式为：

$$Z^{(l)} = W^{(l)} * \left(O^{(l-1)}\right)^T + \left(b^{(l)}\right)^T$$
$$O^{(l)} = \text{Sigmoid}\left(Z^{(l)}\right)$$

图 20 前向传播公式

在本课程设计中，前向传播算法的流程图为：

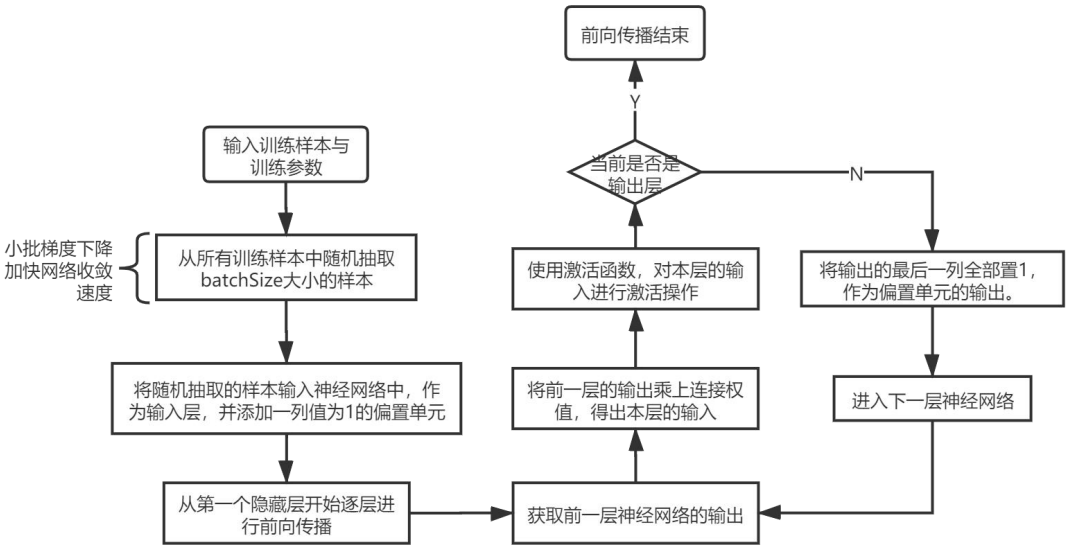

图 21 前向传播流程

BP 算法中的反向传播和权值更新是其关键所在，所谓的反向传播，它传播的是，正向传播最后的输出值与真实值的误差对每一层权值网络 W 的偏导数，然后使用 W 的偏导数对 W 权值进行更新，从而降低整体神经网络权值的误差。

现在我们假设上述正向传播样例中的真实值为 y，我们的输出为$\hat{y}$，那么我们的误差，或者说损失函数的值为：

$$Loss(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$$

图 22 损失函数

BP 算法的权重更新方法，也就是梯度下降法（在具体的课程代码中，使用的是小批梯度下降法，思想是类似的）：

$$\boldsymbol{W}_l = \boldsymbol{W}_l - \eta \frac{\partial L}{\partial \boldsymbol{W}_l}$$

图 23 权重更新

我们无法直接用损失函数对权重 W 进行直接求偏导，但可以使用求导链式法则，进行间接求偏导，，即：

$$\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{W}_l} = \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{z}_l} \frac{\partial \boldsymbol{z}_l}{\partial \boldsymbol{W}_l}$$

图 24 公式 1

我们定义左边的分式为 $\boldsymbol{\xi}$ ，即：

$$\boldsymbol{\xi}_l = \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{z}_l}$$

图 25 公式 2

而右边分式求导后的结果为：

$$\frac{\partial \boldsymbol{z}_l}{\partial \boldsymbol{W}_l} = o_{l-1}^T$$

图 26 公式 3

所以我们将公式 2 与 3 代入公式 1 后，得到：

$$\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{W}_l} = \boldsymbol{\xi}_l o_{l-1}^T$$

图 27 公式 4

然后再根据以下两个公式：

$$\boldsymbol{z}_{l+1} = \boldsymbol{W}_{l+1} \boldsymbol{o}_l$$

图 28 公式 6

$$\boldsymbol{\xi}_l = \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{o}_l} \frac{\partial \boldsymbol{o}_l}{\partial \boldsymbol{z}_l}$$

图 29 公式 7

可得到：

$$\boldsymbol{\xi}_l = \frac{\partial L}{\partial \boldsymbol{z}_{l+1}} \frac{\partial \boldsymbol{z}_{l+1}}{\partial \boldsymbol{o}_l} \frac{\partial \boldsymbol{o}_l}{\partial \boldsymbol{z}_l}$$

图 30 公式 8

从公式 8 的第一个分式，以及第 2 个公式，我们可以得出：

$$\frac{\partial L}{\partial \boldsymbol{z}_{l+1}} = \boldsymbol{\xi}_{l+1}$$

图 31 公式 9

然后，从公式 8 的第二个分式，以及第 6 个公式，我们可以得出：

$$\frac{\partial \boldsymbol{z}_{l+1}}{\partial \boldsymbol{o}_l} = \frac{\partial W_{l+1} O_l}{\partial O_l} = W_{l+1}^T$$

图 32 公式 10

接着，对于公式 8 的最后一个分式，由：

$$o_l = f(z_l) = \text{Sigmoid}(z_l)$$

可得出：

$$\frac{\partial \boldsymbol{o}_l}{\partial \boldsymbol{z}_l} = f^{'}(z_l)$$

图 33 公式 11

最后，将公式 9~11 代入公式 8 中，我们得到：

$$\boldsymbol{\xi}_l = \frac{\partial L}{\partial \boldsymbol{z}_{l+1}} \frac{\partial \boldsymbol{z}_{l+1}}{\partial \boldsymbol{o}_l} \frac{\partial \boldsymbol{o}_l}{\partial \boldsymbol{z}_l} = (W_{l+1}^T * \boldsymbol{\xi}_{l+1}) \odot f^{'}(z_l)$$

图 34 公式 12

从公式 12 中，根据数学归纳法，我们可以知道，只要求出最后一层的误差，就可以将所有的误差都求出来，而最后一层误差为：

$$\begin{aligned} \boldsymbol{\xi}_{Last} &= \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{z}_{Last}} \\ &= \frac{\partial \left(\frac{1}{2} ||\boldsymbol{o}_{Last} - \boldsymbol{y}||^2 \right)}{\partial \boldsymbol{z}_{Last}} \\ &= (\boldsymbol{o}_{Last} - \boldsymbol{y}) \circ \boldsymbol{f}'(\boldsymbol{z}_{Last}) \end{aligned}$$

图 35 公式 13

到此，我们就可以使用梯度下降法，来对权值网络进行更新操作，上述公式整理后为：

$$\begin{aligned} W_{\text{Last}} &= W_{\text{Last}} - \eta \frac{\partial L}{\partial W_{\text{Last}}} \\ &= W_{\text{Last}} - \eta (o_{\text{Last}} - y) \odot f'(z_{\text{Last}}) o_{\text{Last}-1}^T \end{aligned}$$

图 36 公式 14

$$\begin{aligned} W_l &= W_l - \eta \frac{\partial L}{\partial W_l} \\ &= W_l - \eta (W_{l+1}^T * \xi_{l+1}) \odot f^{'}(z_l) o_{l-1}^T \end{aligned}$$

图 37 公式 15

在本课程设计中，反向传播与权值更新算法的流程图为：

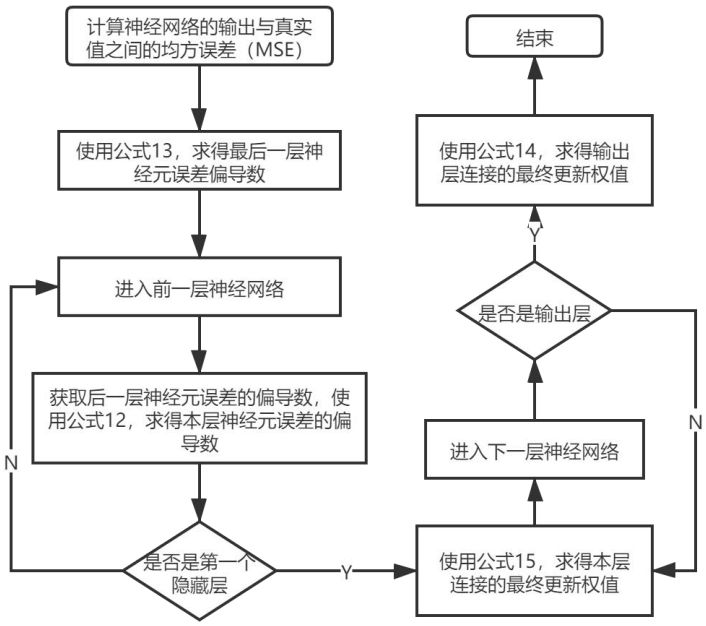

图 38 反向传播与误差更新流程图

4. 系统实现

由于篇幅所限，此部分主要给出本课程设计中的核心代码。

4.1. 神经元层：

该部分主要是对神经网络中单一神经元层的实现，在神经元层中主要包含以下几个参数：神经元的数量，当前神经元层的类型，类型可以是输入、输出和隐藏层，该神经层所使用的的激活函数类型等。

```
/**
 * @param neuronNum     神经元个数
 * @param layerType     网络类型
 * @param activationFn 激活函数
 * @param addBias       是否需要手动添加偏置
 */
public NeuronsLayer(final int neuronNum, final LayerType layerType, final ActivationFn activationFn, boolean addBias) {
    this.layerType = layerType;
    this.activationFn = activationFn;
        this.hasBias = layerType != LayerType.OUTPUT;
        if (addBias && this.hasBias) {
            this.neuronNum = neuronNum + 1;//加一个偏置单元
        } else {
            this.neuronNum = neuronNum;
        }
    }
```

4.2. 连接权值层

该部分主要实现了神经元层之间的连接权值网络，具体底层是使用二维数组进行实现，该部分为权值网络初始化的代码，使用高斯分布来对连接权值进行初始化的操作。

```
/**
 * @description 进行连接权重的初始化工作，初始化使用高斯分布
 */
private void initWeights() {
    int preNum = preLayer.getNeuronNum();
    int nextNum = nextLayer.getNeuronNum();
    double[][] initialMatrix = new double[nextNum][preNum];
    Random random = new Random();
    //填充数据
        for (int nextIndex = 0; nextIndex < nextNum; nextIndex++) {
            for (int preIndex = 0; preIndex < preNum; preIndex++) {
                initialMatrix[nextIndex][preIndex] = random.nextGaussian();//使用高斯分布进行值的初始化
            }
        }
        this.weightMatrix = new Matrix(initialMatrix);
    }
```

4.3. 神经网络训练

该部分为整个神经网络的训练代码，训练的步骤主要为，首先从所有的样例输入中随机抽取指定个数的样本，这些样本用于小批量梯度学习，然后将抽取的样本输入到构建好的神经网络中进行前向传播计算，前向传播完成后，进行整体误差计算，将计算的误差进行反向传播，最后进行更新。

```
/**
 * @param inputMatrix      输入值
 * @param realOutputMatrix 真实值
 * @description 使用所输入的样本进行训练
 */
private void train(final Matrix inputMatrix, final Matrix realOutputMatrix) {
    NeuronsLayer inputLayer = this.neuronsLayers.get(0);
    assert inputLayer.getLayerType() == LayerType.INPUT;
    inputLayer.setInputMatrix(inputMatrix);
        //前向传播
        for (int layerIndex = 1; layerIndex < neuronsLayers.size(); layerIndex++) {

            NeuronsLayer curLayer = neuronsLayers.get(layerIndex);
            curLayer.forward();
        }

        //获取输出层
```

```
        NeuronsLayer outputLayer = neuronsLayers.get(neuronsLayers.size() - 1);
        assert outputLayer.getLayerType() == LayerType.OUTPUT;

        //从输出层开始进行反向误差计算
        outputLayer.calErrorDelta(realOutputMatrix);
     for (int layerIndex = neuronsLayers.size() - 2; layerIndex >= 1; layerIndex--)
        {//注：输入层不需要计算神经元误差
            neuronsLayers.get(layerIndex).calErrorDelta();//计算神经元误差，并进行反向
             传播
        }
        //从第一个隐藏层开始，调整神经网络权重
        for (NeuronsLayer neuronsLayer : neuronsLayers) {
            if (neuronsLayer.getLayerType() != LayerType.INPUT)
                neuronsLayer.adjustment(learningRate);
        }
    }

    /**
     * @param learningRate 学习率
     * @param trainSteps   训练步数
     * @param batchSize    每次批梯度下降的 batch 大小
     * @param inputSet     输入数据
     * @param outputSet    输出数据集
     */
    public void trainStart(double learningRate, int trainSteps, int batchSize, doubl
        e[][] inputSet, double[][] outputSet) {
        this.learningRate = learningRate;
        NeuronsLayer outputLayer = neuronsLayers.get(neuronsLayers.size() - 1);
        Random random = new Random();
        for (int stepIndex = 0; stepIndex < trainSteps; stepIndex++) {
            //从总的训练样本中，随机抽取 batch 大小的训练样本用于本次训练
            double[][] batchInput = new double[batchSize][inputSet[0].length];
            double[][] batchOutput = new double[batchSize][outputSet[0].length];
            for (int batchIndex = 0; batchIndex < batchSize; batchIndex++) {
                int randomIndex = random.nextInt(inputSet.length);
                batchInput[batchIndex] = inputSet[randomIndex];
                batchOutput[batchIndex] = outputSet[randomIndex];
            }
            Matrix batchInputMatrix = new Matrix(batchInput);
            Matrix batchOutputMatrix = new Matrix(batchOutput);
            //开始训练
            this.train(batchInputMatrix, batchOutputMatrix);

            if (stepIndex % 100 == 0) {
```

```
58.                 //每 100 次输出一次训练结果
59.                 System.out.println("stepIndex: " + stepIndex + " of " + trainSteps);
60.                 System.out.println("loss value: " + LossFn.L2Loss(batchOutputMatrix,
                     outputLayer.getOutput()) + "\n");
61.             }
62.             if (stepIndex % 1000 == 0 && stepIndex != 0) {
63.                 //每 1000 次保存一次模型
64.                 System.out.println("输出一个模型，编号" + stepIndex + ",
                     loss value: " + LossFn.L2Loss(batchOutputMatrix,
65.                     outputLayer.getOutput()) + "\n");
66.                 String modelName = DigitalModel.getModelSavedPath() + "\\netModel_"
                         + stepIndex + ".model";
67.                 System.out.println(modelName);
68.                 NeuralNet.saveModel(this, modelName);
69.             }
70.
71.         }
72.     }
```

4.4. BP 算法核心

4.4.1. BP 算法之前向传播

该部分是 BP 算法的前向传播部分，主要操作是权值连接计算，激活操作等，主要的流程见下图。

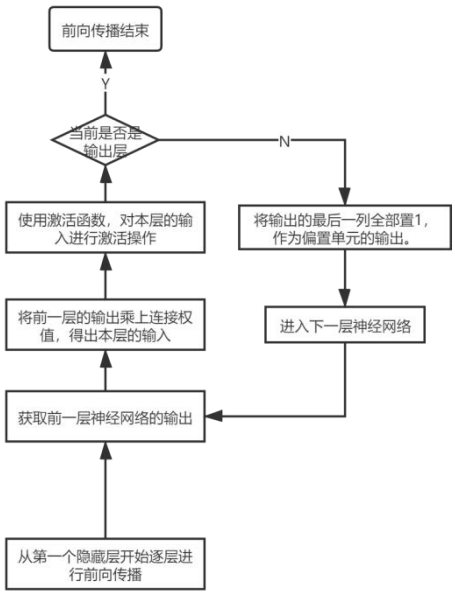

图 39 前向传播计算流程

```
1. /**
2.  * @description 进行前向传播操作
3.  */
4. public void forward() {
5.     assert this.preWeights != null;
```

```
6.      NeuronsLayer preLayer = this.preWeights.getPreLayer();//获取前一层的神经元
7.      assert preLayer != null;
8.
9.      Matrix noneActivateMatrix = MatrixOperation.matrixMul(preLayer.getOutput(),
10.                 MatrixOperation.transposition(this.preWeights.getWeightMatrix()));
11.                //计算传播过来的值，也就是本层神经网络的输入
12.         this.activatedMatrix = ActivationFn.useActivationFn(
13.                 noneActivateMatrix, this.activationFn::activeFn);
14.                 //使用激活函数，对神经元进行激活操作
15.         if (this.hasBias) {
16.             //如果需要偏置单元，则将最后一列设置为常数 1
17.             Matrix.setAppointColumnVal(this.activatedMatrix, this.neuronNum - 1, 1);

18.         }
19.     }
```

4.4.2. BP 算法之反向传播

反向传播主要进行误差的传播计算，求得最终的预测值与样本的输出值之间的误差，然后计算各个神经元层误差的偏导数，该步骤求得的误差偏导数主要用于下一阶段的权值更新中。

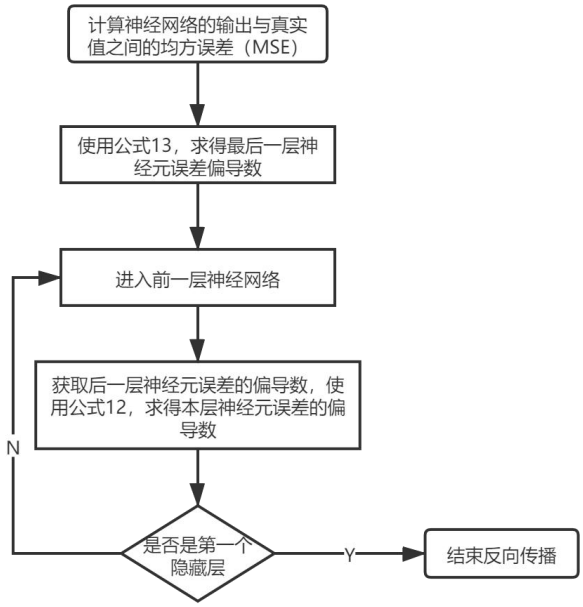

图 40 反向传播计算

```
1. /**
2.  * @param outputMatrix 训练结果集
3.  * @description 反向传播，计算神经元误差，只可用于输出层的神经元
4.  */
5. public void calErrorDelta(Matrix outputMatrix) {
6.     assert this.layerType == LayerType.OUTPUT;
7.     //计算损失
8.     Matrix errorMatrix = MatrixOperation.matrixSub(outputMatrix,
```

```
        this.activatedMatrix);
        //计算每一个神经元的误差值的偏导数值
        this.errorDerivative = MatrixOperation.hadamardMul
                (errorMatrix, ActivationFn.useActivationFn
                        (this.activatedMatrix,
                        this.activationFn::activeFnDerivative));
    }

    /**
     * @description 反向传播，计算神经元误差，只可用于隐藏层
     */
    public void calErrorDelta() {
        assert this.layerType == LayerType.HIDDEN;
        NeuronsLayer nextLayer = this.nextWeights.getNextLayer();
        assert nextLayer != null;
        //计算损失
        Matrix errorMatrix = MatrixOperation.matrixMul
                (nextLayer.errorDerivative, this.nextWeights.getWeightMatrix());
        //计算每一个神经元的误差的偏导数值
        this.errorDerivative = MatrixOperation.hadamardMul
                (errorMatrix, ActivationFn.useActivationFn
                        (this.activatedMatrix, this.activationFn::activeFnDerivative)
);
    }
```

4.4.3. 连接权重调整

以下两段代码主要实现了，使用在反向传播中计算得到的误差偏导数来对各层神经网络权值进行更新的操作，从而对整体的神经网络实现调整，使得预测的结果更加贴近本次样本的真实输出值。

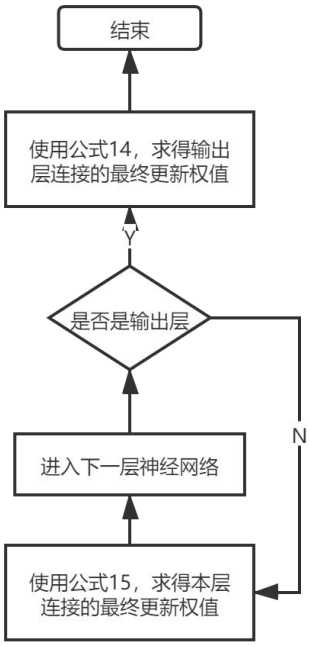

图 41 连接权值更新

```java
/**
 * @description 神经网络，权值误差调整
 */
public void adjustment(double learningRate) {
    assert this.preWeights != null;
    //计算要调整的权重误差
    NeuronsLayer preLayer = this.preWeights.getPreLayer();
    assert preLayer != null;
    //计算权重误差
        Matrix weightsDelta = MatrixOperation.matrixMul(
                MatrixOperation.transposition(this.errorDerivative),
                preLayer.activatedMatrix);
        //使用计算的误差调整权重网络
        this.preWeights.updateWeights(weightsDelta, learningRate);
    }
```

```java
/**
 * @param weightsDelta 当前连接权重的误差偏微分
 */
public void updateWeights(Matrix weightsDelta, double learningRate) {
    //使用梯度下降法计算新的权重值
    this.weightMatrix = MatrixOperation.matrixAdd(this.weightMatrix,
            MatrixOperation.scalarMul(learningRate, weightsDelta));
}
```

5. 程序运行测试

5.1. 系统测试

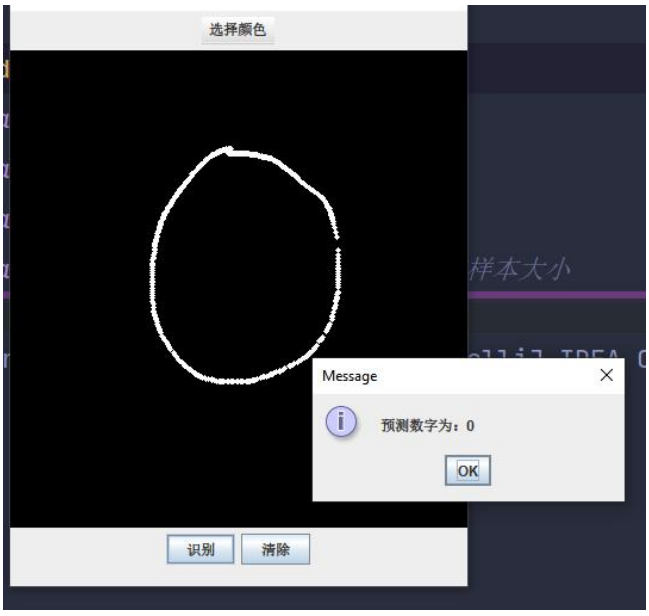

图 42 测试手写数字 0

图 43 测试手写数字 1

图 44 测试手写数字 2

图 45 测试手写数字 3

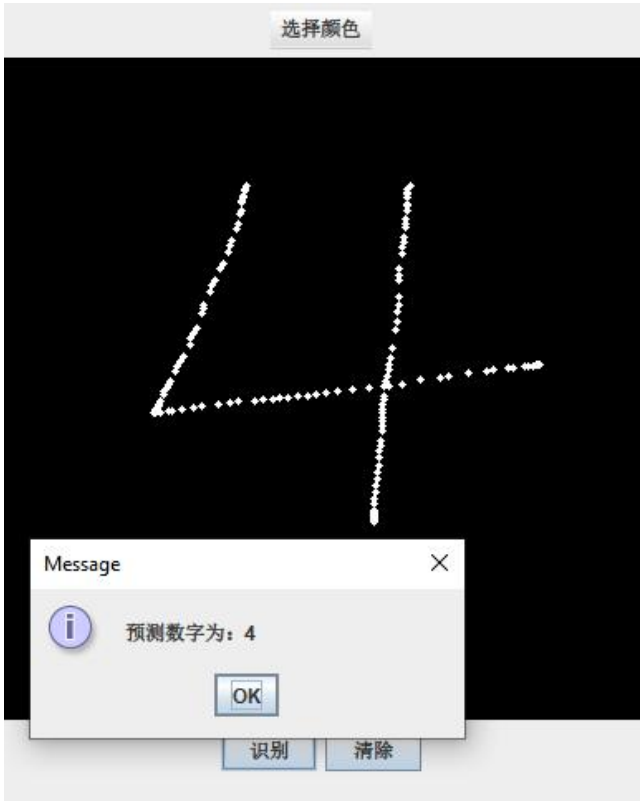

图 46 测试手写数字 4

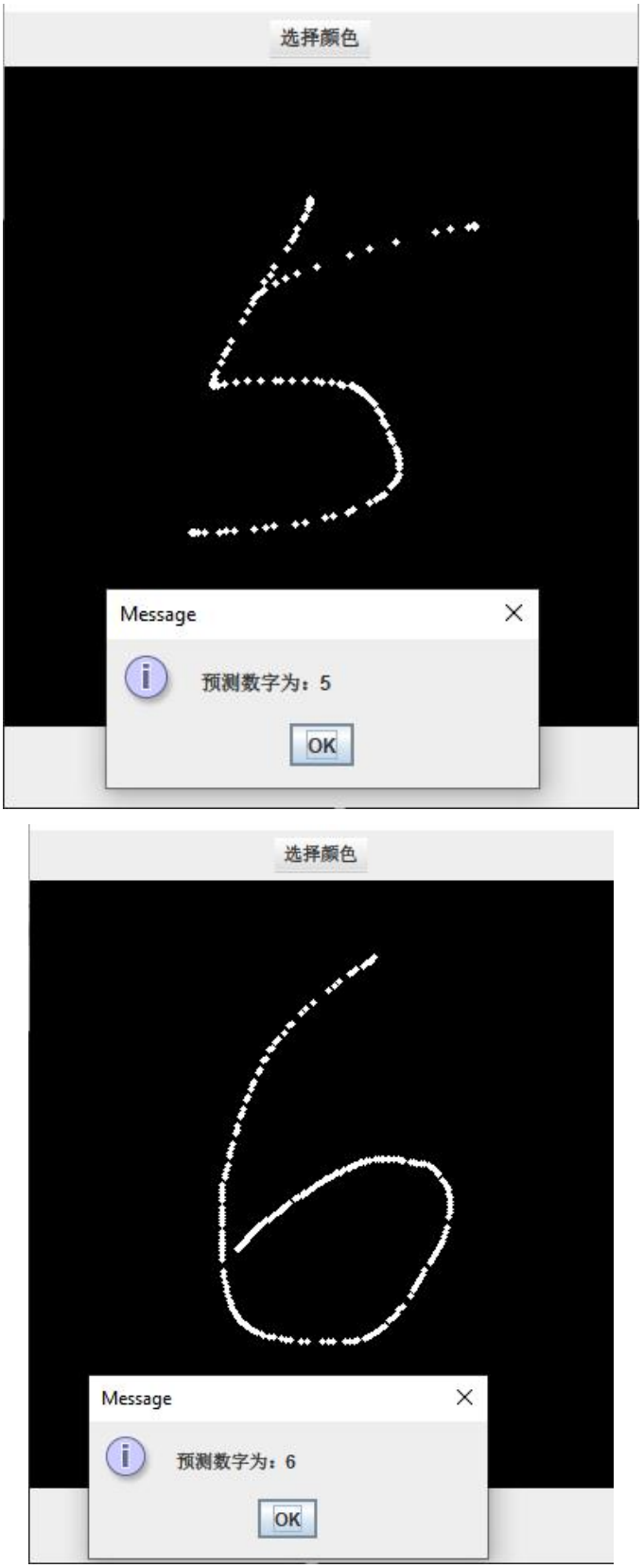

图 47 测试手写数字 6

图 48 测试手写数字 7

图 49 测试手写数字 8

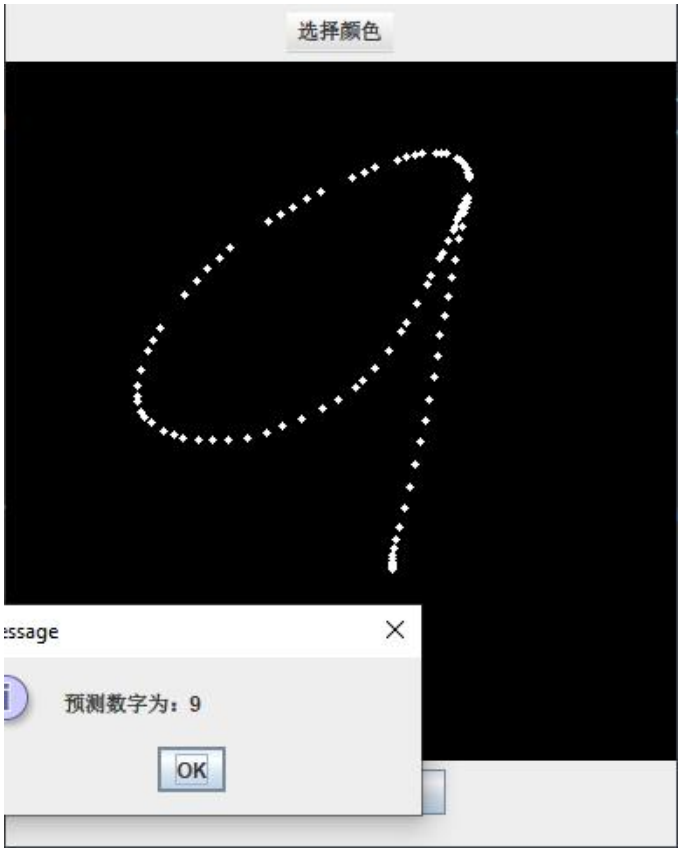

图 50 测试手写数字 9

5.2. 结果分析

经过测试，所实现的神经网络算法基本可以满足预定的需求，基本可以识别 0~9 个手写数字。

5.3. 总结

通过对本次课程设计，进一步的了解了目前比较热门的神经网络的内部运行机制，掌握了 BP 算法的原理与公式推导，知道了如何训练一个有效的神经网络，并学会了手动编写一个神经网络，同时使用该神经网络实现了一个手写数字识别的模型。

参考文献：

[1] wu740027007.BP 神经网络原理和算法推导流程（吴恩达机器学习）[EB/OL].https://blog.csdn.net/wu740027007/article/details/100884238,2019-09-16.

[2] INTERMT.一文彻底搞懂 BP 算法：原理推导+数据演示+项目实战[EB/OL].https://github.com/INTERMT/BP-Algorithm,2019-7-2.

[3] 折射.神经网络，BP 算法的理解与推导[EB/OL].https://zhuanlan.zhihu.com/p/45190898,2018-9-23.

[4] 折射.神经网络，BP 算法，计算图模型，代码实现跟详解[EB/OL].https://zhuanlan.zhihu.com/p/60007345,2019-4-1.

[5] 学习-快乐.BP 神经网络反向传播之计算过程分解（详细版）[EB/OL].https://www.cnblogs.com/liuhuacai/p/11973036.html,2019-12-02.

[6] ladykaka007.手把手教大家实现吴恩达深度学习作业第二周 06-反向传播推导[EB/OL].https://www.bilibili.com/video/BV1QW411g718?from=search&seid=5196456272229839194,2018-03-2