

信号与系统工程设计

目 录

一 . 设计目的与要求	3
二 .设计原理	3
2.1 bp 神经网络的原理与构建	3
2.2 卷积神经网络的原理	9
三 .设计内容	10
3.1 bp 神经网络的实现	10
3.2 可视化程序	13
四 . 结果分析	16
4.1 调整超参数	16
4.2 数据增强	17
4.3 替换激活函数	20
五 . 总结与体会	20
六 . 小组分工	20

1 设计目的与要求

通过神经网络的方法，用于识别 mnist 手写数据，并分析各种超参数对识别效果的影响，自己手写替换所有库函数，尝试通过更换激活函数，进行数据增强来提高识别效果，尝试使用卷积神经网络进行识别。

2 设计原理

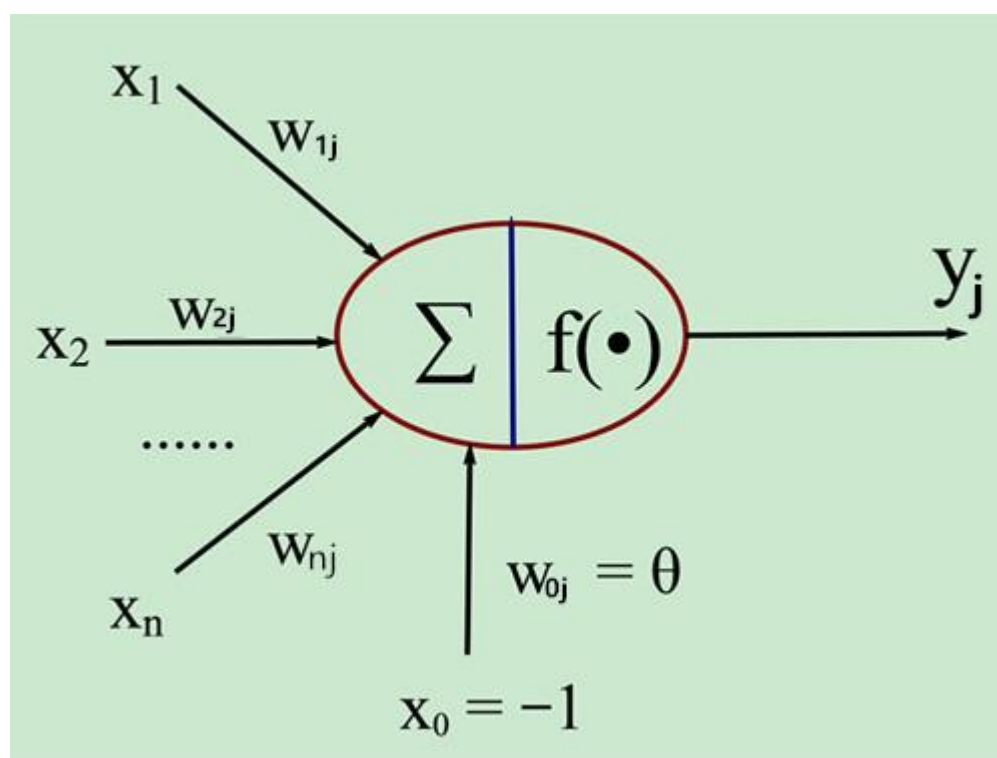
2.1: bp 神经网络的原理与构建:

BP (Back Propagation) 神经网络构建

1. 理论模型

1) 神经元 M-P 模型

按照生物神经元，我们建立 M-P 模型。为了使得建模更加简单，以便于进行形式化表达，我们忽略时间整合作用、不应期等复杂因素，并把神经元的突触时延和强度当成常数。下图就是一个 M-P 模型的示意图。



结合 M-P 模型示意图来看，对于某一个神经元 j ，它可能接受同时接受了许多个输入信号，用 x_i 表示。由于生物神经元具有不同的突触性质和突触强度，所以对神经元的影响不同，我们用权值 w_{ij} 来表示，其正负模拟了生物神经元中突出的兴奋和抑制，其大小则代表了突出的不同连接强度。 θ_j 表示为一个阈值 (threshold)，或称为偏置 (bias)。由于

累加性, 我们对全部输入信号进行累加整合, 相当于生物神经元中的膜电位, 其值就为:

$$net'_j(t) = \sum_{i=1}^n \omega_{ij} \chi_i(t) - \theta_j$$

神经元激活与否取决于某一阈值电平, 即只有当其输入总和超过阈值 θ_j 时, 神经元才被激活而发放脉冲, 否则神经元不会发生输出信号。整个过程可以用下面这个函数来表示:

$$y_j = f(net_j)$$

y_j 表示神经元 j 的输出, 函数 f 称为激活函数 (Activation Function)或转移函数 (Transfer Function), $net'_j(t)$ 称为净激活(net activation)。若将阈值看成是神经元 j 的一个输入 x_0 的权重 w_{0j} , 则上面的式子可以简化为:

$$net'_j(t) = \sum_{i=0}^n \omega_{ij} \chi_i(t)$$

$$y_j = f(net_j)$$

若用 X 表示输入向量, 用 W 表示权重向量, 即:

$$X = [\chi_0, \chi_1, \dots, \chi_n]$$

$$M = \begin{bmatrix} \omega_{0j} \\ \omega_{1j} \\ \vdots \\ \omega_{nj} \end{bmatrix}$$

则神经元的输出可以表示为向量相乘的形式:

$$net_j = XW$$

$$y_j = f(net_j) = f(XW)$$

若神经元的净激活为正，称该神经元处于激活状态或兴奋状态，若净激活为负，则称神经元处于抑制状态。

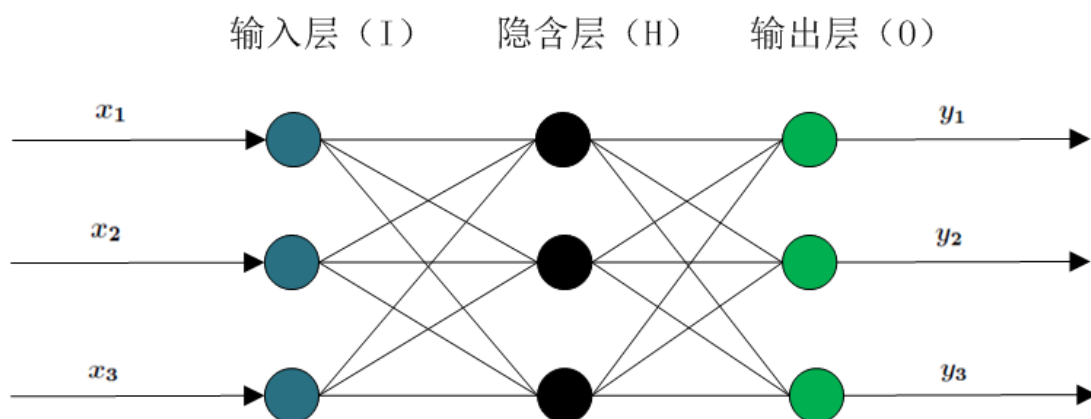
由此我们可以得到总结出 M-P 模型的 6 个特点：

- <1> 每个神经元都是一个多输入单输出的信息处理单元；
- <2> 神经元输入分兴奋性输入和抑制性输入两种类型；
- <3> 神经元具有空间整合特性和阈值特性；
- <4> 神经元输入与输出间有固定的时滞，主要取决于突触延搁；
- <5> 忽略时间整合作用和不应期；
- <6> 神经元本身是非时变的，即其突触时延和突触强度均为常数。

这种“阈值加权和”的神经元模型称为 M-P 模型（McCulloch-Pitts Model），也称为神经网络的一个处理单元（PE, Processing Element）。

2) BP 神经网络

BP（Back Propagation）神经网络分为两个过程：工作信号正向传递子过程与误差信号反向传递子过程。BP(Back Propagation)神经网络的学习过程由信号的正向传播与误差的反向传播两个过程组成。正向传播时，输入样本从输入层传入，经隐含层逐层处理后，传向输出层。若输出层的实际输出与期望输出不符，则转向误差的反向传播阶段。误差的反向传播是将输出误差以某种形式通过隐含层向输入层逐层反传，并将误差分摊给各层的所有单元，从而获得各层单元的误差信号，此误差信号即作为修正各单元权值的依据。在 BP 神经网络中，单个样本有 m 个输入，有 n 个输出，在输入层和输出层之间通常还有若干个隐含层。一个三层的 BP 网络就可以完成任意的 m 维到 n 维的映射。即这三层分别是输入层（I），隐含层（H），输出层（O），如下图所示。



正向传递子过程：现在设节点 i 和节点 j 之间的权值为 w_{ij} ，节点 j 的阈值为 b_j ，每个节点的输出值为 x_j ，而每个节点的输出值是根据上层所有节点的输出值、当前节点与上一

层所有节点的权值和当前节点的阈值还有激活函数来实现的。具体计算方法如下：

$$S_j = \sum_{i=0}^{m-1} w_{ij}x_i + b_j$$

$$x_j = f(S_j)$$

其中 f 为激活函数，一般选取 S 型函数或者线性函数。正向传递的过程比较简单，按照上述公式计算即可。在 BP 神经网络中，输入层节点没有阈值。

反向传递子过程：在 BP 神经网络中，误差信号反向传递子过程比较复杂，它是基于 Widrow-Hoff 学习规则的。假设输出层的所有结果为 d_j ，误差函数如下

$$E(w, b) = \frac{1}{2} \sum_{j=0}^{n-1} (d_j - y_j)^2$$

而 BP 神经网络的主要目的是反复修正权值和阈值，使得误差函数值达到最小。Widrow-Hoff 学习规则是通过沿着相对误差平方和的最速下降方向，连续调整网络的权值和阈值，根据梯度下降法，权值矢量的修正正比于当前位置上 $E(w, b)$ 的梯度，对于第 j 个输出节点有

$$\Delta w(i, j) = -\eta \frac{\partial E(w, b)}{\partial w(i, j)}$$

对激活函数求导，得到

$$\begin{aligned} f'(x) &= \frac{Ae^{-\frac{x}{B}}}{B(1 + e^{-\frac{x}{B}})^2} \\ &= \frac{1}{AB} \cdot \frac{A}{1 + e^{-\frac{x}{B}}} \cdot \left(A - \frac{A}{1 + e^{-\frac{x}{B}}} \right) \\ &= \frac{f(x)[A - f(x)]}{AB} \end{aligned}$$

那么接下来针对 w_{ij} 有

$$\begin{aligned}
\frac{\partial E(w, b)}{\partial w_{ij}} &= \frac{1}{\partial w_{ij}} \cdot \frac{1}{2} \sum_{j=0}^{n-1} (d_j - y_j)^2 \\
&= (d_j - y_j) \cdot \frac{\partial d_j}{\partial w_{ij}} \\
&= (d_j - y_j) \cdot f'(S_j) \cdot \frac{\partial S_j}{\partial w_{ij}} \\
&= (d_j - y_j) \cdot \frac{f(S_j) [A - f(S_j)]}{AB} \cdot \frac{\partial S_j}{\partial w_{ij}} \\
&= (d_j - y_j) \cdot \frac{f(S_j) [A - f(S_j)]}{AB} \cdot x_i \\
&= \delta_{ij} \cdot x_i
\end{aligned}$$

其中有

$$\delta_{ij} = (d_j - y_j) \cdot \frac{f(S_j) [A - f(S_j)]}{AB}$$

同样对于 b_j 有

$$\frac{\partial E(w, b)}{\partial b_j} = \delta_{ij}$$

这就是著名的 δ 学习规则，通过改变神经元之间的连接权值来减少系统实际输出和期望输出的误差，这个规则又叫做 Widrow-Hoff 学习规则或者纠错学习规则。上面是对隐含层和输出层之间的权值和输出层的阈值计算调整量，而针对输入层和隐含层的阈值调整量的计算更为复杂。假设 w_{ki} 是输入层第 k 个节点和隐含层第 i 个节点之间的权值，那么有

$$\begin{aligned}
\frac{\partial E(w, b)}{\partial w_{ki}} &= \frac{1}{\partial w_{ki}} \cdot \frac{1}{2} \sum_{j=0}^{n-1} (d_j - y_j)^2 \\
&= \sum_{j=0}^{n-1} (d_j - y_j) \cdot f'(S_j) \cdot \frac{\partial S_j}{\partial w_{ki}} \\
&= \sum_{j=0}^{n-1} (d_j - y_j) \cdot f'(S_j) \cdot \frac{\partial S_j}{\partial x_i} \cdot \frac{\partial x_i}{\partial S_i} \cdot \frac{\partial S_i}{\partial w_{ki}} \\
&= \sum_{j=0}^{n-1} \delta_{ij} \cdot w_{ij} \cdot \frac{f(S_i) [A - f(S_i)]}{AB} \cdot x_k \\
&= x_k \cdot \sum_{j=0}^{n-1} \delta_{ij} \cdot w_{ij} \cdot \frac{f(S_i) [A - f(S_i)]}{AB} \\
&= \delta_{ki} \cdot x_k
\end{aligned}$$

其中有

$$\delta_{ki} = \sum_{j=0}^{n-1} \delta_{ij} \cdot w_{ij} \cdot \frac{f(S_i) [A - f(S_i)]}{AB}$$

根据梯度下降法，那么对于隐含层和输出层之间的权值和阈值调整如下

$$w_{ij} = w_{ij} - \eta_1 \cdot \frac{\partial E(w, b)}{\partial w_{ij}} = w_{ij} - \eta_1 \cdot \delta_{ij} \cdot x_i$$

$$b_j = b_j - \eta_2 \cdot \frac{\partial E(w, b)}{\partial b_j} = b_j - \eta_2 \cdot \delta_{ij}$$

而对于输入层和隐含层之间的权值和阈值调整同样有

$$w_{ki} = w_{ki} - \eta_1 \cdot \frac{\partial E(w, b)}{\partial w_{ki}} = w_{ki} - \eta_1 \cdot \delta_{ki} \cdot x_k$$

$$b_i = b_i - \eta_2 \cdot \frac{\partial E(w, b)}{\partial b_i} = b_i - \eta_2 \cdot \delta_{ki}$$

如上所述，在实际过程中，BP 神经网络的训练流程可大致分为 5 步：初始化网络的突触权值和阈值矩阵；训练样本的呈现；前向传播计算；误差反向传播计算并更新权值；迭代，用

新的样本进行前向传播与反向传播，直至满足停止准则。

2.2 卷积神经网络的原理

卷积神经网络原理：

首先使用过滤器，将 28×28 的输入层的数据处理并建立卷积层，用 x_{ij} 表示所读入的图像的 i 行 j 列位置的像素数据，用 $w_{ij}^{F_k}$ 表示过滤器 k 的第 i 行第 j 列的权重，用 b^{F_k} 表示过滤器 k 的偏置，本程序中过滤器层数为 30，过滤器边长为 25，则卷积层中第 i 行第 j 列的数值为：

$$z_{ij}^{F_k} = w_{11}^{F_k} x_{ij} + w_{12}^{F_k} x_{ij+1} + w_{13}^{F_k} x_{ij+2} + \dots + w_{2525}^{F_k} x_{i+25j+25} + b^{F_k}$$

设激活函数为 $a(z)$ ，则神经元的输出即卷积层第 i 行第 j 列的值为 $a_{ij}^{F_k} = a(z_{ij}^{F_k})$ 。

然后将卷积层每 2×2 个神经元压缩为一个神经元，形成池化层，这里使用最大化池，从 4 个神经元的输出中选出最大值作为代表，即：

$$a_{ij}^{P_k} = z_{ij}^{P_k} = \text{Max}(a_{2i-1\ 2j-1}^{F_k}, a_{2i-1\ 2j}^{F_k}, a_{2i\ 2j-1}^{F_k}, a_{2i\ 2j}^{F_k})$$

池化层的边长为 $(28-25+1)/2=2$ 。

输出层有 10 个神经元，分别代表十个数。令 $w_{k-ij}^{O_n}$ 表示从池化层第 k 个子层第 i 行第 j 列的神经元指向输出层的第 n 个神经元的权重， b_n^O 为输出层第 n 个单元的神元偏置，则输出层第 n 个神经元的加权输入为：

$$z_n^O = w_{n-11}^{O_n} + w_{n-12}^{O_n} + w_{n-21}^{O_n} + w_{n-22}^{O_n}$$

输出层第 n 个神经元的输出值为 $a_n^O = a(z_n^O)$ ，代价函数：

$$C = [(t_1 - a_1^O)^2 + (t_2 - a_2^O)^2 + \dots + (t_{10} - a_{10}^O)^2]/2$$

其中 t_n 为输出层第 n 个神经元的输出期望值，为 1 或 0。

卷积神经网络误差反向传播法：梯度下降法基本式为：

$$(\Delta w_{11}^{F1}, \dots, \Delta w_{1-11}^{O1}, \dots, \Delta b^{F1}, \dots, \Delta b_1^O, \dots) = -\eta \left(\frac{\partial C}{\partial w_{11}^{F1}}, \dots, \frac{\partial C}{\partial w_{1-11}^{O1}}, \dots, \frac{\partial C}{\partial b^{F1}}, \dots, \frac{\partial C}{\partial b_1^O}, \dots \right)$$

本程序中学习率取的 0.08。定义神经元误差 $\delta_{ij}^{Fk} = \frac{\partial C}{\partial z_{ij}^{Fk}}$ ， $\delta_n^O = \frac{\partial C}{\partial z_n^O}$ ，根据函数求导链规则有输出神经元的梯度分量：

$$\frac{\partial C}{\partial w_{k-ij}^{O_n}} = \delta_n^O a_{ij}^{P_k}, \quad \frac{\partial C}{\partial b_n^O} = \delta_n^O$$

卷积层神经元的梯度分量：

$$\frac{\partial C}{\partial w_{ij}^{Fk}} = \delta_{11}^{Fk} x_{ij} + \delta_{12}^{Fk} x_{ij+1} + \dots + \delta_{44}^{Fk} x_{i+3j+3}, \quad \frac{\partial C}{\partial b^{Fk}} = \delta_{11}^{Fk} + \delta_{12}^{Fk} + \dots + \delta_{44}^{Fk}$$

因此，计算神经元误差即可得到代价函数的梯度。计算输出层单元神经误差：

$$\delta_n^O = (a_n^O - t_n) a'(z_n^O)$$

其中 $a(x) = \frac{1}{1+e^x}$ 时有 $a'(x) = [1 - a(x)]a(x)$ ；卷积层神经元误差递推式：

$$\delta_{ij}^{Fk} = (\delta_1^O w_{k-ij}^{O1} + \dots + \delta_{10}^O w_{k-ij}^{O10}) \times a'(Z_{ij}^{Fk}) \times \begin{cases} 1 & \text{当 } a_{ij}^{Fk} \text{ 在区块中最大时为 1, 否则为 0} \end{cases}$$

根据已有数据进行训练并进行测试，不断调整参数，最后对测试集进行测试，准确率可达到 96.5%。

3 设计内容

3.1 bp 神经网络的实现

1) BP 神经网络基本参数

```
const double learning_rate = 0.08; // 学习率
const int epoch = 15; // 学习轮次
const int nh = 30; // 隐藏单元数量（单隐藏层）
double w1[784][nh]; // 输入层到隐藏层的权重矩阵
double w2[nh][10]; // 隐藏层到输出层的权重矩阵
double bias1[nh]; // 隐藏层的偏置
double bias2[10]; // 输出层的偏置
```

其中输入层节点数为 784，由于手写数字识别的结果只能为 0-9 这十种，故输出层节点数为 10。而对于隐含层节点数，有如下经验公式确定：

$$h = \sqrt{m + n} + a$$

其中 h 为隐含层节点数目， m 为输入层节点数目， n 为输出层节点数目， a 为

$1 \sim 10$ 之间的调节常数。在本次项目中 $\sqrt{m+n} \approx 28.2$ ，故取隐含层节点数为 30。

2) 为权重矩阵和偏置向量随机赋初值

```
void init_parameters() {
    for (int i = 0; i < 784; i++) {
        for (int j = 0; j < nh; j++) w1[i][j] = rand() / (10 * (double)RAND_MAX) - 0.05;
    }

    for (int i = 0; i < nh; i++) {
        for (int j = 0; j < 10; j++) w2[i][j] = rand() / (10 * (double)RAND_MAX) - 0.05;
    }

    for (int i = 0; i < nh; i++) bias1[i] = rand() / (10 * (double)RAND_MAX) - 0.1;
    for (int i = 0; i < 10; i++) bias2[i] = rand() / (10 * (double)RAND_MAX) - 0.1;
}
```

利用 RAND_MAX 函数进行随机初始化，目的是使对称失效。否则的话，所有对称结点的权重都一致，也就无法区分并学习了

3) 激活函数的定义

```

// 激活函数 sigmoid, 可以把 x 映射到 0 ~ 1 之间
//  $s(x) = 1 / (1 + e^{-x})$ 
//  $s'(x) = s(x) * (1 - s(x))$ 
double sigmoid(double x) {
    return 1 / (1 + exp(-x));
}

```

```

// 激活函数 Leaky_Relu
double Leaky_Relu(double x) {
    return (x > 0 ? x : 0.01 * x);
}

```

```

// 激活函数 Relu
double Relu(double x) {
    return (x > 0 ? x : 0);
}

```

提供 sigmoid, Leaky_Relu, Relu 三种激活函数可供选择。

4) 正向传播——由输入图像得到隐含层输出、由隐含层输出得到网络最终输出

```

// 通过图像的输入得到隐藏层的输出
vector<double> get_hidden_out(vector<double>& image) {
    vector<double> hidden_out(nh);
    for (int i = 0; i < nh; i++) {          // 对于每一个隐藏单元
        double sum = 0.0;
        for (int j = 0; j < 784; j++) {
            sum += image[j] * w1[j][i];
        }
        hidden_out[i] = sigmoid(sum + bias1[i]);
    }
    return hidden_out;
}

// 通过隐藏层的输出得到网络最终的输出
vector<double> get_z(vector<double>& hidden_out) {
    vector<double> z(10);
    for (int i = 0; i < 10; i++) {          // 对于每一个输出单元
        double sum = 0.0;
        for (int j = 0; j < nh; j++) {
            sum += hidden_out[j] * w2[j][i];
        }
        z[i] = sigmoid(sum + bias2[i]);
    }
    return z;
}

```

5) 损失函数

// 计算损失函数 (1/2 均方误差)

```
double get_loss(vector<double>& z, double label) {
    double loss = 0;
    int true_label = (int)label;
    for (int i = 0; i < 10; i++) {
        if (i != true_label) loss += z[i] * z[i];
        else loss += (1 - z[i]) * (1 - z[i]);
    }
    return loss / 2;
}
```

6) 训练 BP 神经网络

```
void train_and_test() {
    for (e = 1; e <= epoch; e++) {
        for (int im = 0; im < train_images.size(); im++) {
            double grad_w1[784][nh];
            double grad_w2[nh][10];
            double grad_bias1[nh];
            double grad_bias2[10];

            vector<double> hidden_out = get_hidden_out(train_images[im]);
            vector<double> z = get_z(hidden_out);
            double loss = get_loss(z, train_labels[im]);
            int true_label = (int)train_labels[im];

            // -----计算梯度-----

            for (int j = 0; j < nh; j++) {
                double sum = 0.0;
                for (int k = 0; k < 10; k++) {
                    double labelk = (k == true_label) ? 1.0 : 0.0;
                    sum += (z[k] - labelk) * z[k] * (1 - z[k]) * w2[j][k] * hidden_out[j] * (1 - hidden_out[j]);
                }
                grad_bias1[j] = sum;
            }

            for (int i = 0; i < 784; i++) {
                for (int j = 0; j < nh; j++) {
                    double sum = 0.0;
                    grad_w1[i][j] = grad_bias1[j] * train_images[im][i];
                }
            }

            for (int k = 0; k < 10; k++) {
                double labelk = (k == true_label) ? 1.0 : 0.0;
                grad_bias2[k] = (z[k] - labelk) * z[k] * (1 - z[k]);
            }

            for (int j = 0; j < nh; j++) {
                for (int k = 0; k < 10; k++) {
                    double labelk = (k == true_label) ? 1.0 : 0.0;
                    grad_w2[j][k] = grad_bias2[k] * hidden_out[j];
                }
            }
        }
    }
}
```

```
// -----更新权与偏置-----

for (int i = 0; i < 784; i++) {
    for (int j = 0; j < nh; j++) {
        w1[i][j] -= learning_rate * grad_w1[i][j];
    }
}

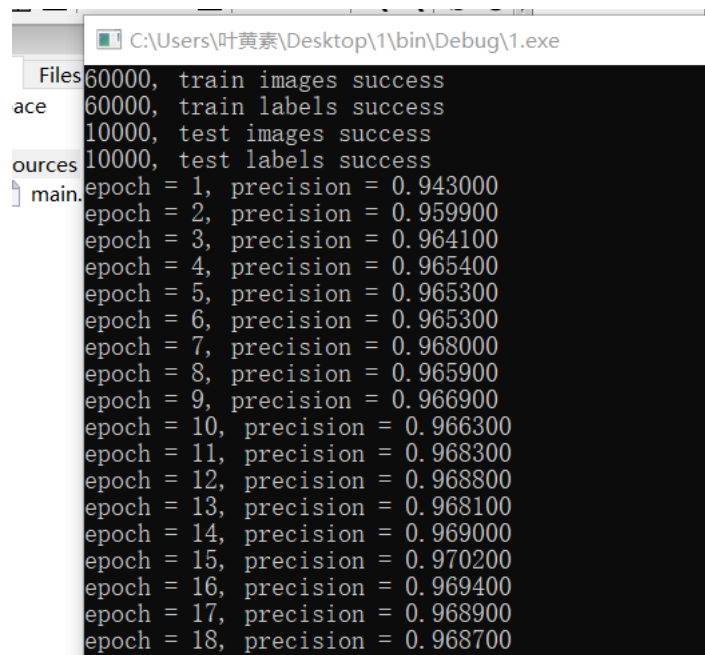
for (int i = 0; i < nh; i++) {
    for (int j = 0; j < 10; j++) {
        w2[i][j] -= learning_rate * grad_w2[i][j];
    }
}

for (int i = 0; i < nh; i++) {
    bias1[i] -= learning_rate * grad_bias1[i];
}

for (int i = 0; i < 10; i++) {
    bias2[i] -= learning_rate * grad_bias2[i];
}
```

3.2 可视化程序

搭建卷积神经网络以及卷积网络参数的调整，并将调整的参数用文件储存起来。



```
C:\Users\叶黄素\Desktop\1\bin\Debug\1.exe
Files 60000, train images success
ace 60000, train labels success
10000, test images success
ources 10000, test labels success
main: epoch = 1, precision = 0.943000
epoch = 2, precision = 0.959900
epoch = 3, precision = 0.964100
epoch = 4, precision = 0.965400
epoch = 5, precision = 0.965300
epoch = 6, precision = 0.965300
epoch = 7, precision = 0.968000
epoch = 8, precision = 0.965900
epoch = 9, precision = 0.966900
epoch = 10, precision = 0.966300
epoch = 11, precision = 0.968300
epoch = 12, precision = 0.968800
epoch = 13, precision = 0.968100
epoch = 14, precision = 0.969000
epoch = 15, precision = 0.970200
epoch = 16, precision = 0.969400
epoch = 17, precision = 0.968900
epoch = 18, precision = 0.968700
```

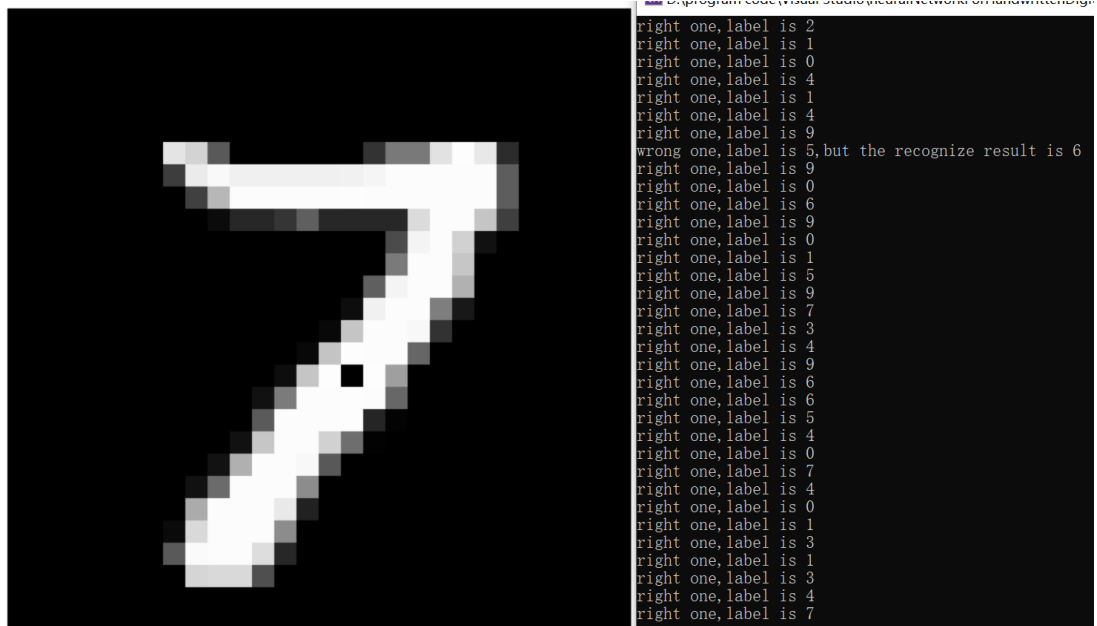
上图为开始卷积层设 50 时的训练过程，训练后发现利用训练好的参数，识别过程花的时间太长，因此进行了简化，将卷积层降到了 30，训练后的准确率同样也与层数为 50 时差不了多少。可以看到准确率并不是一味的增加，而是先增后减，尝试了在一轮轮学习过程中一旦准确率降低就将学习率减半，但结果准确率仍是不停跳变，也就作罢。最后在卷积网络层数为 30 时，训练过程中准确率最高的一次的参数保存下来。

使用 esayX 创建图形化界面，可以用鼠标在画板上写数字，原理是鼠标按键后，隔一段

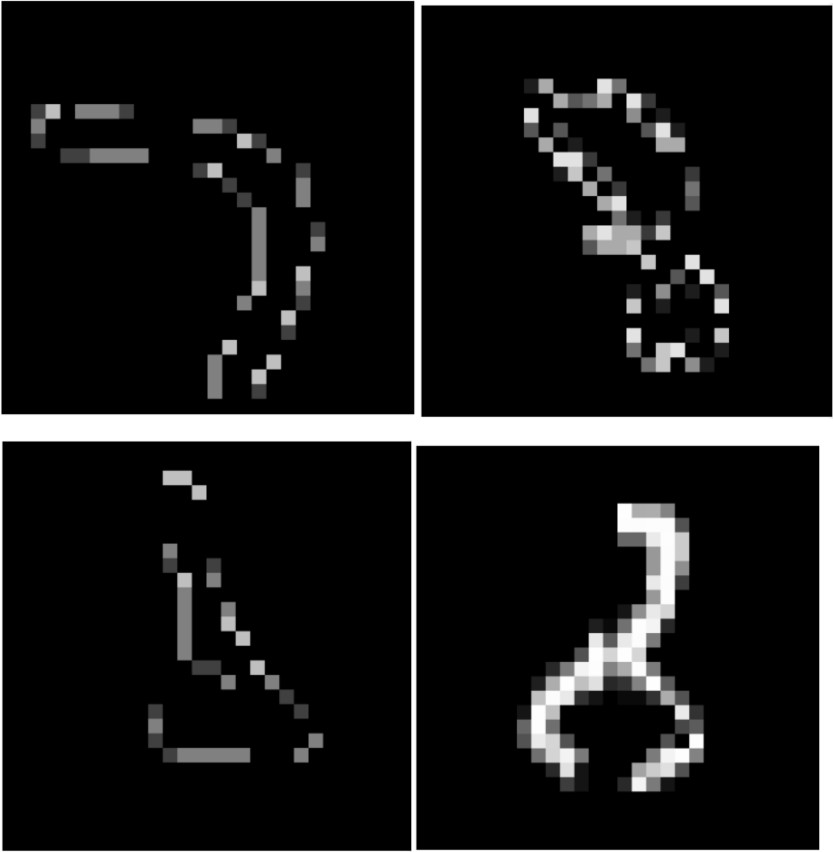


A screenshot of a calculator interface. The display shows the number '2'. Below the display, there are several buttons labeled in Chinese: '识别1' (Identify 1), '识别2' (Identify 2), '清除' (Clear), '处理' (Process), '测试1' (Test 1), and '测试2' (Test 2). The '识别2' button is highlighted with a yellow border.

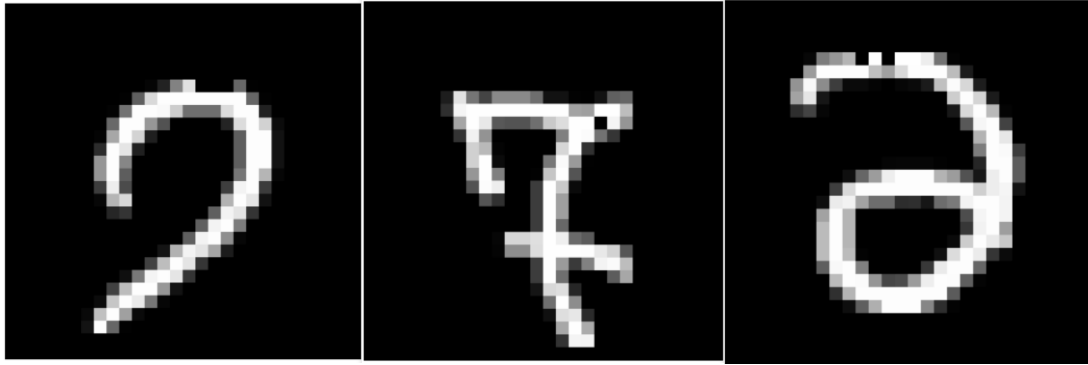
在测试过程中可以依次查看测试数据，如图



期间也能看到疑似数据中出现的问题，例如数字上的黑斑，笔画的缺失，只剩下边框等问题，如下图：



查看神经网络识别错误的图像能发现很多奇奇怪怪的图形，例如下面的数字的标签分别是 7，7，2，分别被识别成了 9，8，0。



4 结果分析

4.1 调整超参数

需要调整的超参数大致是学习率和学习轮数，经过控制变量之后，训练了三次寻找规律。

1. 学习率：0.08；学习轮数：10

```
60000, train images success
60000, train labels success
10000, test images success
10000, test labels success
epoch = 1, precision = 0.924900
epoch = 2, precision = 0.940300
epoch = 3, precision = 0.945800
epoch = 4, precision = 0.948200
epoch = 5, precision = 0.951500
epoch = 6, precision = 0.954200
epoch = 7, precision = 0.956100
epoch = 8, precision = 0.956600
epoch = 9, precision = 0.957900
epoch = 10, precision = 0.958800
sh: 1: precision not found
```

可以看出随着学习轮数上升，预测置信度还在上升，所以增加学习轮数

2. 学习率：0.08；学习轮数：15

```
60000, train images success
60000, train labels success
10000, test images success
10000, test labels success
epoch = 1, precision = 0.924900
epoch = 2, precision = 0.940300
epoch = 3, precision = 0.945800
epoch = 4, precision = 0.948200
epoch = 5, precision = 0.951500
epoch = 6, precision = 0.954200
epoch = 7, precision = 0.956100
epoch = 8, precision = 0.956600
epoch = 9, precision = 0.957900
epoch = 10, precision = 0.958800
epoch = 11, precision = 0.959400
epoch = 12, precision = 0.960200
epoch = 13, precision = 0.960200
epoch = 14, precision = 0.960800
epoch = 15, precision = 0.961400
```


可以看出学习轮数增加到 15 的时候，置信度已经到了 0.9614，达到了一个比较高的水平，且增加速度已经变慢，再增加学习轮数训练时间会比较长，加上可能会造成过拟合现象，遂认为 15 轮是一个理想的学习轮数。

3. 学习率：0.2；学习轮数：15

```
epoch = 1, precision = 0.927700
epoch = 2, precision = 0.944000
epoch = 3, precision = 0.949400
epoch = 4, precision = 0.950300
epoch = 5, precision = 0.953300
epoch = 6, precision = 0.953100
epoch = 7, precision = 0.953400
epoch = 8, precision = 0.954000
epoch = 9, precision = 0.955200
epoch = 10, precision = 0.957900
epoch = 11, precision = 0.958900
epoch = 12, precision = 0.959000
epoch = 13, precision = 0.959400
epoch = 14, precision = 0.959800
epoch = 15, precision = 0.960300
```

增加学习率，会使得收敛速度变快，但是模型精度会下降，得不偿失，由上述分析，最后选择学习率：0.08；学习轮数：15。

4.2 数据增强

因为 minst 数据集只有 6 万张图片作为训练集，所以我想进行数据增强，将训练集进行扩充，希望达到提高识别率的作用

代码中先将训练集由二进制文件数据存在数组里，然后通过随机数，以每种数据增强 30% 的几率进行，这里尝试了 6 种数据增强，顺时针 90 度，逆时针 90 度，旋转 180 度，左右翻转，上下翻转，平移，均是直接对数组进行操作，操作完成之后的数据存在一个新的数组里面，然后再将原训练集和增强后的数据集拼接起来形成一个新的训练集，对模型进行训练。

```

void data_strengthen()
{
    int i = 0;
    int num = 0;
    srand((int)time(0));
    for(i=0; i<60000; i++)
    {
        // if((rand() % (100) / 100.0)<0.3)
        // {
        //     RotationLeft90(train_images[i]);
        //     strengthen_labels.push_back(train_labels[i]);
        //     num++;
        // }

        // if((rand() % (100) / 100.0)<0.3)
        // {
        //     RotationRight90(train_images[i]);
        //     strengthen_labels.push_back(train_labels[i]);
        //     num++;
        // }

        // if((rand() % (100) / 100.0)<0.3)
        // {
        //     RotationDown(train_images[i]);
        //     strengthen_labels.push_back(train_labels[i]);
        //     num++;
        // }

        if((rand() % (100) / 100.0)<0.3)
        {
            geometryTransl(train_images[i]);
            strengthen_labels.push_back(train_labels[i]);
            num++;
        }
    }
}

```

进行了三次训练:

1. 学习率: 0.08; 学习轮数: 15, 翻转平移旋转

```

60000, train images success
60000, train labels success
10000, test images success
10000, test labels success
72245
epoch = 1, precision = 0.306900
epoch = 2, precision = 0.340200
epoch = 3, precision = 0.361000
epoch = 4, precision = 0.381800
epoch = 5, precision = 0.400200
epoch = 6, precision = 0.412900
epoch = 7, precision = 0.420000
epoch = 8, precision = 0.430500
epoch = 9, precision = 0.437200
epoch = 10, precision = 0.445800
epoch = 11, precision = 0.450400
epoch = 12, precision = 0.456000
epoch = 13, precision = 0.457300
epoch = 14, precision = 0.464900
epoch = 15, precision = 0.471500

```

2. 学习率: 0.08; 学习轮数: 15, 翻转平移

```
60000, train images success
60000, train labels success
10000, test images success
10000, test labels success
35918
epoch = 1, precision = 0.552500
epoch = 2, precision = 0.426700
epoch = 3, precision = 0.447000
epoch = 4, precision = 0.480500
epoch = 5, precision = 0.518900
epoch = 6, precision = 0.536000
epoch = 7, precision = 0.546000
epoch = 8, precision = 0.562700
epoch = 9, precision = 0.573100
epoch = 10, precision = 0.583700
epoch = 11, precision = 0.591500
epoch = 12, precision = 0.596100
epoch = 13, precision = 0.603200
epoch = 14, precision = 0.607200
epoch = 15, precision = 0.609600
sh: 1: pause: not found
```

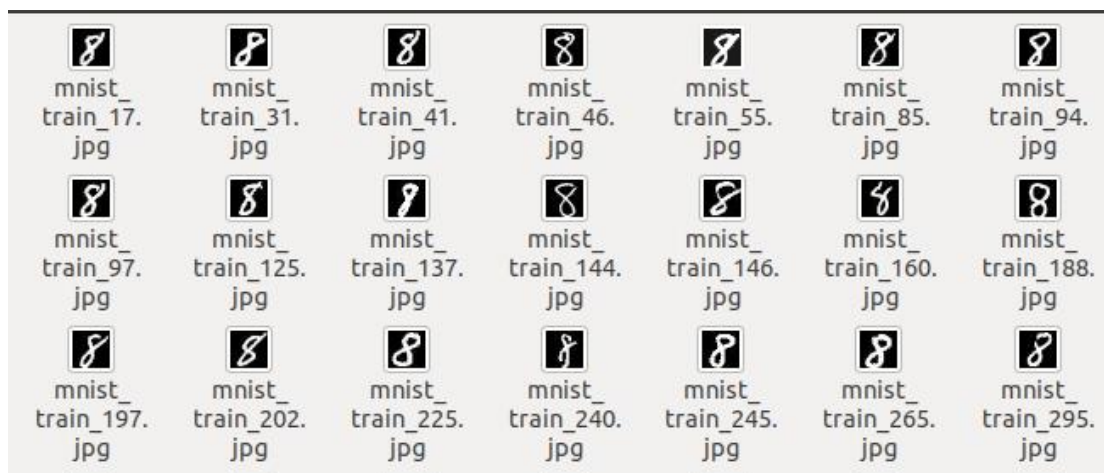
3. 学习率: 0.08; 学习轮数: 15, 平移

```
60000, train images success
60000, train labels success
10000, test images success
10000, test labels success
17951
epoch = 1, precision = 0.886800
epoch = 2, precision = 0.917800
epoch = 3, precision = 0.935300
epoch = 4, precision = 0.941600
epoch = 5, precision = 0.945900
epoch = 6, precision = 0.951300
epoch = 7, precision = 0.946300
epoch = 8, precision = 0.951500
epoch = 9, precision = 0.954100
epoch = 10, precision = 0.953900
epoch = 11, precision = 0.955800
epoch = 12, precision = 0.957500
epoch = 13, precision = 0.958200
epoch = 14, precision = 0.955800
epoch = 15, precision = 0.958400
```

对上述三次训练分析可见, 旋转翻转两个操作对模型精度伤害极大, 经过分析应该是 minst 数据集全由数字组成, 旋转翻转可能会使得数据特征减弱。

平移也对模型精度有一点减弱, 对 minst 数据集由二进制格式文件通过 python 转为 jpg 文件可以发现, 图片像素是 28*28, 而大部分数字都填满了整张图片, 左右

上下平移一个像素都有可能会使得数字移出图片，导致数据集出现错误，遂最后放弃了数据增强



4.3 替换激活函数

之前用的激活函数均是 sigmoid，尝试了更换激活函数为 Relu 和 Leaky_Relu，左图为 Relu，右图为 Leaky_Relu

rosetta@rosetta-OMEN-by-HP-Laptop-15-dc1xxx: ~/mnist/build	rosetta@rosetta-OMEN-by-HP-Laptop-15-dc1xxx: ~/mnist/build
<pre>[100%] Built target mnist rosetta@rosetta-OMEN-by-HP-Laptop-15-dc1xxx:~/mnist/build\$./mnist 50000, train images success 50000, train labels success 10000, test images success 10000, test labels success 35889 epoch = 1, precision = 0.896400 epoch = 2, precision = 0.925200 epoch = 3, precision = 0.935600 epoch = 4, precision = 0.938700 epoch = 5, precision = 0.941700 epoch = 6, precision = 0.943600 epoch = 7, precision = 0.945000 epoch = 8, precision = 0.946700 epoch = 9, precision = 0.949500 epoch = 10, precision = 0.949300 epoch = 11, precision = 0.950800 epoch = 12, precision = 0.951900 epoch = 13, precision = 0.952900 epoch = 14, precision = 0.953300 epoch = 15, precision = 0.953900 sh: 1: pause: not found rosetta@rosetta-OMEN-by-HP-Laptop-15-dc1xxx:~/mnist/build\$</pre>	<pre>[100%] Built target mnist rosetta@rosetta-OMEN-by-HP-Laptop-15-dc1xxx:~/mnist/build\$./mnist 60000, train images success 60000, train labels success 10000, test images success 10000, test labels success 36040 epoch = 1, precision = 0.851800 epoch = 2, precision = 0.890600 epoch = 3, precision = 0.914300 epoch = 4, precision = 0.925200 epoch = 5, precision = 0.935600 epoch = 6, precision = 0.942500 epoch = 7, precision = 0.938900 epoch = 8, precision = 0.945400 epoch = 9, precision = 0.948200 epoch = 10, precision = 0.950400 epoch = 11, precision = 0.951800 epoch = 12, precision = 0.952500 epoch = 13, precision = 0.952400 epoch = 14, precision = 0.953600 epoch = 15, precision = 0.954000 sh: 1: pause: not found rosetta@rosetta-OMEN-by-HP-Laptop-15-dc1xxx:~/mnist/build\$</pre>

5 总结与体会

通过本次课设，我们初步了解到计算机语言中神经网络的魅力，这是用代码对人脑的识别系统的简单模拟，是人类创造的计算机世界与自然创造的人脑智慧相衔接的初步桥梁。课设任务中，我们成功完成了利用汇编语言实现对人脑识别数字的初步模拟，同时也面临了-的问题，显然，人脑神经结构旁支交错，现阶段的模拟程度还远远不够。课设中，我们将任务分工，难点一步步攻克，最终的成功结果也让我们明白了团队合作、个人学习精神的重要性。