高级语言程序设计

课设报告

题　　目　手写数字分类

学　　号　保护隐私，此处省略

姓　　名　保护隐私，此处省略

指导教师　保护隐私，此处省略

提交日期　2022 年 5 月 6 日

目录

1 需求分析.....3
1.1 功能需求.....3
1.2 数据需求.....3
1.3 界面需求.....3
1.4 开发与运行环境需求.....4
1.5 其他方面需求.....4
2 项目程序总体设计.....4
2.1 程序模块设计.....4
2.2 程序动态流程设计.....7
2.3 主要数据结构.....8
3 项目原理.....9
3.1 激活函数与损失函数.....9
3.2 梯度下降法推导.....11
3.3 反向传播算法.....12
3.4 实现技术点分析.....15
4 测试.....16
5 用户手册.....20
6 总结提高.....20
6.1 课程设计总结.....20

1 需求分析

1.1 功能需求

利用多层感知机和反向传播网络，识别 MNIST 手写数字数据集。每训练 10000 个数据，就打印一次阶段性实验数据，包括训练轮数、当前数据索引、当前数据标签、各输出层神经元的激活值。每完成一轮训练，打印该轮训练中，测试集在神经网络模型上的准确率。

1.2 数据需求

数据来源是 MNIST 手写数字数据集，其中共有 60000 份训练集数据和 10000 份测试集数据。数据集的官网链接是 http://yann.lecun.com/exdb/mnist/（已设为超链接）。每份 MNIST 数据都以二进制文件形式储存，并包括图片（Image）和标签（Label）两部分，图片部分是一张 28×28 像素的手写数字图片，标签部分是该图片所对应的数字。

由于图片和标签是各自作为一个二进制文件单独存放的，所以共有 4 份数据文件，即训练集图片、训练集标签、测试集图片和测试集标签。

对于图片文件而言，组成结构为 4 字节的幻数（此处可理解为一个标识）、4 字节的图片张数、4 字节的图片高度、4 字节的图片宽度以及多个 1 字节的像素点灰度值。像素值取值为 0 到 255,0 和 255 分别表示纯白和纯黑，且数值越靠近 0 就越白，越靠近 255 就越黑。

对于标签文件而言，组成结构为 4 字节的幻数、4 字节的图片张数和多个 1 字节的图片标签值。标签值取值为 0 到 9，对应 0 到 9 的数字。

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	60000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
....			
xxxx	unsigned byte	??	label

The labels values are 0 to 9.

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
....			
xxxx	unsigned byte	??	pixel

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

TEST SET LABEL FILE (t10k-labels-idx1-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	10000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
....			
xxxx	unsigned byte	??	label

The labels values are 0 to 9.

TEST SET IMAGE FILE (t10k-images-idx3-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	10000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
....			
xxxx	unsigned byte	??	pixel

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

Fig.1 MNIST 官网截图

1.3 界面需求

暂无界面需求。

1.4 开发与运行环境需求

开发环境：Visual Studio 2022

建议运行环境：Visual Studio 2022

1.5 其他方面需求

VS 项目的资源文件栏中需要附上四份数据文件，数据文件下载地址为 http://yann.lecun.com/exdb/mnist/，下载后需按程序中所写内容更改文件名称。

2 项目程序总体设计

在项目启动的时候，首先从文件中读入 MNIST 手写数字的数据并逐一储存。由于文件的难操控性，项目不直接对文件进行操作，而是将数据在对应数组中并对数组进行操作。接下来是对模型进行初始化，随后利用多层感知机和反向传播神经网络组成的简单的全连接网络对模型进行训练和检测，并返回测试集在模型上的准确率。

项目的总体流程图如图 Fig.2 所示。

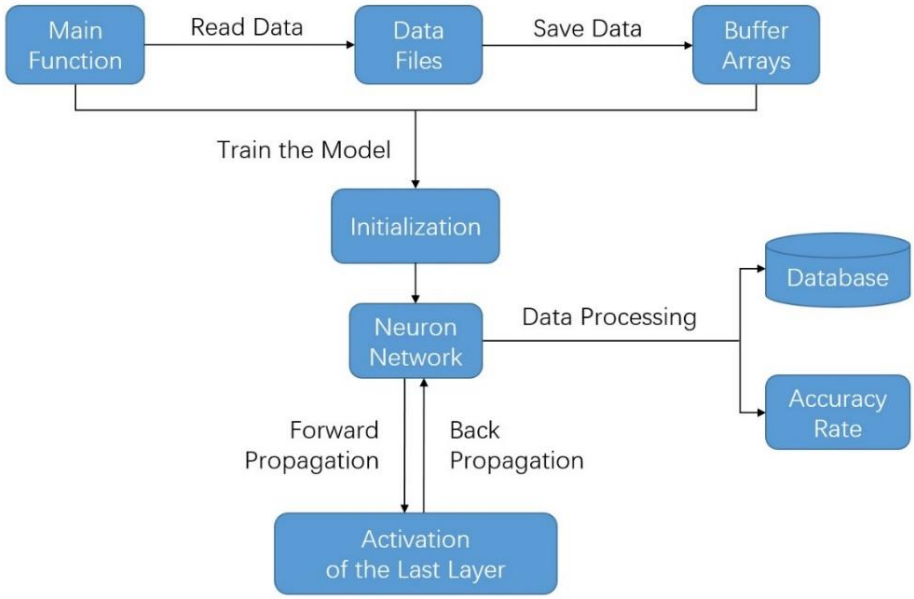

Fig.2 整体程序流程图

2.1 程序模块设计

模块编号：Function1 主函数

函数原型：int main( )

功能：①读取训练集和测试集文件并存储所需数据；

②分配所需内存空间；

③初始化所需参数；

④训练神经网络；
⑤打印阶段性训练结果并存储数据；
⑥打印每一轮训练结果的准确率；
⑦学习率动态衰减；
⑧释放所申请的空间和内存；
参数：无参数
返回值：无返回值

模块编号：Function2　激活函数
函数原型：double Activation_Function(double x)
功能：实现激活函数
参数：参数为 x 的值，双精度浮点型，表示前一层神经元参数的加权和
返回值：激活函数的函数值 function_result

模块编号：Function3　随机数生成器
函数原型：double Random_W_B( )
功能：为权重 w 和偏置 b 的数值初始化
参数：无参数
返回值：取值区间为[-1, 1]的随机数

模块编号：Function4 神经元初始化
函数原型：void InitialNeuronNet(NeuronNet* neuronnet,int layer_num,int* Array_Of_Neuron_Num_Each_Layer)
功能：①为神经层开辟动态空间；
②为每层神经元分配动态内存；
③从第二层开始初始化神经元参数；
参数：①参数 neuronnet，指向结构体 NeuronNet 的指针，表示单个神经元；
②参数 layer_num，整型，表示神经层层数；
③Array_Of_Neuron_Num_Each_Layer，指向整型数组的指针，表示存放参数每层神经元个数的数值的数组；
返回值：无返回值

模块编号：Function5 正向传播网络
函数原型：void Forward_Propagation(NeuronNet* neuronnet, double* inputs)
功能：①初始化输入层神经元的激活值；
②从第二层开始依次计算本层神经元的激活值，直至输出层;
参数：①参数 neuronnet，指向结构体 NeuronNet 的指针，表示单个神经元;
②参数 inputs，指向双精度浮点型数组的指针，表示读入的图片数据的灰度值;
返回值：无返回值

模块编号：Function6 反向传播网络
函数原型：void Back_Propagation(NeuronNet* neuronnet, double* targets)
功能：求最后一层每个神经元的梯度
参数：①参数 neuronnet，指向结构体 NeuronNet 的指针，表示单个神经元;

②参数 targets，指向双精度浮点型数组的指针，表示模型的真实值;
返回值：无返回值

模块编号：Function7　数据储存
函数原型：void Save_Database(NeuronNet* neuronnet, FILE* fpModel)
功能：储存二进制的模型数据，即从第二层开始记录每一神经层的每一神经元的权重及偏置
参数：①参数 neuronnet，指向结构体 NeuronNet 的指针，表示单个神经元;
②参数 fpModel，指向文件的指针，表示用以储存二进制模型数据的文件;
返回值：无返回值

模块编号：Function8　数据读取
函数原型：void Read_Database(NeuronNet* neuronnet, FILE* fpModel)
功能：读取二进制的模型数据，即从第二层开始读取每一神经层的每一神经元的权重及偏置
参数：①参数 neuronnet，指向结构体 NeuronNet 的指针，表示单个神经元;
②参数 fpModel，指向文件的指针，表示用以储存二进制模型数据的文件;
返回值：无返回值

模块编号：Function9 数据灰度值的读取与存储
函数原型：void initImgArray(FILE* fpImg, double** BufferArray, int Image_Num)
功能：读取数据集中的灰度值并存储到数组中
参数：①参数 fpImg，指向文件的指针，表示存储图片数据的文件；
②参数 BufferArray，指向双精度浮点型二维数组指针的指针，以 BufferArray[i][j]为例，表示第 i 张图片的第 j 格像素点;
③参数 Image_Num，整型，表示图片张数；
返回值：无返回值

模块编号：Function10 数据标签的读取与存储
函数原型：void initLabelArray(FILE* fpLabel, int* BufferArray, int Label_Num)
功能：读取数据集中的标签并存储到数组中
参数：①参数 fpLabel，指向文件的指针，表示存储标签数据的文件；
②参数 BufferArray，指向整型数组的指针，以 BufferArray[i]为例，表示第 i 张图片的标签；
③参数 Label_Num，整型，表示标签个数；
返回值：无返回值

模块编号：Function11 准确率计算器
函数原型：double Accuracy_Rate(NeuronNet* neuronnet, double** Test_Buffer_Array_Image, int* Test_Buffer_Array_Label)
功能：①将测试集数据送入模型进行测试并得到判断正确的数据个数；
②以公式

$$模型准确率 = \frac{正确判断的数据个数}{测试集数据总数} \times 100\%$$

进行计算，得到模型的准确率；

参数：①参数 neuronnet，指向结构体 NeuronNet 的指针，表示单个神经元;

②参数 Test_Buffer_Array_Image，指向双精度浮点型数组指针的指针，以 Test_Buffer_Array_Image[i][j]为例，表示第 i 组测试集数据的第 j 个输入层神经元的值（也可理解为第 i 张测试集图片的第 j 格像素点的灰度值）;

③参数 Test_Buffer_Array_Label，指向整型数组的指针，以 Test_Buffer_Array_Label[i]为例，表示第 i 组测试集数据的标签值;

返回值：模型准确率

2.2 程序动态流程设计

本项目共有四个重要的模块，即神经元初始化模块、正向传播模块、反向传播模块和准确率计算模块。

在神经元初始化模块中，首先为神经层开辟动态空间并分配动态内存给每层的神经元，随后从第二层以赋随机值的方式初始化每层神经元与前一层神经元之间的权重ω和偏置b。

在正向传播模块中，先初始化输入层每个神经元的激活值，再构造函数关系计算出下一层神经元的加权和及激活值。

在反向传播网络中，根据输出层神经元的激活值和损失函数，使用梯度下降法对权重ω和偏置b求偏导并依次通过反向传播的方法更新参数。

在准确率计算模块中，导入测试集数据后，代入训练好的神经网络模型中并通过正向传播模块得到输出层的激活值，将模型的真实值和测试结果进行比对，再利用准确率公式计算得出模型的准确率。

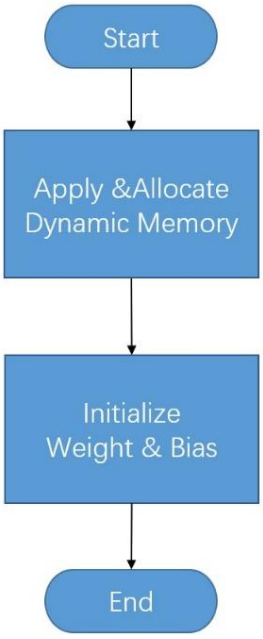

Fig.3 神经元初始化

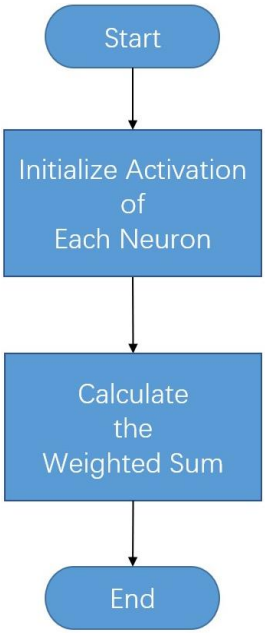

Fig.4 正向传播

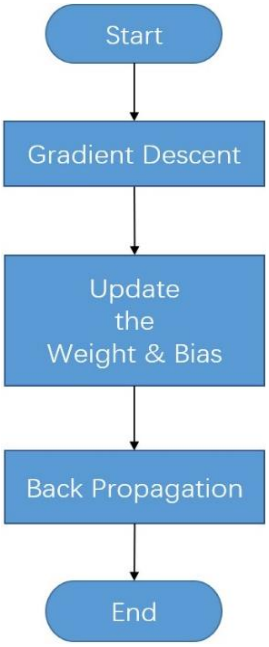

Fig.5 反向传播

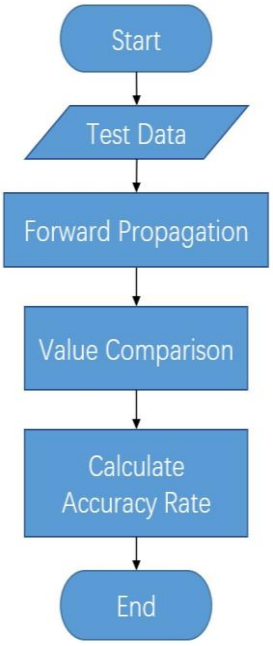

Fig.6 准确率计算

2.3 主要数据结构

2.3.1 宏定义

```
# define TRAIN_NUM    60000    // 训练数据量
# define TEST_NUM    10000    // 测试数据量
# define LEARNING_RATE 0.1    // 初始学习率
# define LAYERS_NUM 5    // 神经网络层数
# define INPUT_SIZE 784    // 输入层神经元个数
# define HIDDEN_SIZE1 50    // 隐含层一神经元个数
# define HIDDEN_SIZE2 16    // 隐含层二神经元个数
# define HIDDEN_SIZE3 16    // 隐含层三神经元个数
# define OUTPUT_SIZE 10    // 输出层神经元个数
# define EPOCHS 120 // 训练轮数
```

2.3.2 全局变量

```
double learning_rate = LEARNING_RATE;    // 学习率
```

2.3.3 结构体与链表

（1）神经网络

```
typedef struct NeuronNet
{
    int Layer_Num;    // 该神经网络中的神经层层数
    Layer* layer;    // 指向神经层结构体的指针
} NeuronNet;    // 自定义结构体名称
```

（2）神经层

```
typedef struct Layer
{
    int Neuron_Num; // 该神经层中神经元个数
    Neuron* neuron; // 指向神经元结构体的指针
} Layer; // 自定义结构体名称
```

（3）神经元

```
typedef struct Neuron
{
    double b;    // 神经元的偏置
    double a;    // 神经元的激活值
    double z;    // 神经元的加权和（Z = ω·X + b，其中ω和X为张量）
    double* w;    // 指向权重张量的指针
    double pd;    // 损失函数对当前神经元偏置的偏导（梯度）
} Neuron;    // 自定义结构体名称
```

3 项目原理

本项目是基于多层感知机和反向传播网络的机器学习项目，目标是实现识别 MNIST 手写数字数据集。此模块将展示本项目的基本原理以及数学推导过程。

3.1 激活函数与损失函数

词条上对激活函数的解释是“在人工神经网络的神经元上运行的函数，负责将神经元的输入映射到输出端”。常用的激活函数有 Sigmoid 函数、Tanh 函数、ReLU 函数等，本项目依次尝试了 Sigmoid 函数、Elu 函数和 Tanh 函数作为激活函数的效果，并选择 Sigmoid 函数作为最终的激活函数。

以下是对几种激活函数的简单介绍。

Sigmoid 函数的函数值取值范围为(0,1)，多用来做二分类，一般情况下在特征相差比较复杂或是相差不是特别大时效果比较好。但是在反向传播时，Sigmoid 函数可能会由于其指数级的运算而出现梯度消失现象，即随着隐含层层数增加，模型准确率不增反减的现象。Sigmoid 函数的公式为

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

其导数可以用自身表示，即为

$$\frac{d\,Sigmoid(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = Sigmoid(x)\big(1 - Sigmoid(x)\big)$$

以下为用 Python 绘制的 Sigmoid 函数图像。

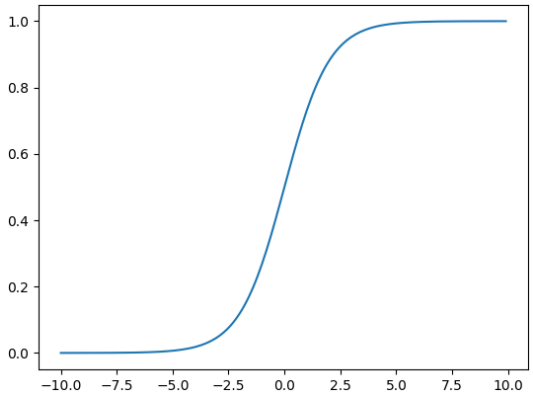

Fig.7 Sigmoid 函数图像

提到 Elu 函数，就不得不说 ReLU 函数。在训练神经网络的时候，由于 ReLU 函数会使负值全部变成 0，极有可能导致某一神经元的参数全部归零而无法进行后续操作，形成神经元“坏死”的情况。因此，Elu 函数在 ReLU 函数的基础上进行改进，解决了神经元坏死的现象。两种函数的函数表达式和图像如下。

$$ReLU(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

$$Elu(x) = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$$

Elu 函数的导函数公式为

$$\frac{d\ Elu(x)}{dx} = \begin{cases} 1, & x \geq 0 \\ \alpha(e^x - 1) + \alpha, & x < 0 \end{cases}$$

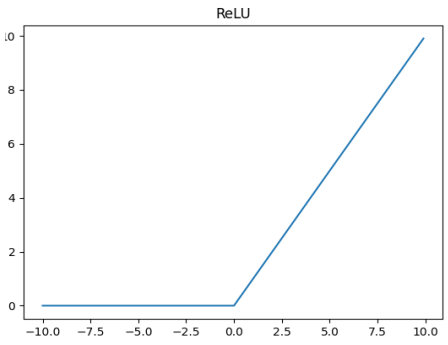

Fig.8 ReLU 函数图像

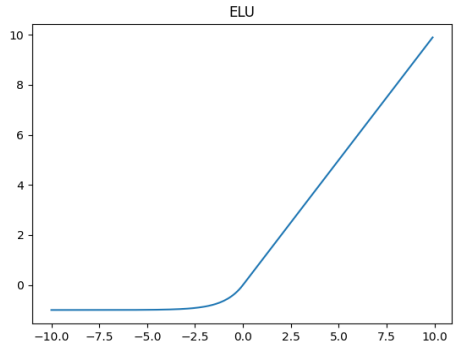

Fig.9 Elu 函数图像

一般情况下，Elu 函数的α值都是人为规定的常数值，在本项目中，该值取 1。

还有一种常用的激活函数是 Tanh 函数，即双曲正切函数，是双曲正弦函数和双曲余弦函数的比值。双曲函数是三角函数通过欧拉公式等复指数形式变换得到的另一种基本初等函数。双曲正切函数的函数表达式为

$$tanhx = \frac{sinhx}{coshx} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

其导数为

$$\frac{d\ tanhx}{dx} = \frac{1}{(coshx)^2} = \frac{4e^{2x}}{(e^{2x} + 1)^2} = 1 - (tanhx)^2$$

函数图像为

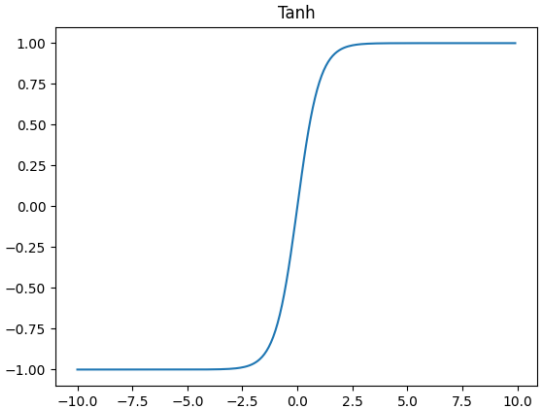

Fig.10 Tanh 函数图像

对于 Tanh 函数和 Sigmoid 函数，还要一条性质，即二者表达式之间的相互转化。转换公式如下。

$$Tanh(x) = 2\big(Sigmoid(2x)\big) - 1$$

除了激活函数以外，对于整个模型来说，还有一个非常重要的函数，即损失函数。损失函数又名代价函数，往往被用来衡量事件的“风险”或“损失”。在本项目中，损失函数被用以评价模型的准确性。

损失函数的种类有很多，此处使用的是修改过的均方差损失函数，即

$$J(\omega, b) = \frac{1}{N}\sum_{1}^{N}\|y - a\|^2$$

$$a = \sigma(z) = \sigma(\omega \cdot x + b)$$

其中，$J(\omega, b)$表示损失函数，a表示神经元激活值，$\sigma(\cdot)$是激活函数。

3.2 梯度下降法推导

梯度下降是迭代法的一种，可以用于求解最小二乘问题(线性和非线性都可以)。在求解机器学习算法的模型参数，即无约束优化问题时，梯度下降是最常采用的方法之一；在求解损失函数的最小值时，可以通过梯度下降法来一步步的迭代求解，得到最小化的损失函数和模型参数值。

以上为“梯度下降”词条的部分解释，本项目利用梯度下降法求解最小损失函数。

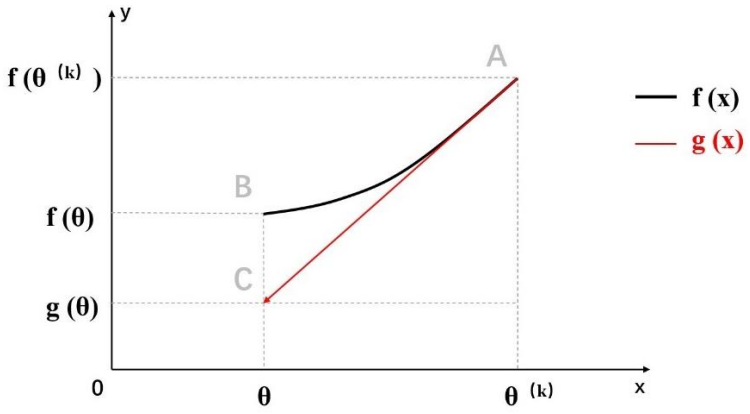

Fig.11 梯度下降法原理推导图解

对于直线 $g(x) = g'(x)x + \mathrm{b}$, $g'(x)$为$f(x)$的梯度$\nabla f(x)$，即用点 C 近似得到点 B。

则有

$$f(\theta) \approx g(\theta) = f\big(\theta^{(k)}\big) - \big[f\big(\theta^{(k)}\big) - g(\theta)\big]$$

$$= f\big(\theta^{(k)}\big) - \nabla f\big(\theta^{(k)}\big)\big(\theta^{(k)} - \theta\big)$$

整理后，得

$$f(\theta) \approx f\left(\theta^{(k)}\right) + \left(\theta - \theta^{(k)}\right) \nabla f\left(\theta^{(k)}\right)$$

设$\theta - \theta^{(k)} = \eta' \upsilon$，其中$\eta'$为标量，$\upsilon$为单位向量（$\eta' > 0$）

$$\therefore f(\theta) \approx f\left(\theta^{(k)}\right) + \eta' \upsilon \nabla f\left(\theta^{(k)}\right)$$

∵ 要更新θ使函数值不断减小，达到下降效果

∴ 有$\eta' \upsilon \nabla f\left(\theta^{(k)}\right) < 0$　　又 ∵ $\eta' > 0$　　∴ $\upsilon \nabla f\left(\theta^{(k)}\right) < 0$

又 ∵ υ为单位向量， $\nabla f\left(\theta^{(k)}\right)$为梯度，亦为向量　　∴ υ*与* $\nabla f\left(\theta^{(k)}\right)$*方向相反*

即

$$\upsilon = -\frac{\nabla f\left(\theta^{(k)}\right)}{\|\nabla f(\theta^{(k)})\|}$$

更新θ得，$\theta^{(k+1)} = \theta^{(k)} + \left(\theta^{(k+1)} - \theta^{(k)}\right)$

即

$$\begin{aligned}\theta^{(k+1)} &= \theta^{(k)} + \eta' \upsilon \\ &= \theta^{(k)} + \eta' \left(-\frac{\nabla f\left(\theta^{(k)}\right)}{\|\nabla f(\theta^{(k)})\|}\right)\end{aligned}$$

令 $\eta = \eta' \left(-\frac{1}{\|\nabla f(\theta^{(k)})\|}\right)$ 得

$$\theta^{(k+1)} = \theta^{(k)} + \eta \nabla f\left(\theta^{(k)}\right)$$

证毕。

3.3 反向传播算法

反向传播算法是非常简单，也是很基础的机器学习算法。

在训练一个神经网络的时候，我们的目的就是使模型中的参数都取到最理想的数值。对于一个简单的神经网络模型，我们可以通过人工计算解得对应参数并在程序的开头由宏定义设置好，以达到最优的效果。但是，神经网络往往是庞大的，有上万成亿个参数，甚至数十亿个参数。本项目计算量不算很大，但参数量也是以万计的。也在这种情况下，人力计算非常低效。因此，本项目选择反向传播算法。反向传播网络分为两大部分，第一是正向传播部分，目标是计算求解出损失函数；第二是反向传播部分，目标是修改模型中的参数，以提高模型性能。

一般情况下，神经网络是由一个输入层、多个隐含层和一个输出层组成的。输入层的神经元数值为X_i（$i = 1, 2, 3 \ldots$），即为输入到神经网络模型中的样本。隐含层数目和每层网络的神经元个数都是人为规定的。通常来说，样本数量越大，隐含层层数就要越多才能保证模型的性能。而每层的神经元个数则会根据构造法、删除法、黄金分割法等有数学理论支撑的计算方法选择出较优的神经元个数。输出层用以输出神经网络的运算结果，该层的神经元个

数与项目需求有关。

在正向传播过程中，神经网络通过权重ω和偏置 b 依次构造前一层每个神经元的激活值与后一层每个神经元初值之间的函数关系，从而计算出后一层神经元的初值，由激活函数加工后作为新的激活值与再下一层神经网络进行运算。

以八个神经元的四层神经网络为例（如图 Fig.12 所示）。

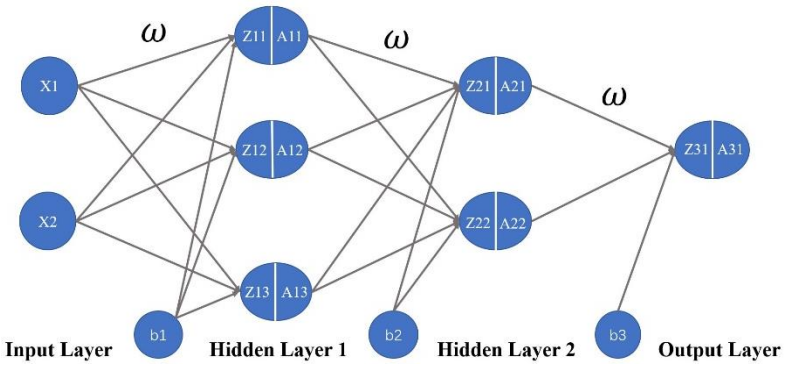

Fig.12 正向传播示意图

该网络共有一个输入层、两个隐含层和一个输出层，除第一层之外，每一层与前一层的神经元之间都有一个权重向量ω和一个偏置 b。因此，由X_1、X_2、ω和 b_1组成的函数可以计算出Z_{11}的数值。将Z_{11}代入激活函数中，可以求解出该神经元的激活值A_{11}。同理可得A_{12}。和A_{13}，再由A_{12}、A_{13}和新的ω、b可以求解出Z_{21}和Z_{22}，代入激活函数得到A_{21}和A_{22}。依次类推，即可得到输出层的结果A_{31}。由于激活函数不一定是线性函数，所以这一过程可能产生非线性的结果，加之以神经网络的复杂性，整个模型也可以有效地处理线性问题之外的其他情况。

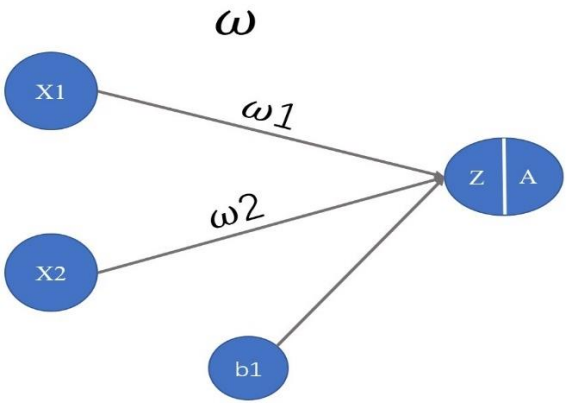

Fig.13 神经元间链接方式示意图

在神经网络中，每两个神经元之间的链接方式都呈线性函数关系。以图 Fig.13 为例，初值Z的具体数值可以由以权重向量ω和偏置 b_1作为常量且前一层神经元激活值X_1和X_2作为变量的线性函数求得。假设激活函数为$\sigma(\cdot)$，则激活值A可以Z和$\sigma(\cdot)$表示。

需要注意的是，没有角标的ω均指向量$(\omega_1,\ \omega_2)$。

即

$$Z = \sum_{i}^{N} (\omega \cdot X) + b$$
$$= \omega_1 \times X_1 + \omega_2 \times X_2 + b$$
$$A = \sigma(Z)$$
$$= \sigma(\omega_1 \times X_1 + \omega_2 \times X_2 + b)$$

在反向传播的过程中，将均方差损失函数的算式完全展开后可以得到关于各层权重和偏置的多元函数。为了求解出这些参数以确定其最佳值，我们需要对所有参数求偏导，即求解损失函数的梯度。

以五个神经元的三层神经网络为例（如图 Fig.14 所示）。

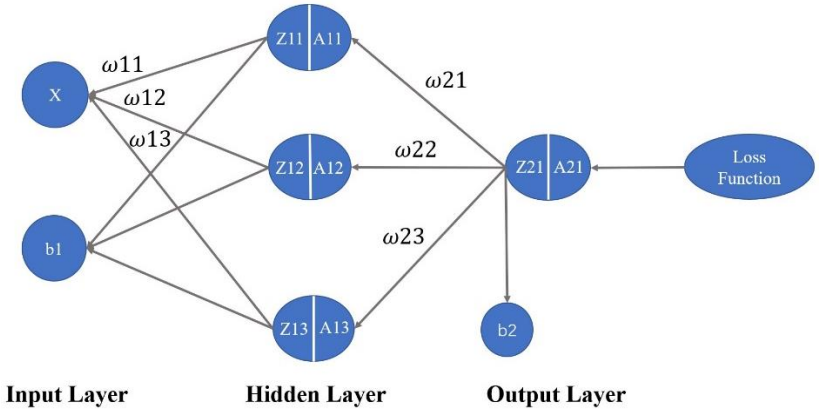

Fig.14 反向传播示意图

由公式可得，模型的损失函数为

$$Loss\ Function = \frac{1}{N}\sum_{1}^{N} \|y - a\|^2$$

代入图中模型可以求得损失函数展开后的表达式为

$$Loss\ Function = (y - a)^2$$

其中，y 为模型的真实值，a为关于权重ω和偏置b的函数，$\sigma(\cdot)$表示激活函数，且a的表达式如下。

$$a = \sigma(\omega_{21} \times \sigma(\omega_{11} \times X + b_1) + \omega_{22} \times \sigma(\omega_{12} \times X + b_1) + \omega_{23} \times \sigma(\omega_{13} \times X + b_1) + b_2)$$

为了求出所有的权重ω和偏置b，对表达式求关于参数的偏导数。以权重ω_{21}为例，得到

$$\frac{\partial\ Loss\ Function}{\partial\ \omega_{21}} = -(targetA_{21} - A_{21}) \times \frac{d\sigma(x)}{dx}$$

$$\omega_{21}{}^{*} = \omega_{21} - \eta \times \frac{\partial\ Loss\ Function}{\partial\ \omega_{21}}$$

其中，$targetA_{21}$表示A_{21}的真实值，$\omega_{21}{}^{*}$表示更新后的权重ω_{21}，η为学习率（或称模型

的学习步长）。

如以上例子所述，可通过反向传播算法更新神经网络中的所有参数并再次训练模型。当如此反复进行多次正反向传播后，即可确定效果较好的参数，提高模型的性能，完成预定的目标。

3.4 实现技术点分析

3.4.1 Box-Muller 方法

Box-Muller 方法是获取服从高斯分布（即正态分布）的随机数的常用方法。其思想是先得到服从均匀分布的随机数再将服从均匀分布的随机数转变为服从正态分布的随机数。该方法的原理和推导过程较为复杂，这里不多赘述，仅列举该方法的表述和公式。

定理（Box-Muller 变换）如果随机变量$U1$、$U2$独立且$U1, U2 \sim Uniform[0,1]$，则

$$Z_0 = \sqrt{-2\ln U_1}\, cos(2\pi U_2)$$

$$Z_1 = \sqrt{-2\ln U_1}\, sin(2\pi U_2)$$

则，Z_0、Z_1独立且服从标准正态分布。

3.4.2 归一化处理

在某论坛对归一化有这样一种解释：“在机器学习领域，往往不同评价指标（即特征向量中的不同特征就是所述的不同评价指标）往往具有不同的量纲和量纲单位，这样的情况会影响到数据分析的结果，为了消除指标之间的量纲影响，需要进行数据标准化处理，以解决数据指标之间的可比性。原始数据经过数据标准化处理后，各指标处于同一数量级，适合进行综合对比评价。”

但需要注意的是，归一化和标准化是有区别的，但很多初学者会弄混二者之间的概念。归一化处理后的数据是被严格限制在[0,1]的区间内的，而标准化只对数据的均值和标准差有限制，对数据本身的限制并不大。补充一句，常见的标准化方法是零均值标准化。

常见的归一化方法有均值归一化、线性归一化等，本项目使用的是线性归一化方法。

即

$$Xi^* = \frac{Xi - Xmin}{Xmax - Xmin}$$

其中，Xi^*是原数值Xi由归一化处理变换得到的结果，$Xmin$是X的最小值，$Xmax$是X的最大值。

4 测试

本模块将阐述整个项目的参考资料及测试、调试过程。

初期，本项目参考了很多博客园、CSDN、Github 等论坛的文章、代码，以及知乎论坛、BiliBili 视频网站等平台的讲解视频，其中帮助最大的是 BiliBili 视频网站的某个视频（视频链接：https://www.bilibili.com/video/BV1NT4y1G7Rw?share_source=copy_web）。

在明确了思路之后，代码初稿很快完工，最初的超参数设置与参考视频中设置一致，即学习率恒为 0.3，采用 2 层隐含层结构，每层 50 个神经元，随机数设置为[-1, 1]之间的随机数，激活函数使用 Sigmoid 函数，训练 50 轮。但是初代模型的性能较差，准确率低于 10%。

于是，本项目尝试更改隐含层的神经元个数。多次尝试后发现，虽然准确率有所提升，但一直维持在 10%至 20%之间，效果不佳。以其中一次实验为例，保持其他参数不变，仅更改隐含层神经元个数为 100 和 50 个，记录每轮的准确率结果并绘制折线统计图如下。

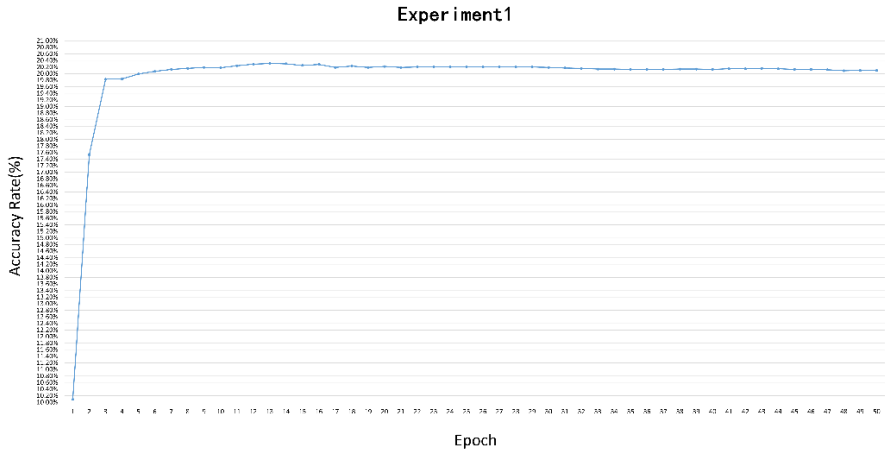

Fig.15 实验一

可以看出，本轮实验的准确率收敛在 20.1%上下。

根据参考视频和相关资料，原始的随机数模块公式为

$$Random\_Num = 2 \times \frac{rand()}{RAND_{MAX}} - 1.0$$

其中，$rand()$是 C 语言标准库<stdlib.h>中所含的伪随机数生成函数，RAND_MAX 则 C 语言标准库<stdio.h>中定义的一个宏，表示$rand()$所能返回的最大数值，对于 int 来说，RAND_MAX 的值为 32767，对于 unsigned int 来说，则是 65535。

为提高模型的准确率，项目更改了权重ω和偏置b的随机数公式。新的公式参考 C 语言中生成[n,m]范围内随机数的公式来生成[-1,1]间的随机数。

即

$$Random\_Num = (-1) + rand()\%(1 - (-1) + 1)$$

修改后的模型准确率大幅提升，但是产生了明显的震荡现象。以其中一次实验为例，

报错隐含层层数为 2，每层神经元个数为 50，学习率为 0.3，激活函数为 Sigmoid 函数不变，仅修改随机数公式，训练 50 轮后得到以下图像。

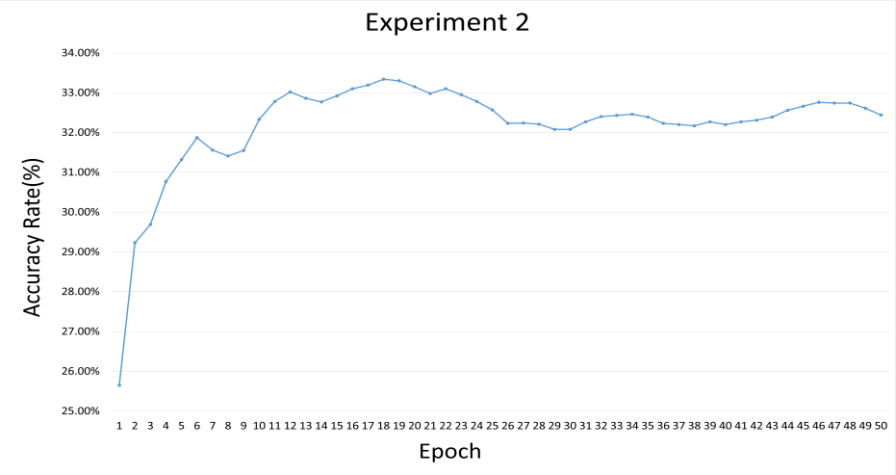

Fig.16 实验二

将实验一和实验二结合，即同时更改隐含层神经元个数为实验一神经元个数并修改随机数公式为实验二公式，得到效果更好的实验三。实验三中，以准确率为纵轴、训练轮数为横轴的折线统计图如下。

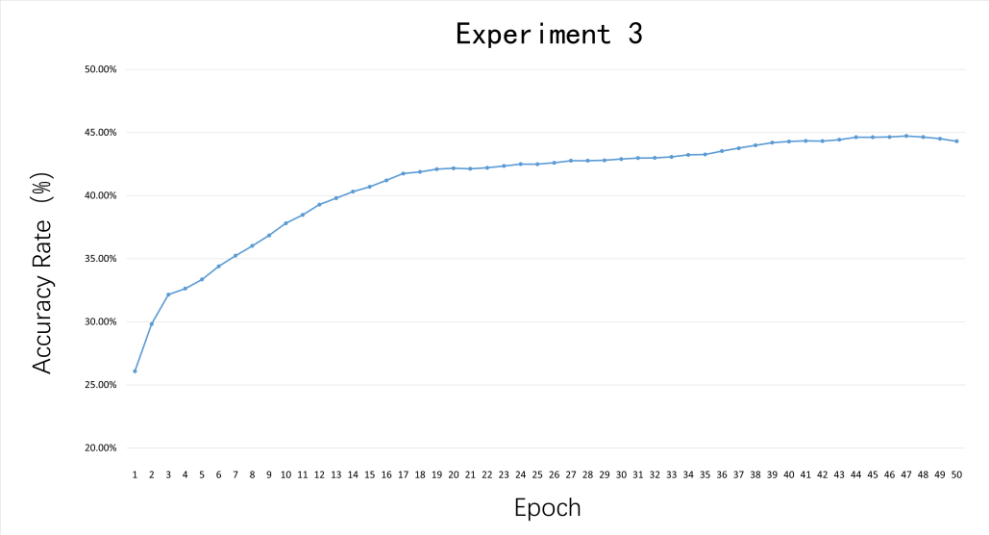

Fig.17 实验三

在前三种实验方式的基础上，项目新增一层神经层，希望通过增加网络的复杂程度以提高整个系统的鲁棒性。同时，模型通过缩减学习率来缩短梯度下降的步长，放弃速率而追求精度。在进行了多轮实验后，模型的性能再次实现质的飞跃。

训练过程中，出现一个非常有意思的现象，即每层的神经元个数较少的时候，其性能也可能比大量神经元训练效果更好。到目前为止，在所有的数十次实验中，效果最佳的是一个五层神经网络，即 1 层输入层、3 层隐含层、1 层输出层。隐含层神经元个数依次为 50、16 和 16，权重向量和偏置的初始值是以实验二公式生成的[-1,1]的随机数，学习率为 0.1。该次实验的准确率一度达到 47.11%且保持上升趋势。训练 50 轮的实验四图像如下。

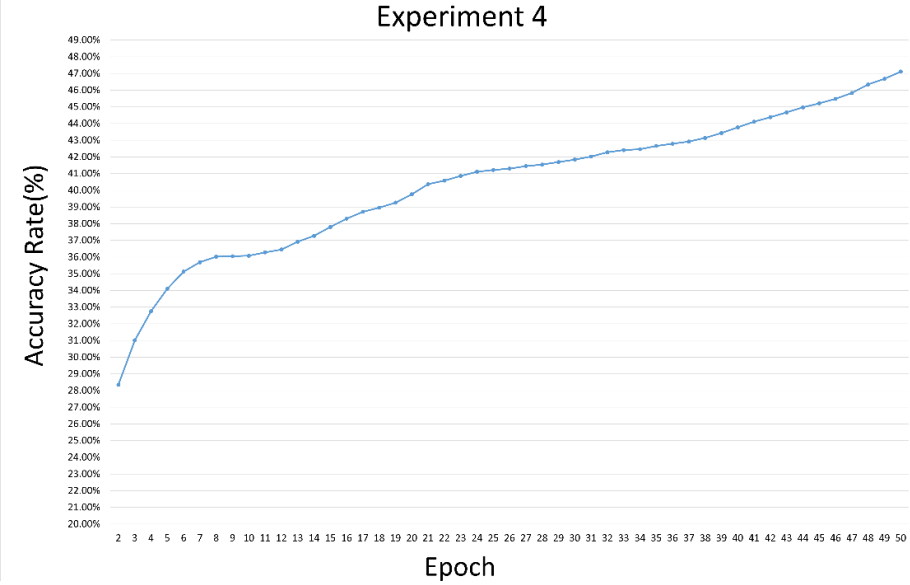

Fig.18 实验四

在后续的实验中，项目修改了损失函数的公式，将原公式乘以$\frac{1}{2}$，再次进行训练时发现，准确率上升速率剧烈增长，但是实验中前期就出现明显的过拟合现象，即模型在达到较良好的效果之后，过分追求细枝末节上的精确程度，而导致模型参数往尽可能满足训练集的角度上更新，最终逐渐削弱对除训练集之外样本的识别能力的现象。

典型案例是当学习率为 0.3，隐含层设为 3 层，神经元个数分别为 100、50 和 16，且随机数取值-1 到 1 之间，满足实验二公式的时候。当均方差损失函数公式被更改后，进行 50 轮训练。模型在第 1 轮的准确率就高于实验四 3.7 个百分点，更是在第 9 轮训练时就达到了 46.77%的准确率，仅比实验四第 50 轮的准确率低不到 0.5 个百分点。但是当实验轮数往后推进时，本次实验的准确率开始呈缓慢下降趋势，在第 50 轮实验时跌破 40%。

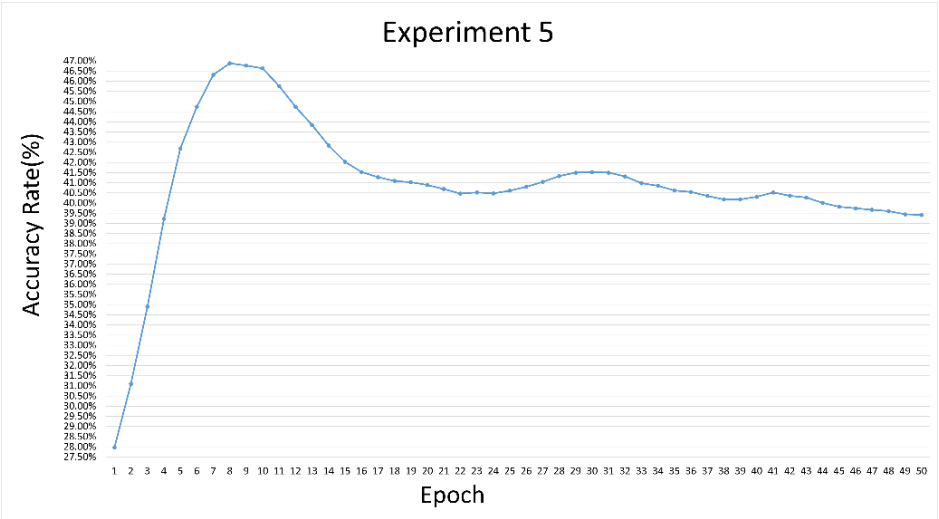

Fig.19 实验五

综合比对模型现状与过拟合原因后，项目尝试在损失函数后增加正则化项，但是效果不佳。保持实验五参数不变，修改学习率为 0.1 后，模型依旧呈过拟合现象，且准确率不

及实验五。在学习率改为 0.1 的基础上，修改实验五模型的激活函数，由 Sigmoid 函数改为 Elu 函数，其余参数不变，训练 50 轮后出现惊人的现象，即准确率保持 10.09%不变。考虑到当输入值 $X \to \infty$之后，有$Elu(X) \to \infty$成立，所以对 Elu 函数的函数值进行归一化处理，使激活值保持在 0 到 1 之间。

但遗憾的是，归一化处理并没有从根本上改变模型的缺陷，而是小幅度提高模型准确率后稳定在另一个值上。这在理论上是很难达到的结果。Elu 函数处理下的折线统计图见图 Fig.16。

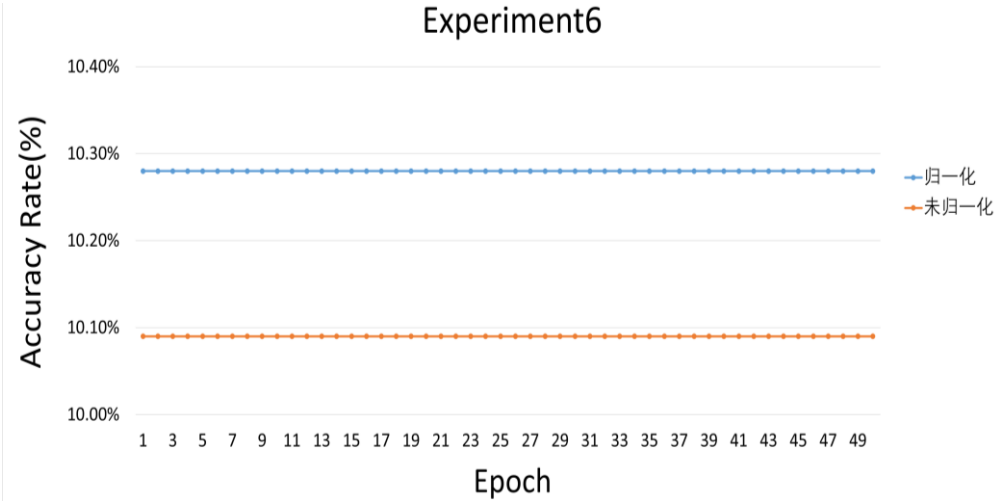

Fig.20 实验六

将激活函数设为 Tanh 函数，删去归一化处理并保持其他参数不变，得到的准确率收敛于 10%，更改学习率为 0.1，结果不变。修改其它参数，使学习率恒为 0.3，3 层隐含层神经元个数依次为 50、16 和 16，可得到震荡现象非常明显的神经网络模型。

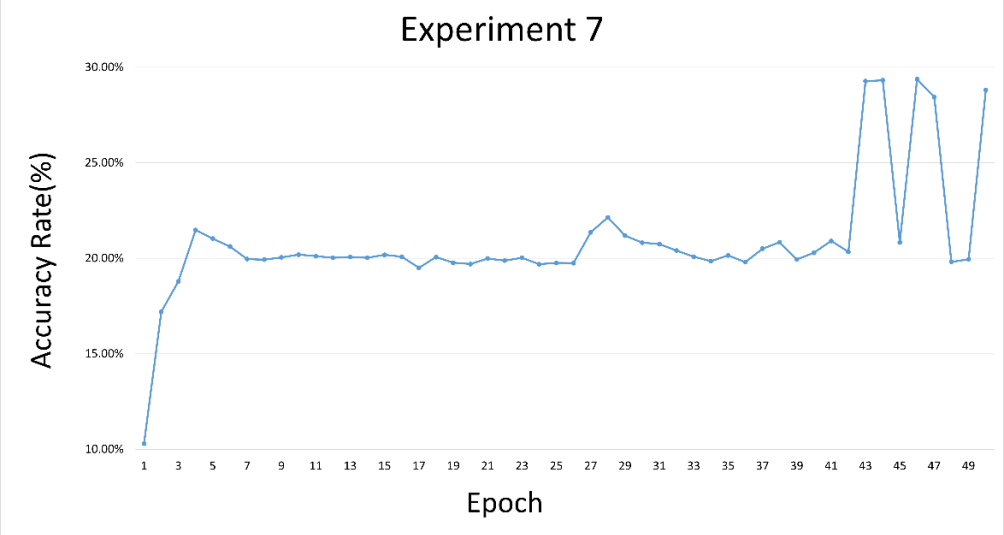

Fig.21 实验七

除此之外，项目还尝试了学习率的动态衰减，即每训练一定轮数，学习率自乘一个 0 到 1 之间的小数，以减小届时的步长，达到提高准确率的效果。再增加神经网络层数、调整神经元个数、增加训练轮数等常规方法也经常作为实验步骤。同时，项目亦尝试修改随

机数生成公式为基于 Box-Muller 方法的正态分布随机数生成模型，但可惜的是上述方法均表现不佳。

目前，性能最好的超参数还是如实验四模型所设。

5 用户手册

5.1 应用程序功能的详细说明

项目是基于多层感知机和反向传播算法的手写数字识别模型。训练、测试过程均无需用户操作。

5.2 应用程序运行环境的要求

请在 Windows 系统上进行操作。

5.3 应用程序的安装与启动方法

5.3.1 安装

本项目可下载在设备的任意目录下。

5.3.2 启动

双击打开手写数字识别程序的 exe 文件。

5.4 备注

1、本项目 exe 程序须与数据集文件放在统一文件夹下；

2、用户请勿随意更改代码内容。

6 总结提高

6.1 课程设计总结

在开发阶段，项目遇到了很多问题，包括但不限于算法实现问题、超参数设置问题、激活函数编写问题、损失函数求导问题等。在测试阶段，代码运行速率慢和准确率低成为难点。

由于本项目是在 CPU 上运行的，暂时没有加入基于 CUDA 编程的功能，以及激活函数中含有指数部分，故训练时间较长，且随着隐含层层数和神经元个数的增加，训练所需时间会爆炸式增长。

到目前为止，运行时间最长的一次是 6 层神经网络（即，1 层输入层、4 层隐含层、1 层输出层），共训练 120 轮，隐含层神经元个数依次为 485、485、256、256，初始学习率为 0.1，每训练 10 轮，学习率自乘 0.9，激活函数使用 Sigmoid 函数，权重和偏置的初始值均为取值范围为[-1,1]的随机数。该模型从 2022 年 4 月 29 日晚 23 时开始运行，到次日同时间段，共运行 24 小时左右。同时，模型准确率较低，仅从第 1 轮的 13.12%提升到第 120 轮的 33.11%，提升了 20 个百分点，最后收敛在 33%。

必须承认，模型性能并不是很好，还有改进和提升的空间。