《模式识别与机器学习 A》实验报告

实验题目：多层感知机实验

学　　号：2021112845

姓　　名：张智雄

实验报告内容

1、 实验目的

自行构造一个多层感知机，完成对某种类型的样本数据的分类（如图像、文本等），也可以对人工自行构造的二维平面超过 3 类数据点（或者其它标准数据集）进行分类。

2、 实验内容

1) 能给出与线性分类器（自行实现）作对比，并分析原因。
2) 用不同数据量，不同超参数，比较实验效果。
3) 不许用现成的平台，例如 Pytorch，Tensorflow 的自动微分工具。
4) 实现实验结果的可视化。

3、 实验环境

Windows11; Anaconda+python3.11; VS Code

4、 实验过程、结果及分析（包括代码截图、运行结果截图及必要的理论支撑等）

4.1 算法理论支撑

4.1.1 神经元模型

如图 1 所示，在这个模型中,神经元接收到来自n个其他神经元传递过来的输入信号，这些输入信号通过带权重的连接(connection)进行传递,神经元接收到的总输入值将与神经元的阈值（偏置）进行比较,然后通过“激活函数”(activation function) 处理以产生神经元的输出。

理想中的激活函数是阶跃函数$sgn(\cdot)$，它将输入值映射为输出值“0”或“1”，显然“1”对应于神经元兴奋，“0”对应于神经元抑制。然而，阶跃函数具有不连续、不光滑等不太好的性质，实际常用 Sigmoid、ReLU 等函数作为激活函数

把许多个这样的神经元按一定的层次结构连接起来,就得到了神经网络。

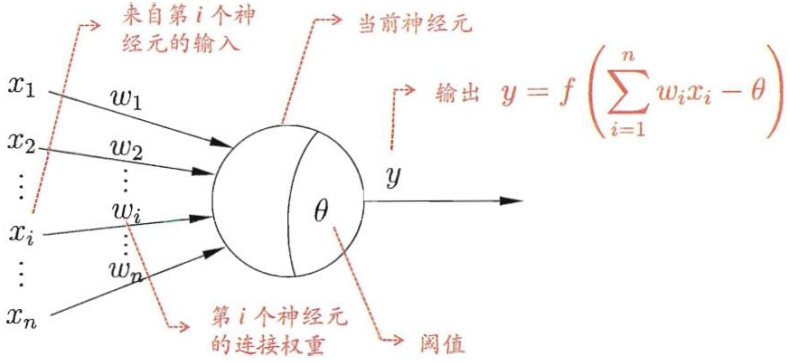


图 1 M-P 神经元模型

4.1.2 多层感知机模型

多层感知机(Multilayer Perceptron, MLP)是一种前向结构的人工神经网络，映射一组输入向量到一组输出向量，MLP 可以被看作是一个有向图，由多个的节点层所组成，每一层都全连接到下一层，除了输入节点，每个节点都是一个带有非线性激活函数的神经元。MLP 网络结构包含输入层、输出层及多个隐藏层，其中输入层神经元接收外界输入，隐藏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。3 层感知机的神经网络图如下所示：

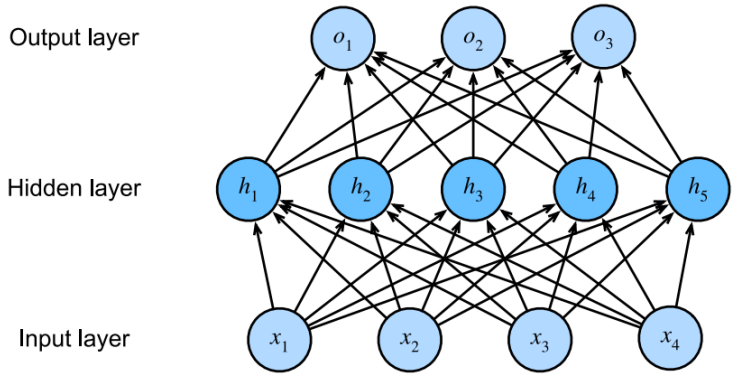


图 2 多层感知机图示

一个 MLP 可以视为包含了许多参数的数学模型，这个模型是若干个函数$y_j = f(\sum_i w_i b_i - \theta_i)$相互(嵌套)代入得到的。而对于给定由$d$个属性描述，输出为$l$维实值向量的训练集$D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}, x \in \mathbb{R}^d, y \in \mathbb{R}^l$，则隐藏层第$h$个神经元接收到的输入为$\alpha_h = \sum_{i=1}^{d} v_{ih} x_i$，输出层第$j$个神经元接收到的输入为$\alpha_h = \sum_{h=1}^{q} w_{hj} b_h$。假设神经元激活函数为$\sigma(\cdot)$。

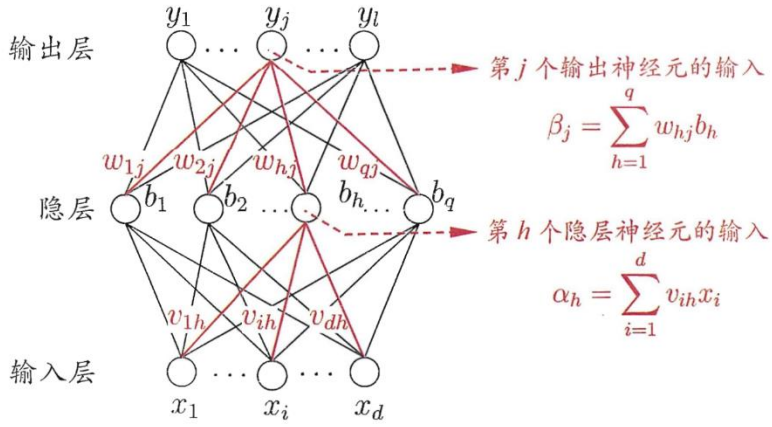


图 3 神经网络中变量符号

模型训练主要包括**前馈传播和反向传播**两个步骤，前馈传播负责计算模型的预测值，而反向传播负责计算梯度并更新模型的参数，降低损失函数，以便在训练中不断改进模型的性能。

具体而言，前馈传播是神经网络中的正向计算过程，它从输入层开始，沿着网络的层级顺序将数据传递到输出层，从而计算模型的预测值，但此过程并不涉

及权重和偏差的更新，即计算

$$y_k = g\left\{\sum_{j=1}^{m} w_{kj}^{(s)} \cdots \left[a\left(\sum_{i=1}^{n} w_{ji}^{(1)} x_i + b_j^{(1)}\right)\right] \cdots + b_k^{(s)}\right\}, k = 1,2,\cdots,l$$

反向传播则是使用前馈传播计算模型的输出，并将其与实际目标进行比较，计算损失（误差）。

ALGORITHM 1 Backpropagation(反向传播算法)

1：**input** $f(x;\theta)$ ←神经网络，θ ←参数向量，(x', y') ←样本，η ←学习率；

2：**for** $t = 1,2,\ldots,s$ **do** ***# 正向传播***

3：$h^{(0)} = x'; h^{(t)} = a\left(W^{(t)} h^{(t-1)} + b^{(t)}\right)$（$a(\cdot)$为激活函数）

4：**for** $t = s, s-1, \ldots, 1$ **do** ***# 反向传播***

5：$\delta^{(s)} = h^{(s)} - y'; \nabla_{W^{(t)}} L = \delta^{(t)} \cdot h^{(t-1)^T}, \nabla_{b^{(t)}} L = \delta^{(t)}$

6：$W_{\text{new}}^{(t)} \leftarrow W^{(t)} - \eta \nabla_{W^{(t)}L}, b_{\text{new}}^{(t)} \leftarrow b^{(t)} - \eta \nabla_{b^{(t)}L}$

7：**if** $t > 1$ **do**

8：$\delta^{(t-1)} = \frac{\partial a}{\partial z_j^{(t-1)}} \odot \left(W^{(t)^T} \cdot \delta^{(t)}\right)$

9：**return** θ

基于梯度下降GD或随机梯度下降SGD的学习算法的核心是针对给定样本，计算损失函数对神经网络所有参数的梯度$\frac{\partial L}{\partial \theta}$，并以此更新所有参数$\theta$。考虑一个$s$层神经网络，其中第$t$层的神经元定义为：

$$h_j^{(t)} = a\left(z_j^{(t)}\right), z_j^{(t)} = \sum_{i=1}^{n} w_{ji}^{(t)} h_i^{(t-1)} + b_j^{(t)}, j = 1,2,\cdots,m$$

损失函数对第t层的权重和偏置的梯度分别为$\frac{\partial L}{\partial w_{ji}^{(t)}}$和$\frac{\partial L}{\partial b_j^{(t)}}$。根据链式求导规则，可以展开为：

$$\frac{\partial L}{\partial w_{ji}^{(t)}} = \frac{\partial L}{\partial z_j^{(t)}} \frac{\partial z_j^{(t)}}{\partial w_{ji}^{(t)}}, \qquad \frac{\partial L}{\partial b_j^{(t)}} = \frac{\partial L}{\partial z_j^{(t)}} \frac{\partial z_j^{(t)}}{\partial b_j^{(t)}}$$

考虑损失函数对第t层的净输入的梯度$\delta_j^{(t)} = \frac{\partial L}{\partial z_j^{(t)}}, j = 1,2,\cdots,m$，则上式可写成：

$$\frac{\partial L}{\partial w_{ji}^{(t)}} = \delta_j^{(t)} h_i^{(t-1)}, \qquad \frac{\partial L}{\partial b_j^{(t)}} = \delta_j^{(t)}$$

而对于第t层的$\delta_j^{(t)}$，可展开为

$$\delta_j^{(t)} = \frac{\partial L}{\partial z_j^{(t)}} = \sum_{k=1}^{l} \frac{\partial L}{\partial z_k^{(t+1)}} \frac{\partial z_k^{(t+1)}}{\partial z_j^{(t)}}, j = 1,2,\cdots,m$$

求解得到$\delta_j^{(t)} = \frac{\mathrm{d}a}{\mathrm{d}z_j^{(t)}} \sum_{k=1}^{l} w_{kj}^{(t+1)} \delta_k^{(t+1)}$。其中，$\frac{\mathrm{d}a}{\mathrm{d}z_j^{(t)}}$为第$t$层激活函数关于$z_j^{(t)}$的导

数。也就是说可以根据$t+1$层的$\delta_k^{(t+1)}$计算$\delta_j^{(t)}$。

多分类问题中，输出层由l个输出表示l个类别的概率。损失函数是交叉熵损失$-\sum_{k=1}^{l} y_k \log h_k^{(s)}$，激活函数是 Softmax 函数$g(z)=\frac{e^z}{\sum_{z'} e^{z'}}$，此时误差是

$$\delta_k^{(s)} = h_k^{(s)} - y_k, k = 1,2,\cdots,l$$

而后从输出层开始基于链式法则计算损失对每个权重和偏差的梯度，使用 Adam、SGD 等优化算法来更新网络中的权重和偏差，以减小损失函数的值。

通过反复迭代前馈传播和反向传播过程，多层感知机可以逐渐调整其权重和偏差，从而提高对输入数据的表示能力和泛化能力。

4.2 实验设计

4.2.1 数据处理

读入数据，标签采用独热编码（1 of K）。

```
254     train_images, train_labels, test_images, test_labels = data_prepare()
255     train_label = np.zeros((train_images.shape[0], 10))
256     test_label = np.zeros((test_images.shape[0], 10))
257     for i in range(len(train_images)):
258         train_label[i][train_labels[i]] = 1
259     for i in range(len(test_images)):
260         test_label[i][test_labels[i]] = 1
```

图 4 数据初始化代码截图

4.2.2 权重初始化

使用 np.random.randn 方法随机初始化神经元之间的连接矩阵w和偏置b，同时初始化各层神经元为$h^{(i)}$以保存中间结果（包含输入输出层）

```
51      def init_params(self):
52          '''初始化权重和偏置'''
53          self.weights = []
54          self.biases = []
55          self.h = []
56          for i in range(self.layers - 1):
57              self.weights.append(np.random.randn(self.arch[i+1], self.arch[i]))
58              self.biases.append(np.random.randn(self.arch[i+1], 1))
59              self.h.append(np.zeros((self.arch[i], 1)))
60
61          self.h.append(np.zeros((self.arch[-1], 1)))
62          for j in range(len(self.h)):
63              print(f'第{j}层的神经元数目为',self.h[j].shape)
```

图 5 权重初始化代码截图

4.2.3 前向传播

将输入w赋值给$h^{(0)}$，随后逐层计算$a\left(W^{(t)}h^{(t-1)}+b^{(t)}\right)$，使用函数类保存中间结果，同时返回最终输出层$h^{(s)}$。实验激活函数采用 sigmoid 和 softmax 函数。

```
85      def forward(self, x):
86          '''前向传播,给定输入x,计算输出y'''
87          self.h[0] = x
88          for i in range(1, self.layers):
89              self.h[i] = self.sigmoid(np.dot(self.weights[i-1], self.h[i-1]) + self.biases[i-1])
90          self.h[-1] = self.softmax(np.dot(self.weights[-1], self.h[-2]) + self.biases[-1])
91          return self.h[-1]
```

图 6 前向传播代码截图

4.2. 反向传播

给定最终标签计算当前预测值的损失和梯度$\delta^{(s)} = h^{(s)} - y'$。随后按$W_{\text{new}}^{(t)} \leftarrow W^{(t)} - \eta\nabla_{W^{(t)}L}, b_{\text{new}}^{(t)} \leftarrow b^{(t)} - \eta\nabla_{b^{(t)}L}, \delta^{(t-1)} = \frac{\partial a}{\partial z_j^{(t-1)}} \odot \left(W^{(t)^T} \cdot \delta^{(t)}\right)$依层记录梯度和参数并返回。

```python
def backward(self, y, learning_rate):
    '''反向传播,给定输入x和标签y,计算梯度'''
    loss, prime = self.cross_entropy_loss(y, self.h[-1])
    weights, biases = [], []
    for i in range(self.layers - 1):
        weights.append(np.zeros((self.arch[i+1], self.arch[i])))
        biases.append(np.zeros((self.arch[i+1], 1)))
    for t in range(self.layers - 1, 0, -1):
        grad_w = np.dot(prime, self.h[t-1].T)
        grad_b = prime
        # 更新参数
        weights[t-1] = grad_w
        biases[t-1] =  grad_b
        # 更新梯度
        prime = np.dot(self.weights[t-1].T, prime) * self.sigmoid_prime(self.h[t-1])

    return loss, weights, biases
```

图 7 反向传播代码截图

4.2.5 随机梯度下降

使用 random.shuffle 方法打乱数据，每次取数据中的前 batch_size 组使用前向传播和反向传播计算更新梯度和偏置，随后取算数平均对参数进行更新。

```python
def SGD(self, training_data, batch_size, eta):
    '''随机梯度下降法训练神经网络'''
    random.shuffle(training_data)
    data = training_data[:batch_size]
    loss, weight, biase = [], [], []
    for img,label in data:
        img = img.reshape((-1, 1))
        label = label.reshape((-1, 1))
        y = self.forward(img)
        los, grad_w, grad_b = self.backward(label, eta)
        loss.append(los)
        weight.append(grad_w)
        biase.append(grad_b)
    # 更新参数
    for j in range(len(weight)):
        for i in range(self.layers - 1):
            self.weights[i] -= eta * weight[j][i] / batch_size
            self.biases[i] -= eta * biase[j][i] / batch_size
    return np.mean(loss)
```

图 8 SGD 代码截图

随后设置迭代次数，重复上述步骤，即可训练得到可以分类的神经网络。

4.2.6 Softmax 回归

使用和逻辑回归相似的 Softmax 回归构造线性分类器，和逻辑回归唯一不同的地方在于，由于此问题中对应的是多分类问题，因此 Softmax 回归最后改用 Softmax()替代 Sigmoid()进行概率的选择。其余交叉熵损失函数及梯度下降求导的式子完全一致。

4.3 实验结果及分析

4.3.1 实验结果

在本次实验中，使用神经元数目分别为784,100,10的三层感知机，采用 Sigmoid 函数作为激活函数和 Softmax 函数对 MNIST 数据集进行分类。

使用交叉熵损失函数和 SGD 优化器，将模型输入通道根据数据集设为 1，并设置训练超参数 epoch 为 100，batch size 为 256（相当于只用了 25600 组训练数据），学习率 learning rate 为 0.1。训练过程中损失函数 loss 的值和在测试集上的准确率变化如下图所示。

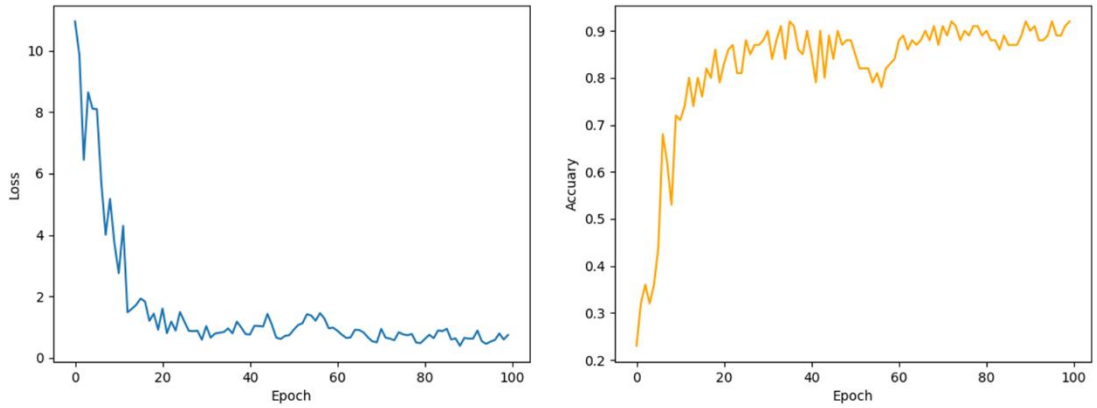


图 9 多层感知机训练结果

实验发现，随训练过程的进行，损失函数不断降低，在测试集上准确率逐渐升高，最终测试正确率最高能够达到约 92%。损失函数和测试准确率在训练最后阶段呈现波动态，可能原因是在局部最优点附近振荡。

图 10 测试集上预测结果

4.3.2 与线性分类器的对比

在本次实验中，使用 Softmax 回归对 MNIST 数据集进行分类。同样地，使用交叉熵损失函数和 SGD 优化器，将模型输入通道根据数据集设为 1，并设置训练超参数 epoch 为 100，batch size 为 256，学习率 learning rate 为 0.1。训练过程中损失函数 loss 的值和在测试集上的准确率变化如下图所示。

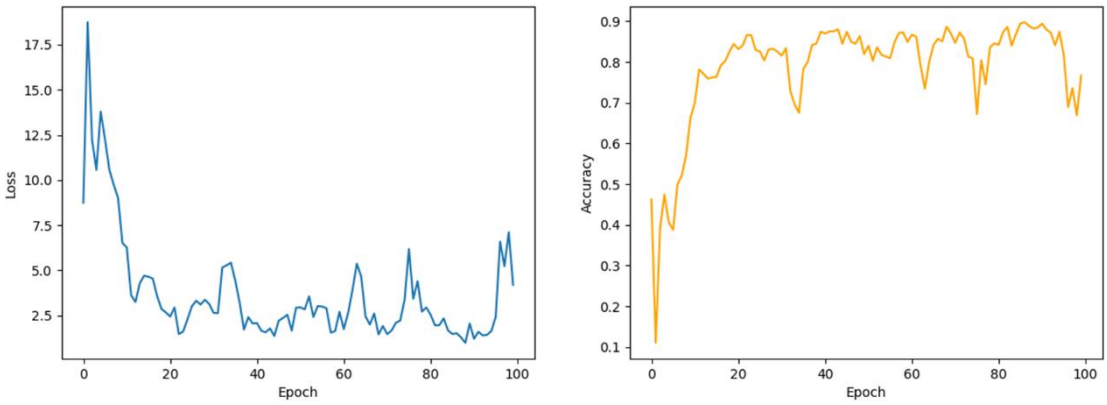


图 11 线性分类器训练结果（lr = 0.1，epoch = 100）

将学习率设为 0.001，epoch 设为 1000，训练过程中损失函数 loss 的值和在测试集上的准确率变化如下图所示

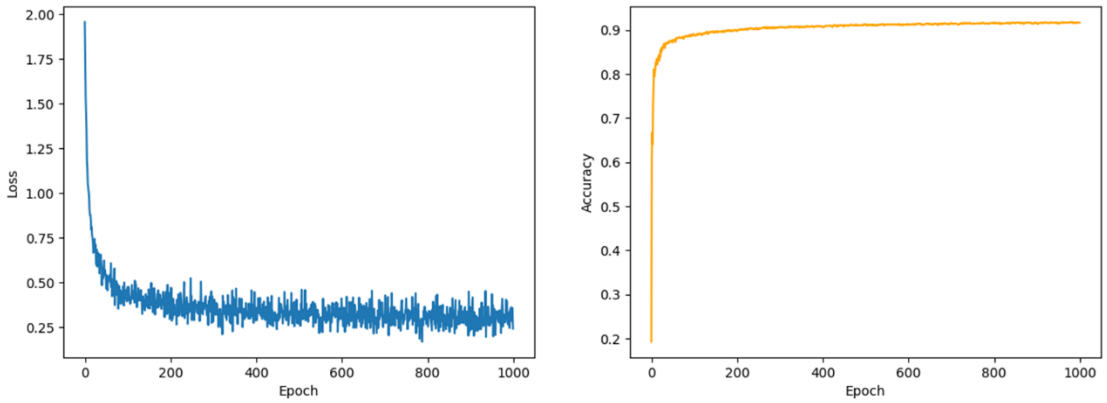
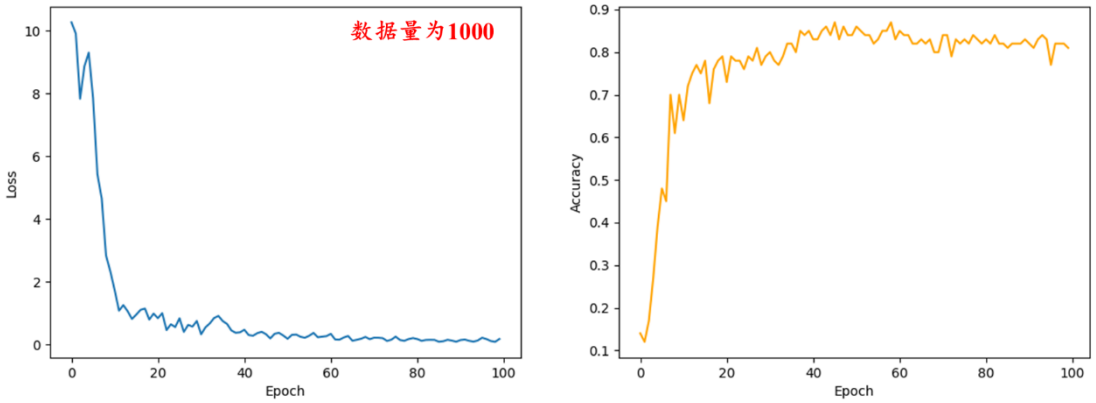


图 12 线性分类器训练结果（lr = 0.001，epoch = 1000）

可以发现，在同样的参数设置条件下，Softmax 回归训练过程中损失函数震荡幅度大，较难收敛，且由于没有引入非线性因素的原因，最高准确率相较于多层感知机要低，约 88%左右。但是 Softmax 回归的训练速度快于多层感知机，这是因为模型参数更少，并且反向传播没有复杂的传递过程。

4.3.3 不同数据量下的数据结果

分别设置数据量为 10000 和 1000，多层感知机结构和超参数均不变，得到实验结果如下图：



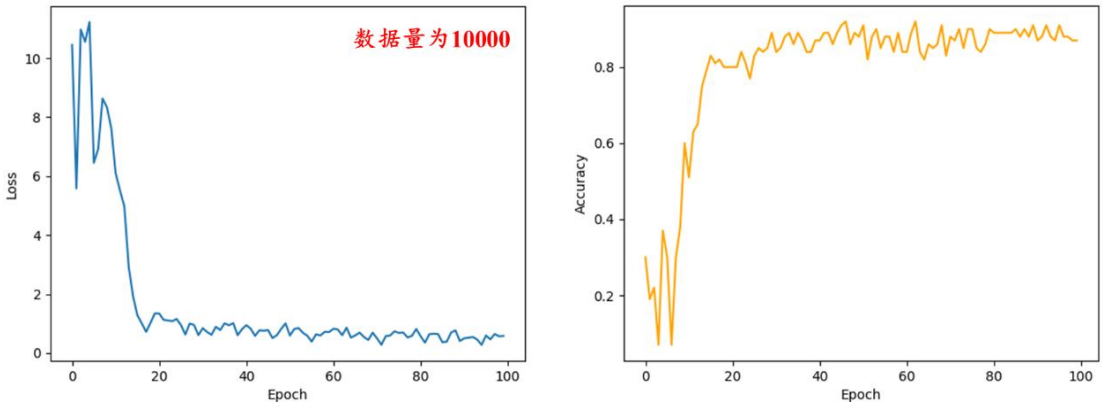


图 12 不同数据量对比（MLP）

同样地，将回归的数据量减少至 10000，分别在学习率 0.01 和 0.1 的情况下进行测试结果如下：

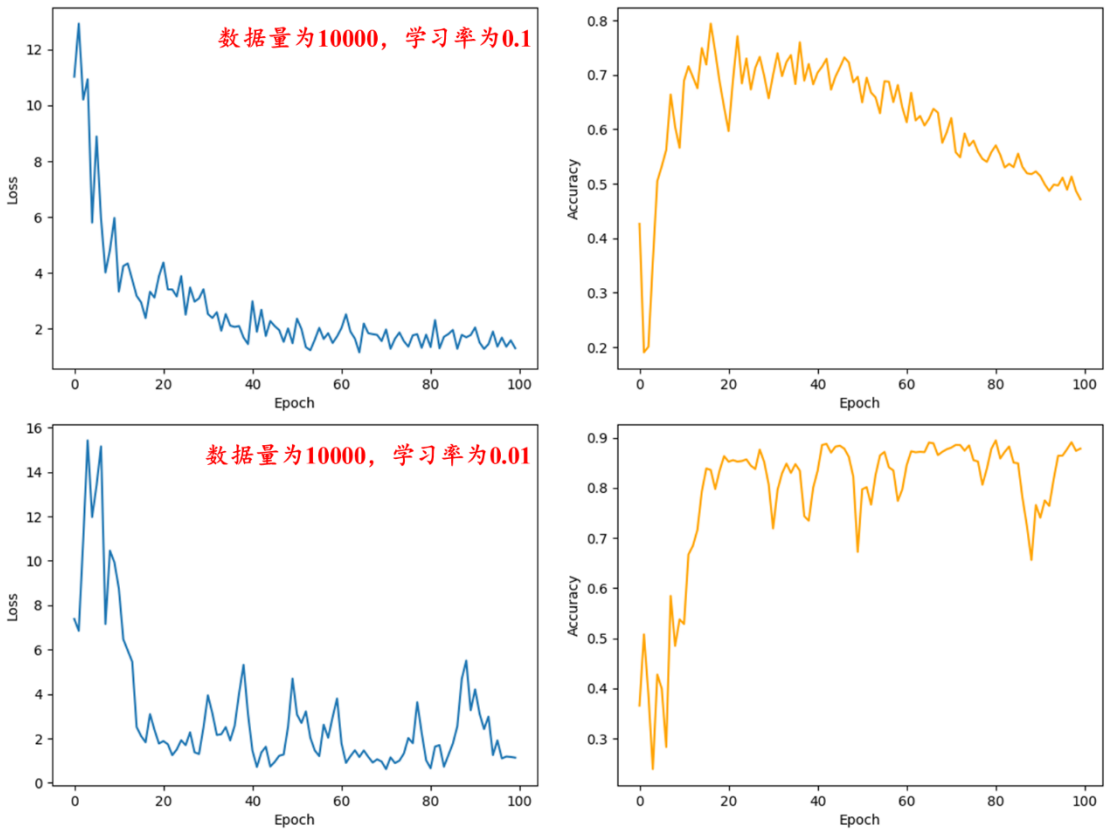


图 13 不同数据量对比（Softmax 回归）

可以发现，数据量减少对多层感知机的影响不大，正确率略有下降，而在 Softmax 回归上在学习率较大时会出现过拟合的现象。这应该是由于回归的参数量较小，相较于 MLP 更容易受训练数据的影响。

4.3.4 不同超参数下的数据结果

针对多层感知机，分别修改其**学习率、训练轮数和网络结构**等超参数，观察实验结果如下：

a) **学习率：**

设置学习率为 0.01，可以发现，降低学习率，准确率有一定下降，这是由于学习率较小，参数更新较慢，而 MLP 参数较多，在较小的学习率上需要更多的迭代次数才能得到较好的结果。

在此情形下，增大训练轮数会有一定效果，但这会大大加长训练时间，因此在实际研究中需要选择一个合适的学习率。

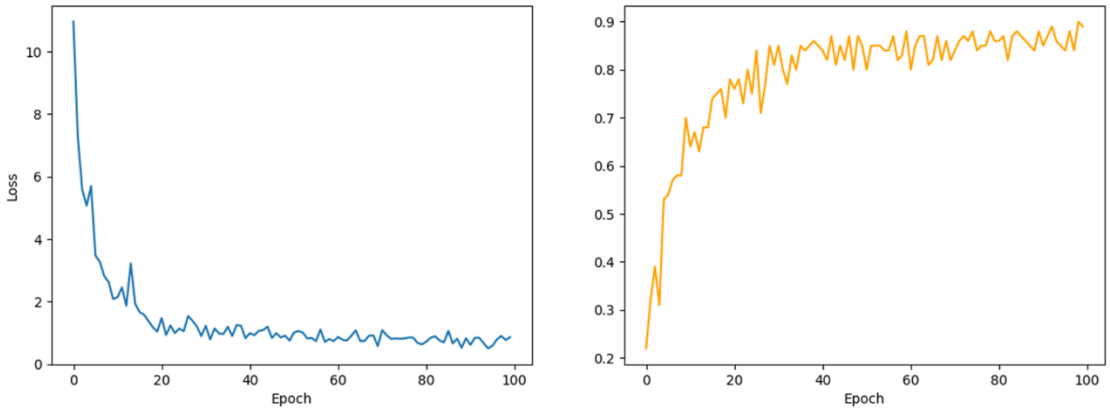


图 14 多层感知机训练结果（lr = 0.01）

b) **训练轮数：**

设置 epoch 为 300，可以发现，增大训练轮数，准确率能够有一定提升，能够达到 95%左右，这是因为 MLP 具有较强的表达能力，包含多个隐藏层，每个隐藏层都可以学习到不同层次的特征表示。但一般很难收敛，因此增加训练轮数能够有效提高模型性能，但也会提高模型的训练开销。

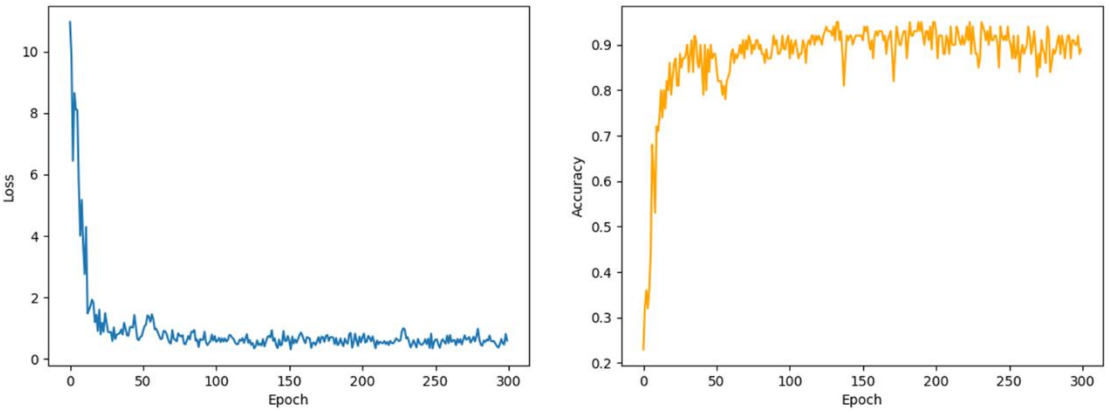


图 15 多层感知机训练结果（epoch = 300）

c) **网络结构：**

修改感知机结构为784,300,10，即加大隐藏层的神经元数量，在同样的训练轮数和学习率下，测试结果如下：

可以发现，增加隐藏层的神经元数能够一定程度上提升 MLP 的准确率，同时运行时间也会有所加长。这是因为每个隐藏层神经元的数量代表了该层的维度，增加维度可以提高模型的表达能力，使网络更加灵活，学习到更多种类的特征。但由于参数增加也会导致损失函数震荡较难收敛等问题。

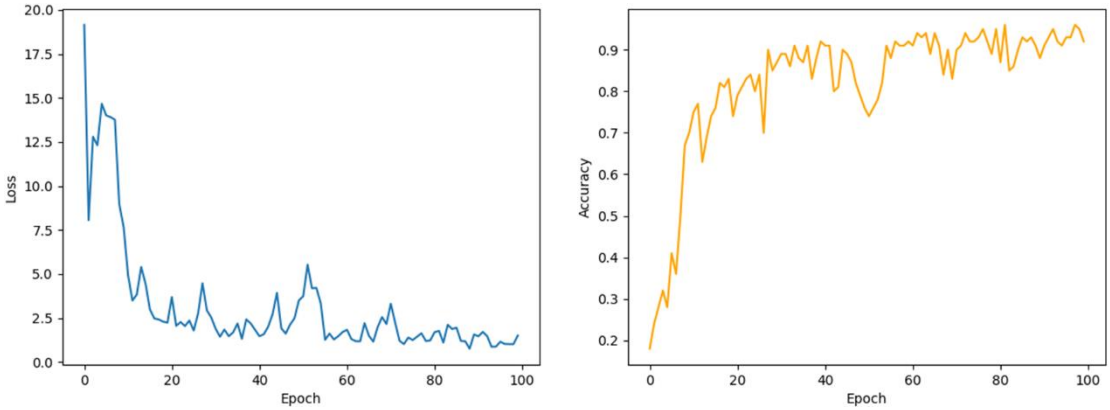


图 16 多层感知机训练结果（结构为784,300,10）

修改感知机结构为784,100,50,10，即加深神经网络的层数，在同样的训练轮数和学习率下，测试结果如下：

可以发现，当增加隐藏层层数时准确率却下降了，这可能因为随着层数的增加，梯度在反向传播过程中可能出现梯度消失或梯度爆炸的问题，导致模型难以训练。这个问题可以通过权重初始化方法（如 Xavier 或 Kaiming_He 初始化）或者改用其他激活函数（如 ReLU 等）来缓解。

但最好的方法还是引入更加高效的模型结构，如残差模块等，能够有效减少深度学习中较深层网络中的恒等映射导致的梯度消失或爆炸问题。

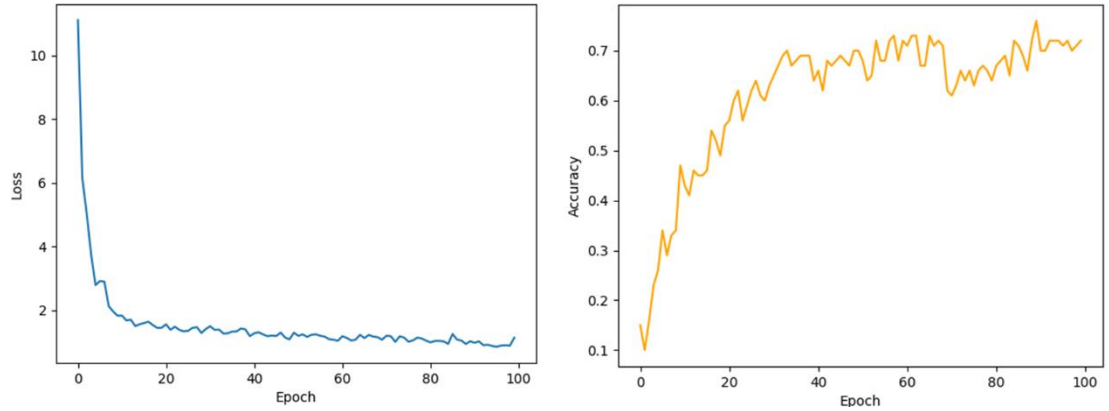


图 17 多层感知机训练结果（结构为784,100,50,10）

5、 实验结论

多层感知机模型是矩阵与向量的乘积的非线性变换的多次重复，其核心在与引入了**非线性因素**，能够学习和捕捉复杂的非线性关系，其基本结构较为简单，其具有较强的表达能力，可适应图像分类、识别等多种人工智能任务。

Softmax 回归相当于 2 层（只有输入输出层）的多层感知机，没有引入非线性因素，在一定程度上对数据更为敏感，且对非线性问题表达能力较弱。

选择合适的学习率能够减少模型训练的时间，但梯度下降法较难收敛，提高训练轮次可能会提供模型的能力。同时，合适的权重初始化也能减少模型的训练时间和提高模型的训练效果。

但是，单纯加深模型深度没有特别大的意义，除了训练时间会增加，还可能会出现梯度消失或者梯度爆炸等问题，可以考虑引入残差等结构。

6、 完整实验代码

MLP.py

```
import numpy as np
import matplotlib.pyplot as plt
import random
import pandas as pd
from sklearn.model_selection import train_test_split
import struct

def random_seed(seed):
    np.random.seed(seed)
    random.seed(seed)

def load_mnist_images(filename):
    with open(filename, 'rb') as file:
        _, num_images, rows, cols = struct.unpack('>IIII', file.read(16))
        images = np.fromfile(file, dtype=np.uint8).reshape(num_images, rows * cols)
    return images

def load_mnist_labels(filename):
    with open(filename, 'rb') as file:
        _, num_labels = struct.unpack('>II', file.read(8))
        labels = np.fromfile(file, dtype=np.uint8)
    return labels

def data_prepare():
    train_images = load_mnist_images('./data/train-images-idx3-ubyte')
    train_labels = load_mnist_labels('./data/train-labels-idx1-ubyte')
    test_images = load_mnist_images('./data/t10k-images-idx3-ubyte')
    test_labels = load_mnist_labels('./data/t10k-labels-idx1-ubyte')

    return train_images, train_labels, test_images, test_labels

def iris():
    data = pd.read_csv('./Iris.data')
```

```
34.         data["Species"] = data["Species"].map({"Iris-
    setosa": 1, "Iris-versicolor": 0, "Iris-virginica": 2})
35.         data = np.array(data[:].values)
36.
37.         characters = data[:, :-1].astype(float)
38.         labels = data[:, -1].astype(float)
39.
40.
41.         X_train, X_test, y_train, y_test = train_test_split(charact
    ers, labels, test_size=0.2, random_state=10)
42.         return X_train, X_test, y_train, y_test
43.
44.     def visualize_images_with_labels(images, true_labels, predicted
    _labels):
45.         image = images[:10]
46.         true_label = true_labels[:10]
47.         predicted_label = predicted_labels[:10]
48.
49.         _, axes = plt.subplots(2, 5, figsize=(10, 4))
50.
51.         for i, ax in enumerate(axes.flat):
52.             ax.imshow(image[i].reshape(28, 28), cmap='gray')
53.             ax.axis('off')
54.             ax.text(0.5, -
    0.1, f'True: {true_label[i].argmax()}', transform=ax.transAxes,
55.                     horizontalalignment='center', verticalalignment
    ='center', color='black')
56.             ax.text(0.5, -
    0.2, f'Predicted: {predicted_label[i]}', transform=ax.transAxes
    ,
57.                     horizontalalignment='center', verticalalignment
    ='center', color='red')
58.         plt.suptitle('MNIST Images With Labels after MLP', fontsize
    =16)
59.         plt.show()
60.
61.     class MLP(object):
62.         def __init__(self, arch, layers):
63.             self.arch = arch
64.             self.layers = layers
65.
66.         def init_params(self):
67.             '''初始化权重和偏置'''
68.             self.weights = []
```

```
69.             self.biases = []
70.             self.h = []
71.             for i in range(self.layers - 1):
72.                 self.weights.append(np.random.randn(self.arch[i+1],
       self.arch[i]))
73.                 self.biases.append(np.random.randn(self.arch[i+1],
      1))
74.                 self.h.append(np.zeros((self.arch[i], 1)))
75.
76.             self.h.append(np.zeros((self.arch[-1], 1)))
77.             for j in range(len(self.h)):
78.                 print(f'第{j}层的神经元数目为',self.h[j].shape)
79.
80.         # 激活函数
81.         def sigmoid(self, z):
82.             z = np.clip(z, -500, 500)
83.             return 1 / (1 + np.exp(-z))
84.
85.         def sigmoid_prime(self, z):
86.             return self.sigmoid(z) * (1 - self.sigmoid(z))
87.
88.         def softmax(self, z):
89.             z = np.clip(z, -500, 500)
90.             return np.exp(z) / np.sum(np.exp(z))
91.
92.
93.         def cross_entropy_loss(self, y_true, y_pred):
94.             '''计算多分类交叉熵损失'''
95.             y_pred = np.clip(y_pred, 1e-10, 1 - 1e-10)
96.             loss = -np.sum(y_true * np.log(y_pred))
97.             prime = y_pred - y_true
98.             return loss, prime
99.
100.        def forward(self, x):
101.            '''前向传播,给定输入 x,计算输出 y'''
102.            self.h[0] = x
103.            for i in range(1, self.layers):
104.                self.h[i] = self.sigmoid(np.dot(self.weights[i-
      1], self.h[i-1]) + self.biases[i-1])
105.            self.h[-1] = self.softmax(np.dot(self.weights[-
      1], self.h[-2]) + self.biases[-1])
106.            return self.h[-1]
107.
108.        def backward(self, y, learning_rate):
```

```
109.             '''反向传播,给定输入 x 和标签 y,计算梯度'''
110.             loss, prime = self.cross_entropy_loss(y, self.h[-1])
111.             weights, biases = [], []
112.             for i in range(self.layers - 1):
113.                 weights.append(np.zeros((self.arch[i+1], self.arch[
    i])))
114.                 biases.append(np.zeros((self.arch[i+1], 1)))
115.             for t in range(self.layers - 1, 0, -1):
116.                 grad_w = np.dot(prime, self.h[t-1].T)
117.                 grad_b = prime
118.                 # 更新参数
119.                 weights[t-1] = grad_w
120.                 biases[t-1] =  grad_b
121.                 # 更新梯度
122.                 prime = np.dot(self.weights[t-
    1].T, prime) * self.sigmoid_prime(self.h[t-1])
123.
124.             return loss, weights, biases
125.
126.         def SGD(self, training_data, batch_size, eta):
127.             '''随机梯度下降法训练神经网络'''
128.             random.shuffle(training_data)
129.             data = training_data[:batch_size]
130.             loss, weight, biase = [], [], []
131.             for img,label in data:
132.                 img = img.reshape((-1, 1))
133.                 label = label.reshape((-1, 1))
134.                 y = self.forward(img)
135.                 los, grad_w, grad_b = self.backward(label, eta)
136.                 loss.append(los)
137.                 weight.append(grad_w)
138.                 biase.append(grad_b)
139.             # 更新参数
140.             for j in range(len(weight)):
141.                 for i in range(self.layers - 1):
142.                     self.weights[i] -
    = eta * weight[j][i] / batch_size
143.                     self.biases[i] -
    = eta * biase[j][i] / batch_size
144.             return np.mean(loss)
145.
146.         def train(self, data, epochs, batch_size, eta, test_data):
147.             '''训练神经网络'''
148.             self.init_params()
```

```
149.          loss = []
150.          acc = []
151.          random.shuffle(data)
152.          for i in range(epochs):
153.              # train_data = data[:batch_size]
154.              if (i*batch_size) % len(data) > ((i+1)*batch_size)
      % len(data):
155.                  train_data = data[(i*batch_size) % len(data):]
      + data[:(i+1)*batch_size % len(data)]
156.              else:
157.                  train_data = data[(i*batch_size) % len(data):((
      i+1)*batch_size) % len(data)]
158.              loss.append(self.SGD(train_data, batch_size, eta))
159.              acc.append(self.evaluate(test_data))
160.              print(f"Epoch {i}: {loss[i]}, Accuary: {acc[i]}")
161.          plt.figure()
162.          plt.plot(loss)
163.          plt.xlabel("Epoch")
164.          plt.ylabel("Loss")
165.          plt.show()
166.
167.          plt.figure()
168.          plt.plot(acc, color='orange')
169.          plt.xlabel("Epoch")
170.          plt.ylabel("Accuracy")
171.          plt.show()
172.
173.      def evaluate(self, test_data):
174.          '''评估神经网络的性能'''
175.          num = 0
176.          test_data = test_data[:100]
177.          test_img, test_label = zip(*test_data)
178.          pred_label = []
179.          for x,y in test_data:
180.              x = x.reshape((-1, 1))
181.              y = y.reshape((-1, 1)).argmax()
182.              y_pred = self.forward(x).argmax()
183.              pred_label.append(y_pred)
184.              if y_pred == y:
185.                  num += 1
186.          #visualize_images_with_labels(test_img, test_label, pre
      d_label)
187.          return num / len(test_data)
188.
```

```
189.  # 多项回归模型
190.  class LogisticRegression:
191.      def __init__(self, learning_rate, bacth, num_epochs):
192.          self.learning_rate = learning_rate
193.          self.num_epochs = num_epochs
194.          self.theta = None
195.          self.batch = bacth
196.          self.loss_history = []
197.          self.acc_history = []
198.
199.      def softmax(self, z):
200.          z = np.clip(z, -500, 500)
201.          for i in range(len(z)):
202.              z[i] = np.exp(z[i]) / np.sum(np.exp(z[i]))
203.          return z
204.
205.      def loss_func(self, X, y, theta):
206.          '''计算损失函数'''
207.          y_pred = self.softmax(np.dot(X, theta))
208.          y_pred = np.clip(y_pred, 1e-10, 1 - 1e-10)
209.          loss = -np.sum(y * np.log(y_pred)) / len(X)
210.          return loss
211.
212.      def gredient_decent(self, X, y, test_images, test_label, la
      mda=0):
213.          '''梯度下降法'''
214.          train_data = list(zip(X, y))
215.          self.theta = np.zeros((X.shape[1], y.shape[1]))
216.          random.shuffle(train_data)
217.          for epoch in range(self.num_epochs):
218.              if (epoch*self.batch) % len(train_data) > ((epoch+1
      )*self.batch) % len(train_data):
219.                  data = train_data[(epoch*self.batch) % len(trai
      n_data):] + train_data[:(epoch+1)*self.batch % len(train_data)]
220.              else:
221.                  data = train_data[(epoch*self.batch) % len(trai
      n_data):((epoch+1)*self.batch) % len(train_data)]
222.              X_batch, y_batch = zip(*data)
223.              X_batch = np.array(X_batch)
224.              y_batch = np.array(y_batch)
225.              y_pred = self.softmax(np.dot(X_batch, self.theta))
226.              # 计算梯度
227.              gradient = np.dot(X_batch.T, (y_pred - y_batch)) /
      len(X) + lamda * self.theta
```

```
228.                 # 更新权重
229.                 self.theta -= self.learning_rate * gradient
230.                 loss = self.loss_func(X_batch, y_batch, self.theta)
231.                 acc = self.evaluate(test_images, test_label)
232.                 print(f"第{epoch+1}次迭代，损失函数为{loss}，准确率为
    {acc}")
233.                 self.loss_history.append(loss)
234.                 self.acc_history.append(acc)
235.
236.     def predict(self, X):
237.         '''预测'''
238.         y_pred = self.softmax(np.dot(X, self.theta))
239.         y_pred = np.argmax(y_pred, axis=1)
240.         return y_pred
241.
242.     def evaluate(self, X, y):
243.         '''评估模型性能'''
244.         num = 0
245.         y_pred = self.predict(X)
246.         y_ture = np.argmax(y, axis=1)
247.         for i in range(len(y_pred)):
248.             if y_pred[i] == y_ture[i]:
249.                 num += 1
250.         acc = num / len(y_pred)
251.         return acc
252.
253.     def draw_loss(self):
254.         '''绘制损失函数变化曲线'''
255.         plt.plot(self.loss_history)
256.         plt.xlabel('Epoch')
257.         plt.ylabel('Loss')
258.         plt.show()
259.
260.     def draw_acc(self):
261.         '''绘制准确率变化曲线'''
262.         plt.plot(self.acc_history, color='orange')
263.         plt.xlabel('Epoch')
264.         plt.ylabel('Accuracy')
265.         plt.show()
266.
267. def regression():
268.     # X_train, X_test, y_train, y_test = iris()
269.     # train_label = np.zeros((X_train.shape[0], 3))
270.     # test_label = np.zeros((X_test.shape[0], 3))
```

```
271.      # for i in range(len(X_train)):
272.      #     train_label[i][int(y_train[i])] = 1
273.      # for i in range(len(X_test)):
274.      #     test_label[i][int(y_test[i])] = 1
275.      # Regression = LogisticRegression(0.01, 120, 2000)
276.      # Regression.gredient_decent(X_train, train_label, lamda=0)
277.      # Regression.draw_loss()
278.      # acc = Regression.evaluate(X_test, test_label)
279.      # print(f"Accuracy: {acc}")
280.
281.      train_images, train_labels, test_images, test_labels = data
      _prepare()
282.      train_label = np.zeros((train_images.shape[0], 10))
283.      test_label = np.zeros((test_images.shape[0], 10))
284.      # random.shuffle(train_images)
285.      train_images = train_images[:60000]
286.      for i in range(len(train_images)):
287.          train_label[i][train_labels[i]] = 1
288.      for i in range(len(test_images)):
289.          test_label[i][test_labels[i]] = 1
290.      Regression = LogisticRegression(0.001, 256, 1000)
291.      Regression.gredient_decent(train_images, train_label, test_
      images, test_label, lamda=0)
292.      Regression.draw_loss()
293.      Regression.draw_acc()
294.      acc = Regression.evaluate(test_images, test_label)
295.      print(f"Accuracy: {acc}")
296.
297.  if __name__ == '__main__':
298.      # regression()
299.
300.      random_seed(10)
301.      train_images, train_labels, test_images, test_labels = data
      _prepare()
302.      train_label = np.zeros((train_images.shape[0], 10))
303.      test_label = np.zeros((test_images.shape[0], 10))
304.      train_images = train_images[:60000]
305.      for i in range(len(train_images)):
306.          train_label[i][train_labels[i]] = 1
307.      for i in range(len(test_images)):
308.          test_label[i][test_labels[i]] = 1
309.      train_data = list(zip(train_images, train_label))
310.      test_data = list(zip(test_images, test_label))
311.      # 构建神经网络
```

```
312.        mlp = MLP([784, 100, 10],3)
313.        mlp.train(train_data, 100, 256, 0.1, test_data)
314.        acc = mlp.evaluate(test_data)
315.        print(f"Accuracy: {acc}")
316.
317.        # X_train, X_test, y_train, y_test = iris()
318.        # train_label = np.zeros((X_train.shape[0], 3))
319.        # test_label = np.zeros((X_test.shape[0], 3))
320.        # for i in range(len(X_train)):
321.        #     train_label[i][int(y_train[i])] = 1
322.        # for i in range(len(X_test)):
323.        #     test_label[i][int(y_test[i])] = 1
324.        # train_data = list(zip(X_train, train_label))
325.        # test_data = list(zip(X_test, test_label))
326.        # # 构建神经网络
327.        # mlp = MLP([4, 50, 3], 3)
328.        # mlp.train(train_data, 100, 120, 0.01)
329.        # acc = mlp.evaluate(test_data)
330.        # print(f"Accuracy: {acc}")
```

7、 参考文献

[1] 李航. 统计学习方法[M]. 清华大学出版社, 2012.

[2] 周志华. 机器学习[M]. 清华大学出版社, 2016.