《最优化技术》实验报告

一、实验目的

理解梯度下降法，构建人工神经网络，并能应用于求解实际问题。

二、实验项目内容

构建一个神经网络，利用梯度下降法实现参数的更新，最终实现对 0-9 的 10 个手写数字的识别。

其中，MNIST 数据集可在 http://yann.lecun.com/exdb/mnist/获取，它包含四个部分:

• Training set images: train-images-idx3-ubyte.gz (9.9 MB，解压后 47 MB，包含 60,000 个样本)**（训练样本的图片，每张图片以像素值矩阵呈现）**

• Training set labels: train-labels-idx1-ubyte.gz (29 KB，解压后 60 KB，包含 60,000 个标签)**（训练样本的真实值，即 0~9）**

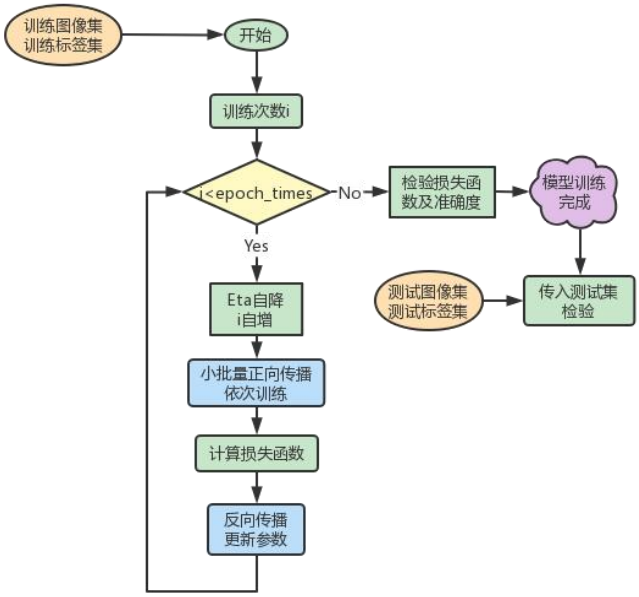
• Test set images: t10k-images-idx3-ubyte.gz (1.6 MB，解压后 7.8 MB，包含 10,000 个样本)**（测试样本的图片）**

• Test set labels: t10k-labels-idx1-ubyte.gz (5KB，解压后 10 KB，包含 10,000 个标签)**（测试样本的真实值）**

注意：所有程序请用 python 语言实现。只提交本电子文档，注意本文件末尾的文件命名要求；源程序一节请用代码备注的方式说明你的算法和思路；实验结果一节需要提供测试结果截图并给出结果分析。

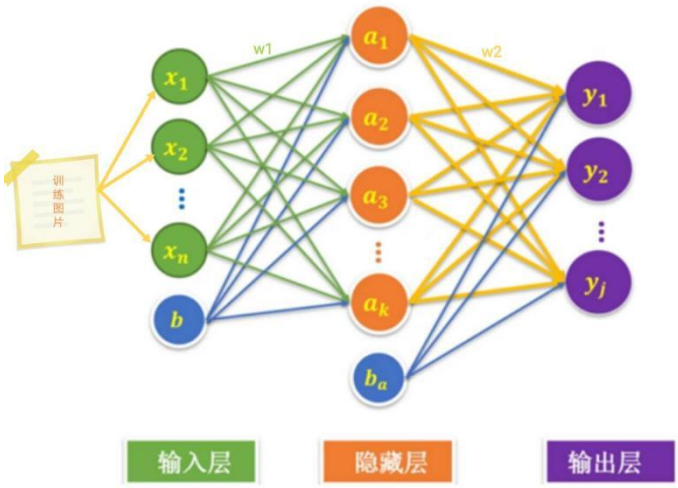
三、实验过程或算法（源程序）

1. 人工神经网络算法流程图



报告创建时间：2022.5.16

2. 一个样本的正向传播训练图



3. 代码

（读取压缩包数据集以及显示像素数组的图像代码均来源报告模板里的参考代码）

（类定义部分参考 Github：https://github.com/Jstar49/Neural-Network）

```
'''Neural-Network
    构建一个神经网络,利用梯度下降法实现参数的更新,最终实现对0-9的10个手写数字的识别
实现神经网络算法识别手写数字集。
'''
import os
import sys
import gzip
import struct
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import expit

def load_mnist_train(path,kind='train'):    #加载训练集文件
    #path:数据集的路径
    #kind:值为train,代表读取训练集(若为't10k'则为测试集)
    #os用于文件路径拼接
    labels_path=os.path.join(path,'%s-labels-idx1-ubyte.gz'%kind)  #样本标签,即真实值
    images_path=os.path.join(path,'%s-images-idx3-ubyte.gz'%kind)  #样本图片
    #使用gzip读取文件(数据集里的图片,以二进制形式存储),'rb'表示读取二进制数据
    #lbpath即为读取到的二进制数据,使用np.fromstring将其转化成numpy需要的数据形式
    with gzip.open(labels_path,'rb') as lbpath:
        #使用struct.unpack方法读取前两个数据,>代表高位在前,I代表32位整型。
```

```
        #lbpath.read(8)表示一次从文件中读取 8 个字节(两个数据)
        #magic number(调用 fromfile 将字节读入 NumPy array 之前在文件缓冲
中的 item 数)和样本数 n
        magic,n=struct.unpack('>II',lbpath.read(8))
        #lbpath.read()表示读取剩下的所有的数据
        labels=np.frombuffer(lbpath.read(),dtype=np.uint8)
    with gzip.open(images_path,'rb') as imgpath:
        magic,num,rows,cols=struct.unpack('>IIII',imgpath.read(16))
        #28*28=784,即为每个样本图片的所有像素点
        images=np.frombuffer(imgpath.read(),dtype=np.uint8).reshape(
len(labels),784)
    '''函数返回两个数组,第一个是一个 n*m 维的 Numpy 数组(images),这里的 n 是
样本数,m 是特征数。训
    练数据集包含 60,000 个样本,测试数据集包含 10000 样本。在 MNIST 数据集中的
每张图片由 28*28 个像
    素点构成,每个像素点用一个灰度值表示,因此 images 每行为一个包含 784 个灰
度值的数组。函数返回
    的第二个数组(labels)包含了相应的目标变量,也就是手写数字的类标签(整数
0~9)'''
    return images,labels
# X_train,y_train=load_mnist_train('OptimizationTechnology\实验
4\MNIST','t10k')
# X_train,y_train=load_mnist_train('OptimizationTechnology\实验
4\MNIST')

def show_images_09(X_train,y_train):        #显示数据集中第一个 0~9 的图
像
    fig,ax=plt.subplots(nrows=2,ncols=5,sharex=True,sharey=True)
    #将 ax 由 n*m 的 Axes 组展平成 1*nm 的 Axes 组,便于循环遍历
    ax=ax.flatten()
    for i in range(10):
        #y_train==i 返回一个 boolen 型数组,X_train[y_train==i]为所有满足
y_train==i 的数据集图像
        #img 选取 X_train 中第一个等于 i 的图像
        img=X_train[y_train==i][0].reshape(28,28)
        #生成热图,主题为“灰色”
        #图像放大算法:最近邻法(Nearest Interpolation)
        ax[i].imshow(img,cmap='Greys',interpolation='nearest')
    ax[0].set_xticks([])
    ax[0].set_yticks([])
    plt.tight_layout()
    plt.show()
# show_images(X_train,y_train)
def show_images_x(X_train,y_train,x,n):     #显示数字 x 的前 n 个不同图像
(请保证 n%5=0)
    fig,ax=plt.subplots(nrows=5,ncols=n//5,sharey=True,sharex=True)
    ax=ax.flatten()
```

```
    for i in range(n):
        img=X_train[y_train==x][i].reshape(28,28)
        ax[i].imshow(img,cmap='Greys',interpolation='nearest')
    ax[0].set_xticks([])
    ax[0].set_yticks([])
    plt.tight_layout()
    plt.show()
# show_images_x(X_train,y_train,5,30)

class Neural_Nets(object):
    def __init__(self,n_input,n_output,n_hidden,
        L1=0.0,L2=0.0,epoch_times=1000,
        Eta=0.001,alpha=0.0,de_Eta=0.0,
        shuffle=True,n_minibatch=1,random_seed=1):
        self.n_input=n_input            #输入层特征数
        self.n_output=n_output          #输出层特征数
        self.n_hidden=n_hidden          #隐藏层特征数
        self.w1,self.w2=self.Init_wgt() #前后两层间的权重 w(包括偏置 b)
        self.L1=L1                      #L1 正则化系数(降低过拟合程度)
        self.L2=L2                      #L2 正则化系数
        self.epoch_times=epoch_times    #迭代次数
        self.Eta=Eta                    #学习率
        self.alpha=alpha
        self.de_Eta=de_Eta              #随迭代次数增加而降低 Eta 的常数
        self.shuffle=shuffle            #每次迭代前打乱训练集的顺序以防
陷入死循环
        self.n_minibatch=n_minibatch    #小批量的每批的数量
        np.random.seed(random_seed)     #用随机数种子生成随机数

    def Encode_label(self,y,n_out):     #解码标签集 y,每个数字对应的图片
        #y 为读取的训练集标签集(60000*1),n_out 为输出层特征数
        labels=np.zeros((n_out,y.shape[0])) #n_out 数(0~9 这 10 个数)作
为行数,y 的行数(60000)作为列数
        for idx,val in enumerate(y):        #顺序枚举 y 中元素,idx 为序号
(0~59999)
            labels[val,idx]=1.0             #这里
        return labels                   #返回 10*60000 的 np 数组,描述每个
数字对应的哪些图片
    def Add_bias(self,X,how='column'):  #为图像集 X 增加一行/列
        #X 为读取的训练集图像集
        #虽然 Init_wgt 中已经生成随机偏置,但矩阵相乘时需要列行数对应,因此将
X 扩一列(行)1
        if how=='column':
            X_new=np.ones((X.shape[0],X.shape[1]+1))
            X_new[:,1:]=X
        elif how =='row':
            X_new=np.ones((X.shape[0]+1,X.shape[1]))
```

```
            X_new[1:,:]=X
        else:
            raise AttributeError("'how' must be 'column' or 'row'")
        return X_new
    def Init_wgt(self): #初始化输入层->隐藏层的权重 w1 和偏置 b1、隐藏层->
输出层的权重 w2 和偏置 b2
        #[-1,1]上均匀分布的随机数并整合成正确大小
        w1=np.random.uniform(-1.0,1.0,size=self.n_hidden*(self.n_inp
ut+1))
        w1=w1.reshape(self.n_hidden,self.n_input+1)
        w2=np.random.uniform(-1.0,1.0,size=self.n_output*(self.n_hid
den+1))
        w2=w2.reshape(self.n_output,self.n_hidden+1)
        return w1,w2

    def Sigmoid(self,z):    #激活函数
        return expit(z)     #expit 等价于 1/(1+np.exp(-z))
    def Sigmoid_gradient(self,z):       #激活函数的梯度
        sg=self.Sigmoid(z)
        return sg*(1-sg)                #仅限 expit(z)
    def L1_Norm(self,L1,w1,w2):         #L1 正则化
        return
(L1/2.0)*(np.abs(w1[:,1:]).sum()+np.abs(w2[:,1:]).sum())
    def L2_Norm(self,L2,w1,w2):         #L2 正则化
        return (L2/2.0)*(np.sum(w1[:,1:]**2)+np.sum(w2[:,1:]**2))

    def Forward_prop(self,X,w1,w2):     #正向传播
        a1=self.Add_bias(X,how='column')
        z2=w1.dot(a1.T)                 #矩阵相乘(w1×a1 的转置)
        a2=self.Sigmoid(z2)             #隐含层的 z
        a2=self.Add_bias(a2,how='row')  #将 z2 激活
        z3=w2.dot(a2)
        a3=self.Sigmoid(z3)
        return a1,z2,a2,z3,a3           #数据集全部返回
    def Get_cost(self,labels,a3,w1,w2): #损失函数(交叉熵)
        #labels 为 10*60000 训练集标签数组,a3 为正向传播的输出
        c1=labels*(np.log(a3))
        c2=(1-labels)*np.log(1-a3)
        cost=np.sum(-c1-c2)             #交叉熵
        L1_norm=self.L1_Norm(self.L1,w1,w2)
        L2_norm=self.L2_Norm(self.L2,w1,w2)
        cost=cost+L1_norm+L2_norm
        return cost

    def Get_gradient(self,a1,a2,a3,z2,labels,w1,w2):    #反向传播,计算
w2 w2 的梯度(下降步长)
        sigma3=a3-labels
        z2=self.Add_bias(z2,how='row')
```

```
        sigma2=w2.T.dot(sigma3)*self.Sigmoid_gradient(z2)
        sigma2=sigma2[1:,:]
        grad1=sigma2.dot(a1)            #w1 调整
        grad2=sigma3.dot(a2.T)          #w2 调整
        grad1[:,1:]+=(w1[:,1:]*(self.L1+self.L2))
        grad2[:,1:]+=(w2[:,1:]*(self.L1+self.L2))
        return grad1,grad2

    def Test(self,X):       #训练结束后,输入图像集 X 得到每张图像的识别结
果
        a1,z2,a2,z3,a3=self.Forward_prop(X,self.w1,self.w2)
        #返回每一列最大元素的行号(由于 z3 中每列只有 1 个 1 其他 9 个 0,因此提
取出的就是每个训练图像的真实值)
        ans=np.argmax(z3,axis=0)
        return ans

    def Train(self,X,y,print_progress=False):
        self.costs=[]       #存储每次训练后的损失函数
        X_data,y_data=X.copy(),y.copy()
        labels=self.Encode_label(y,self.n_output)
        d_w1_prev=np.zeros(self.w1.shape)   #前一次 w 下降步长
        d_w2_prev=np.zeros(self.w2.shape)

        for i in range(self.epoch_times):
            self.Eta/=(1+self.de_Eta*i) #随迭代次数增加而降低 Eta
            # print('\rEpoch times:%d/%d'%(i+1,self.epoch_times))
            if print_progress:          #错误标准输出(后面的可以刷掉前面
的)
                # sys.stderr.write('\rEpoch
times: %d/%d'%(i+1,self.epoch_times))
                sys.stderr.write("\rEpoch times: %d/%d
"%(i+1,self.epoch_times)+"■ "*(i//20))
                sys.stderr.flush()
            if self.shuffle:            #打乱训练集的顺序以防陷入死循环
                #np.random.permutation(n)对[0,3)范围内的数字随机排序
                idx=np.random.permutation(y_data.shape[0])  #y_data的
行之间随机打乱
                X_data,y_data=X_data[idx],y_data[idx]
            #将 y_data 所有行按顺序等分成 n_minibatch 组
            mini=np.array_split(range(y_data.shape[0]),self.n_miniba
tch)
            for sub in mini:            #每个小批量进行正向传播并优化参数
                #正向传播并计算损失函数
                a1,z2,a2,z3,a3=self.Forward_prop(X[sub],self.w1,self
.w2)
                cst=self.Get_cost(labels[:,sub],a3,self.w1,self.w2)
                self.costs.append(cst)
                #反向传播计算梯度
```

```
                grad1,grad2=self.Get_gradient(a1,a2,a3,z2,labels[:,s
ub],self.w1,self.w2)
                #更新权重
                d_w1=self.Eta*grad1     #当前 w 下降步长
                d_w2=self.Eta*grad2
                self.w1-=(d_w1+(self.alpha*d_w1_prev))
                self.w2-=(d_w2+(self.alpha*d_w2_prev))
                d_w1_prev,d_w2_prev=d_w1,d_w2
        return self

def costplt1(nn):       #所有代价
    plt.plot(range(len(nn.costs)),nn.costs)
    plt.ylim([0,2000])
    plt.ylabel('Cost')
    plt.xlabel('Epoch_times*50')
    plt.tight_layout()
    plt.show()
def costplt2(nn):       #每次(训练代价的均值)
    #np.array_split(array,n)将 array 分成 n 组
    idx=np.array_split(range(len(nn.costs)),1000)   #costs 分为 1000 组
(训练次数)的坐标数组
    costs=np.array(nn.costs)
    costs_avg=[np.mean(costs[i]) for i in idx]      #每组 costs 的平均
值
    plt.plot(range(len(costs_avg)),costs_avg,color='red')
    plt.ylim([0,10000])
    plt.ylabel('Cost/train')
    plt.xlabel('Epoch_times')
    plt.tight_layout()
    plt.show()

if __name__ == '__main__':
    #读取训练样本和测试样本
    path = 'OptimizationTechnology\\实验 4\\MNIST'
    X_train,y_train=load_mnist_train(path,kind='train')
#X_train:60000*784
    X_test,y_test=load_mnist_train(path,kind='t10k')    #X_test:1000
0*784
    #训练样本并绘制损失函数
    nn=Neural_Nets(X_train.shape[1],10,50,0.0,0.1,1000,0.001,0.001,0
.00001,True,50,1)
    nn.Train(X_train,y_train,True)      #训练结束后 w 和 b 被优化至最佳
    costplt1(nn)
    costplt2(nn)
    #训练集和测试集的结果及准确度
    train_ans=nn.Test(X_train)          #训练集每个图像识别出来的结果
    acc=np.sum(y_train==train_ans,axis=0)/X_train.shape[0]
    print('\n 训练准确率: %.2f%%'%(acc*100))
```

```
    test_ans=nn.Test(X_test)            #测试集每个图像识别出来的结果
    acc=np.sum(y_test==test_ans,axis=0)/X_test.shape[0]
    print('测试准确率: %.2f%%'%(acc*100))
    #前 30 个错误样本
    wrong_img=X_test[y_test!=test_ans][:30]     #截取前 30 个
    real_lab=y_test[y_test!=test_ans][:30]
    wrong_lab=test_ans[y_test!=test_ans][:30]
    fig,ax=plt.subplots(nrows=5,ncols=6,sharex=True,sharey=True)
    ax=ax.flatten()
    for i in range(30):
        img=wrong_img[i].reshape(28,28)
        ax[i].imshow(img,cmap='Greys',interpolation='nearest')
        ax[i].set_title('train:%d'%(wrong_lab[i]))
    ax[0].set_xticks([])
    ax[0].set_yticks([])
    plt.tight_layout()
    plt.show()
    #前 30 个正确样本
    right_img=X_test[y_test==test_ans][:30]
    real_lab=y_test[y_test==test_ans][:30]
    right_lab=test_ans[y_test==test_ans][:30]
    fig,ax=plt.subplots(nrows=5,ncols=6,sharex=True,sharey=True)
    ax=ax.flatten()
    for i in range(30):
        img=right_img[i].reshape(28,28)
        ax[i].imshow(img,cmap='Greys',interpolation='nearest')
        ax[i].set_title('train:%d'%(right_lab[i]))
    ax[0].set_xticks([])
    ax[0].set_yticks([])
    plt.tight_layout()
    plt.show()
```

四、实验结果及分析和（或）源程序调试过程

1. 实验过程

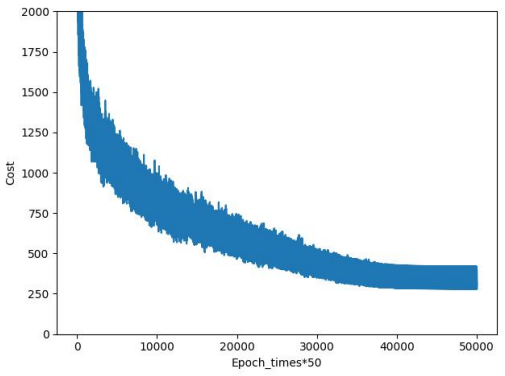
由于**训练次数较多，样本数量庞大**，因此算法**耗时较大**。某些参数下的一套完整的训练甚至需要持续 **7~8** 个小时（见四.3 中参数分析），因此在训练过程加入如下**进度条**，可以较好的监控程序的运行状态。

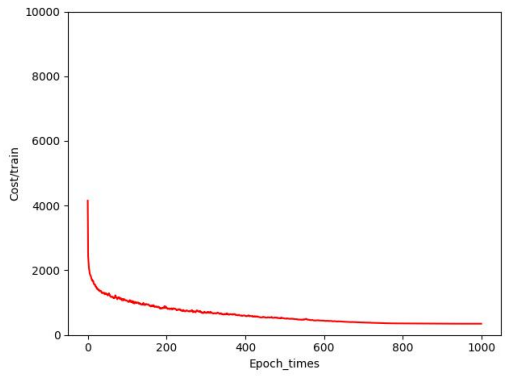
```
Epoch times: 3/1000
Epoch times: 10/1000
Epoch times: 57/1000
Epoch times: 104/1000
Epoch times: 379/1000
Epoch times: 1000/1000
```

2. 实验结果与分析

（1）根据实验设定参数，上述程序的准确率、损失函数、每次训练的平均损失函数如下：

```
训练准确率：97.81%
测试准确率：96.32%
```

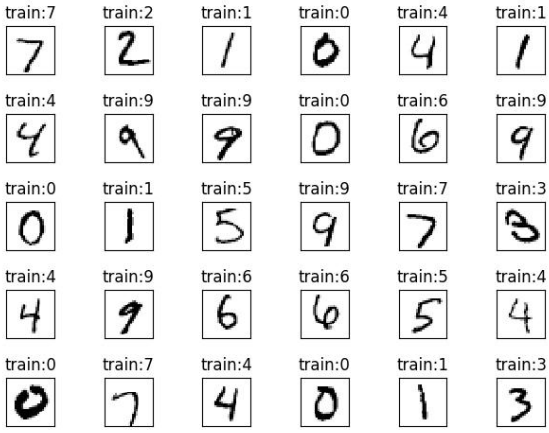




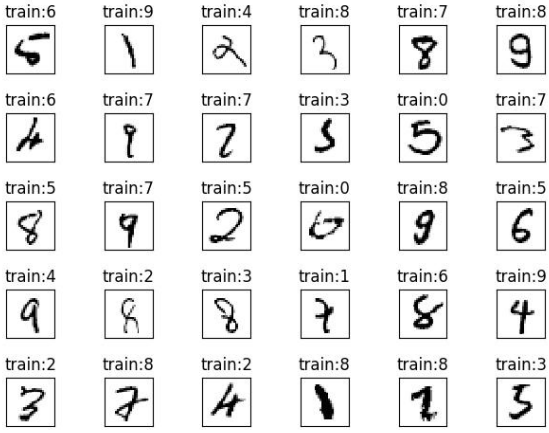
实验的**训练集准确率 97.81%，测试集准确率 96.32%，损失函数**在训练中**上下波动**，总体上**不断减小并趋于 0**。

（2）由于样本数量庞大，正确识别与错误判定的**样本量也相当大**，因此各自只**选取 30 个样例**进行展示如下。每个图像上方标题 train 的值为训练模型最终识别出的结果。

正确样本：



错误样本：



可以看到，错误识别的样本多为书写不规范的样例，将**错误率**控制在 **5%以内**认为是**可以接受**的。

3. 参数分析

神经网络中许多**参数的设定**均来源于**人为经验或约定俗成**，往往直接决定了实验结果的**准确程度**。若参数设定不当，会使得程序无法得到最优解，陷入**局部最优**甚至**死循环**。本程序中影响实验结果的人为参数主要有**隐藏层特征数**、**L1 正则化系数**、**L2 正则化系数**、**迭代次数**、**学习率**、**学习率下降常数**、**小批量数**以及**一次训练完成后是否打乱**。本程序中设定的参数为：

参数	参数值
隐藏层特征数	50
L1 正则化系数	0.0
L2 正则化系数	0.1
迭代次数	1000
学习率	0.001
学习率下降常数	0.00001
小批量数	50
是否打乱训练集	True

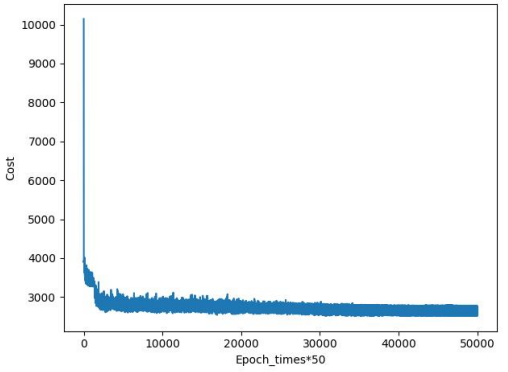
接下来采用**控制变量，固定其他参数**，分别探究各个**参数变量对结果的影响**：以

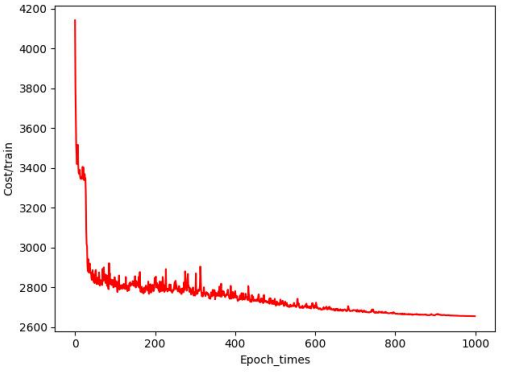
图片呈现为主，文字描述为辅。分别为该参数下的**训练与测试准确率、损失函数、每次训练的平均损失函数。**（**正确样本、错误样本**放在附录中，按需查看）

（1）隐藏层特征数：

n_hidden=2:

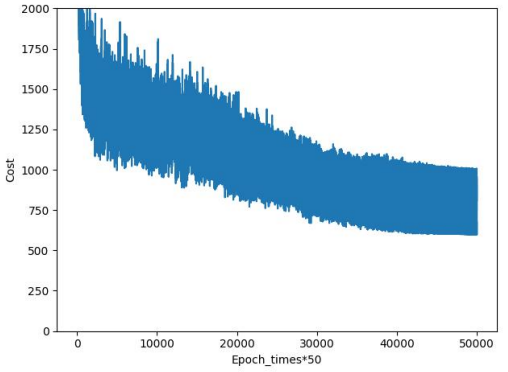
```
训练准确率： 38.75%
测试准确率： 38.20%
```

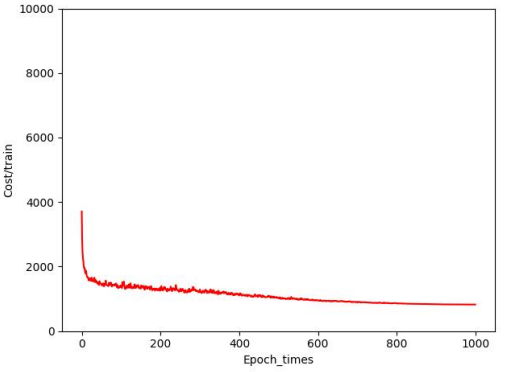




n_hidden=10:

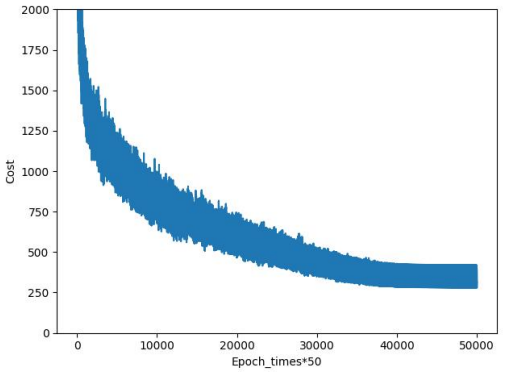
```
训练准确率： 91.72%
测试准确率： 90.65%
```

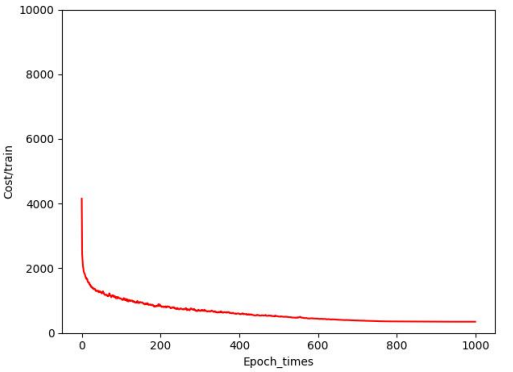




n_hidden=50:

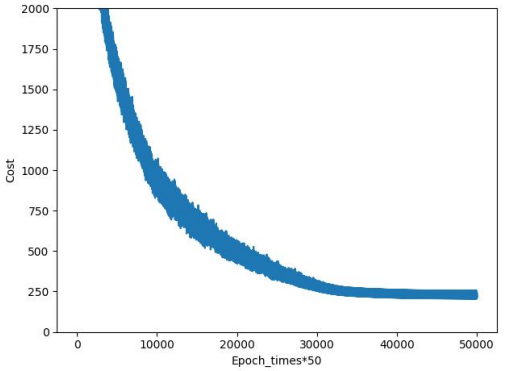
```
训练准确率： 97.81%
测试准确率： 96.32%
```

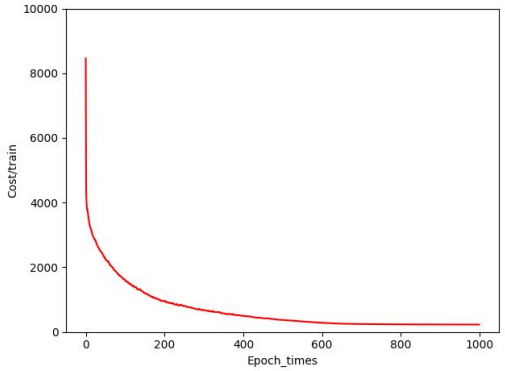




n_hidden=200:

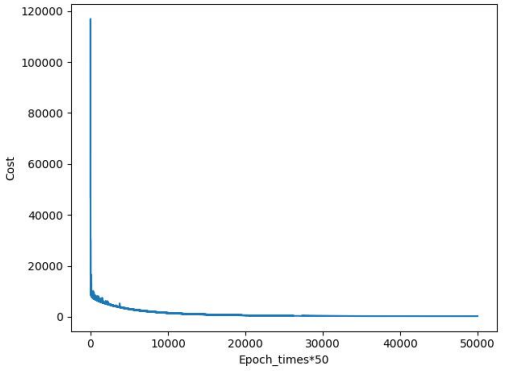
```
训练准确率：99.63%
测试准确率：97.51%
```

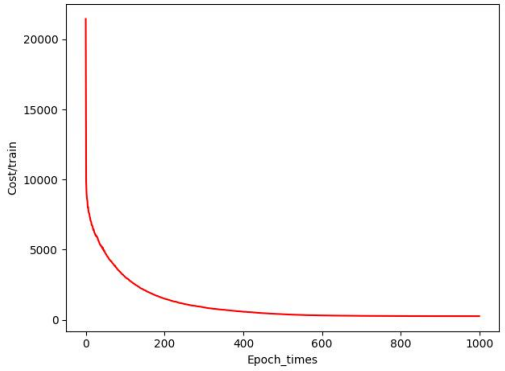




n_hidden=500：

```
训练准确率：99.91%
测试准确率：97.73%
```



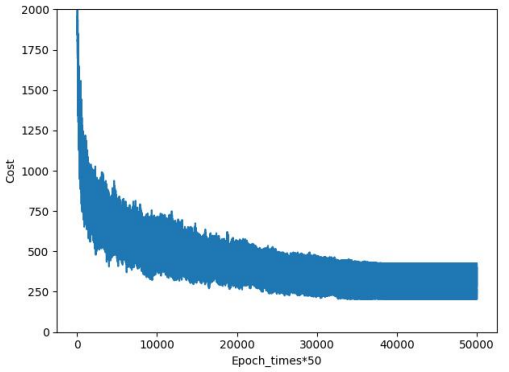


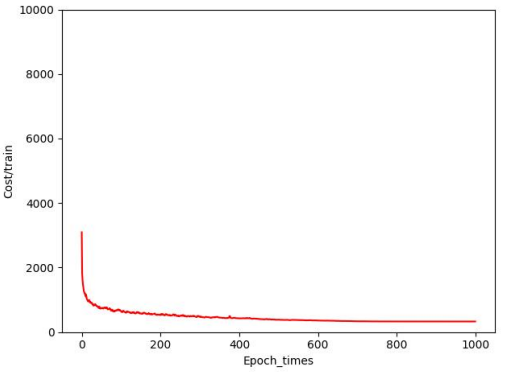
由实验结果可知，**隐藏层特征数越多**，对输入像素的**特征提取越充分，训练模型的准确度越高**。同时，上述五组参数中各自完成一次训练（一整个模型需迭代 1000 次训练）的耗时大致为 `0.2s`、`0.3s`、`1s`、`2s`、`4s`。因此，虽然隐藏层特征数的增加可以提高准确度，但会在**时间上带来巨大消耗**。当 `n_hidden=50` 时训练模型的准确度已经相当可观。

（2）正则化系数：

L2=0：

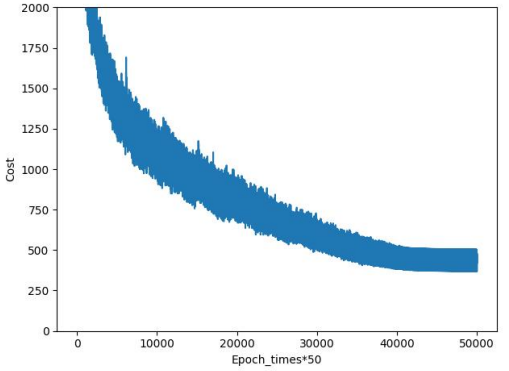
```
训练准确率：96.07%
测试准确率：94.55%
```

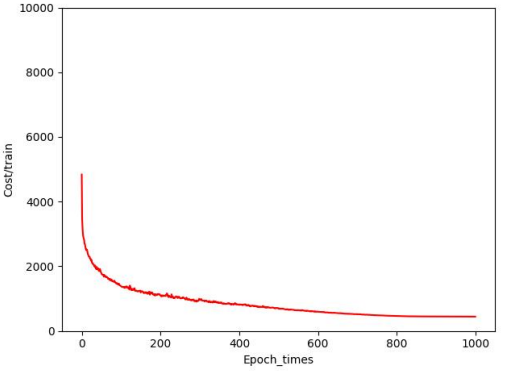




L1=0.1：

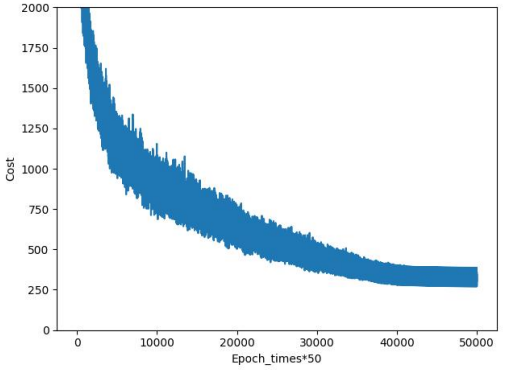
训练准确率：98.12%
测试准确率：96.39%

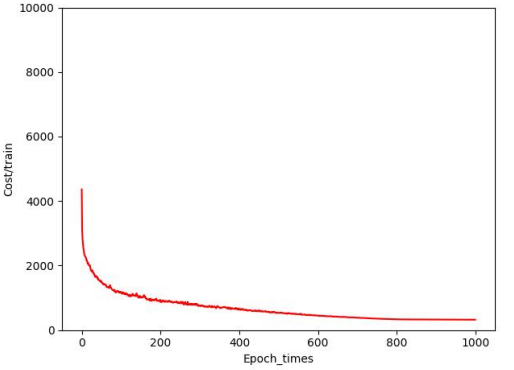




L2=0.2：

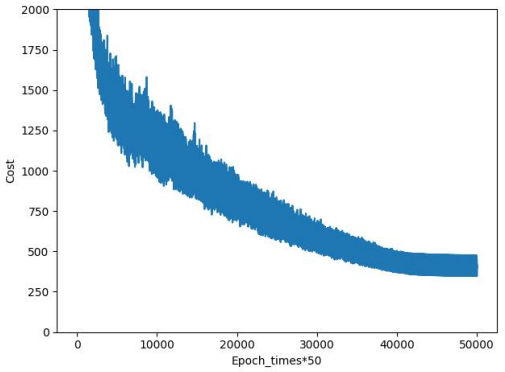
训练准确率：98.19%
测试准确率：96.44%

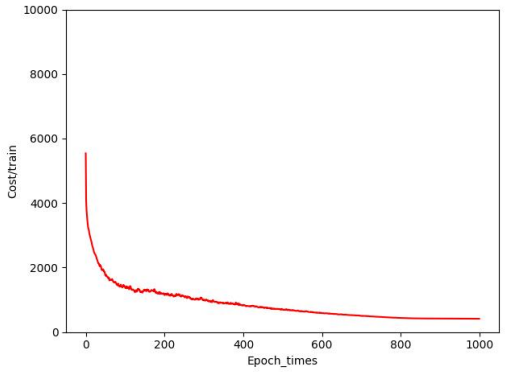




L1=0.1，L2=0.2：

训练准确率：98.31%
测试准确率：96.04%

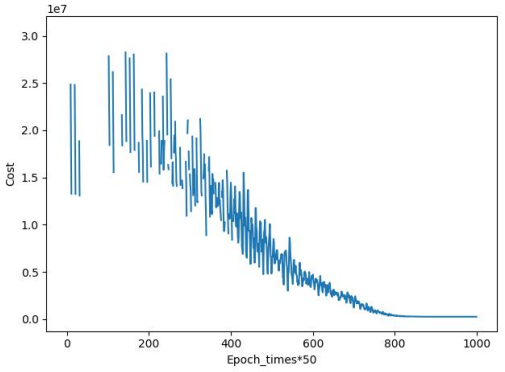


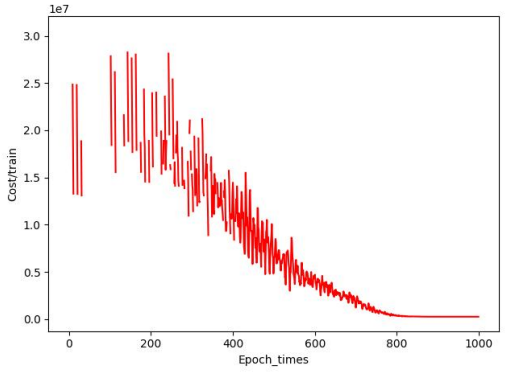


由实验结果知，除了样例 1 中 L2=0 的情况，其他参数围绕 L1=0.1，L2=0.1 设定差别不大。由于 L1、L2 **正则化系数**的作用为**添加正则化项，降低过拟合程度**。若将 L1、L2 都设为 0，则可能出现**过拟合情况，降低准确率**。

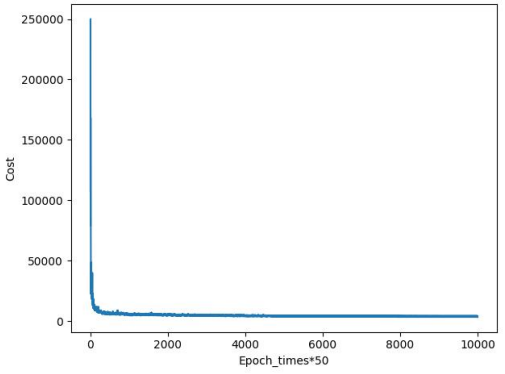
（3）小批量数：

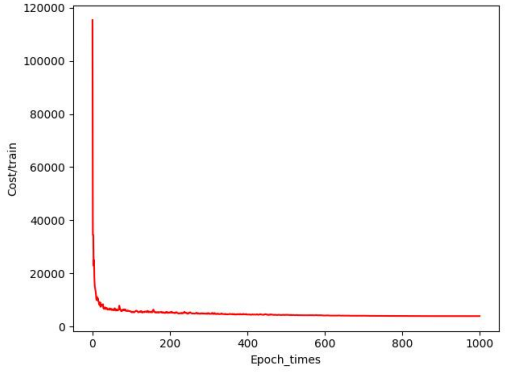
minibatch=1：





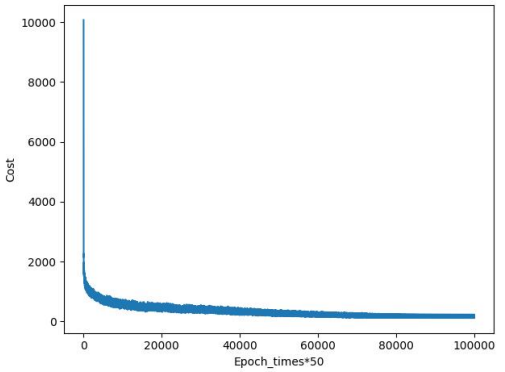
minibatch=10：

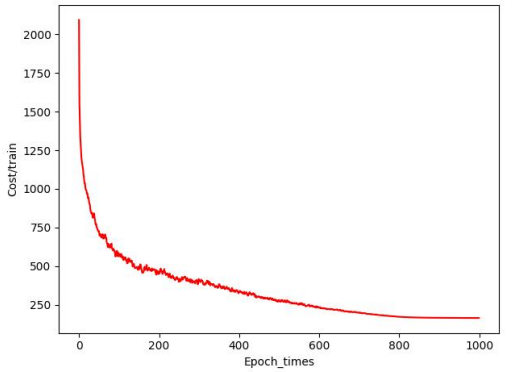




minibatch=100：

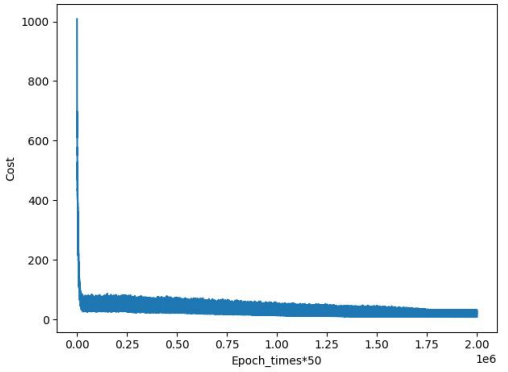
训练准确率：98.07%
测试准确率：95.95%

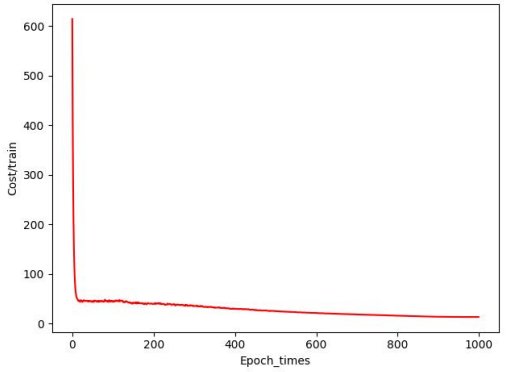




minibatch=2000：

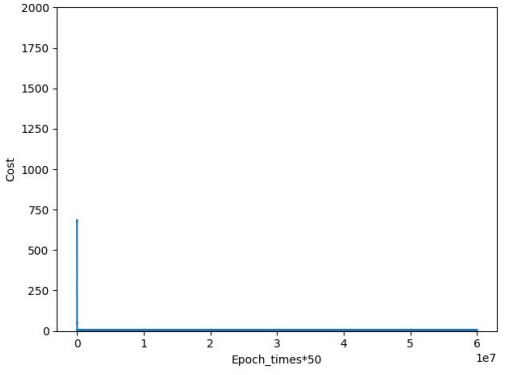
训练准确率：97.86%
测试准确率：95.37%

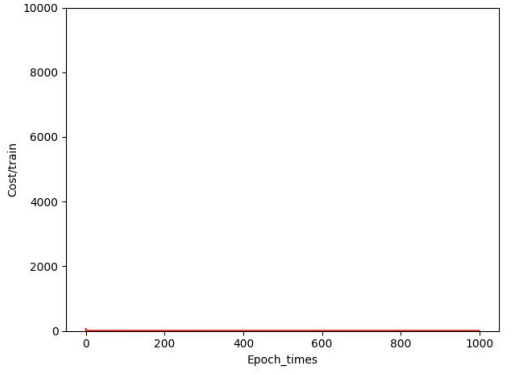




minibatch=n：

训练准确率：99.93%
测试准确率：98.89%



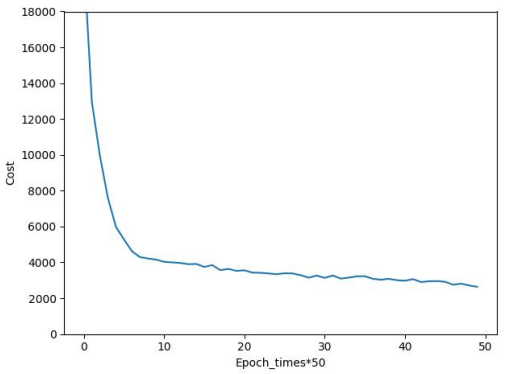


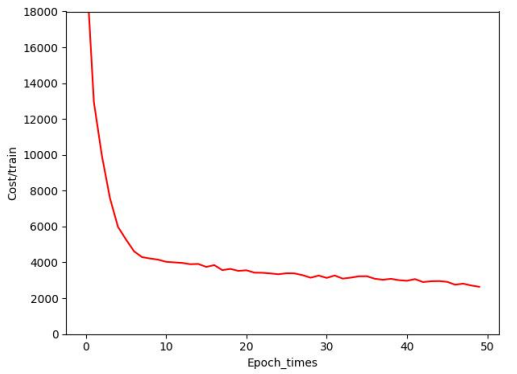
minibatch 为一次训练选取的样本数，**minibatch 越大，耗时越长**，也**越精确**。由实验可知，**minibatch 达到 50** 时，训练结果已经相当**精确**，此时再增大 minibatch 收效甚微。其中 minibatch=1 即为**随机梯度下降**，minibatch=n 即为**批量梯度下降**。批量梯度下降的样例运行时间为 8h 左右（凌晨 1 点~早上 9 点）。

（4）训练次数

epoch_times=1：

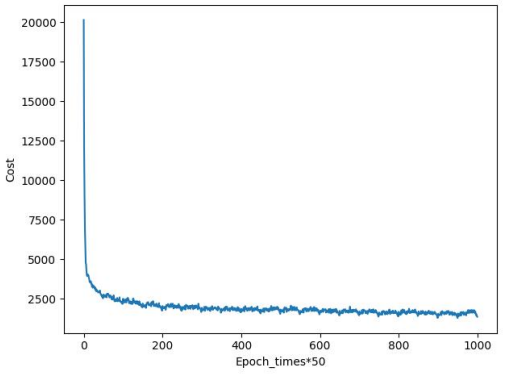
```
训练准确率：64.94%
测试准确率：65.19%
训练准确率：69.90%
测试准确率：69.86%
```

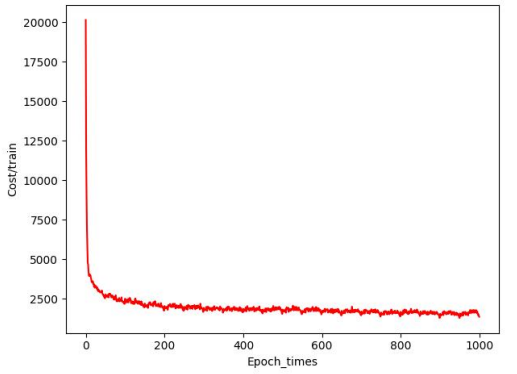




epoch_times=20：

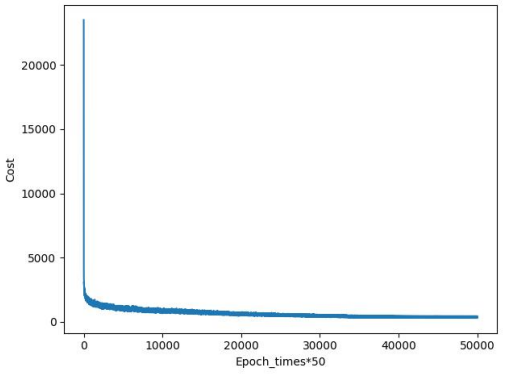
```
训练准确率：87.48%
测试准确率：88.52%
```

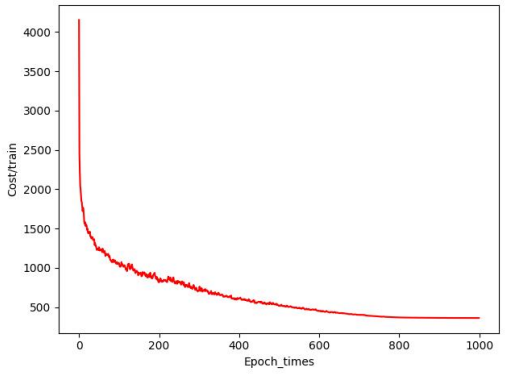




epoch_times=100：

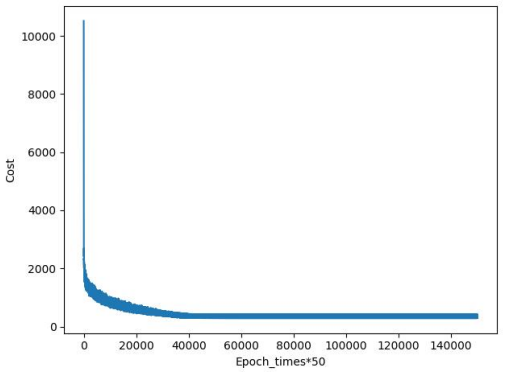
```
训练准确率：97.60%
测试准确率：95.73%
```

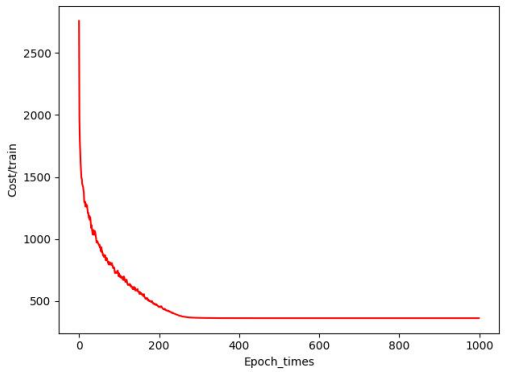




epoch_times=3000：

```
训练准确率：97.60%
测试准确率：96.02%
```

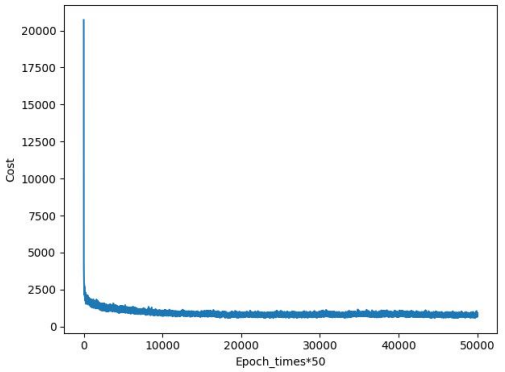


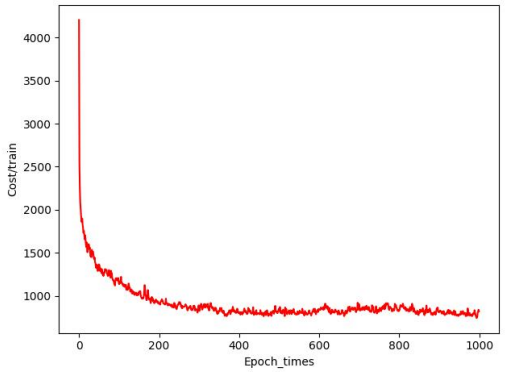


由尝试可知，**训练次数越多，结果越精确**。当**迭代次数达到 100 时结果符合预期**。其中若 epoch_time 较小，不仅结果不精确，得到的结果也不稳定。

（5）Eta 不下降（de_Eta=0）

```
训练准确率：93.36%
测试准确率：92.99%
```

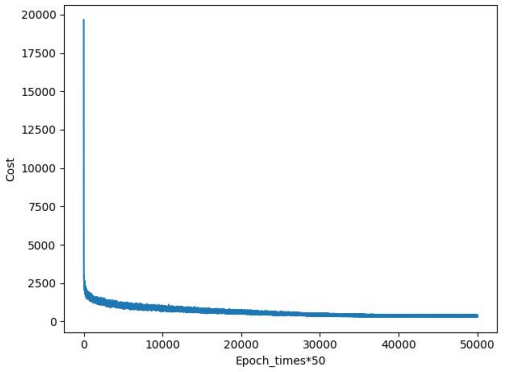


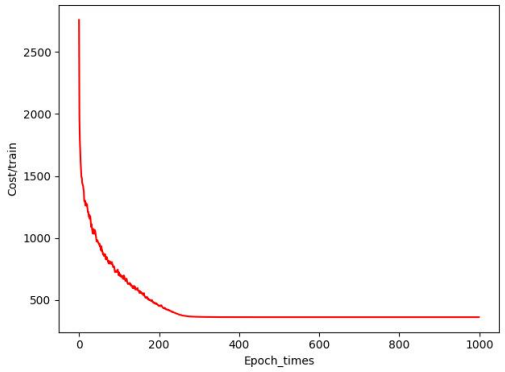


随着迭代次数的增加，梯度下降越来越缓慢。若保持学习率 Eta 不变，可能会使得一小部分解越过最优解。如上所述，将 de_Eta 设为 0 后，准确度降低。

（6）Shffle=False

```
训练准确率：97.85%
测试准确率：96.08%
```

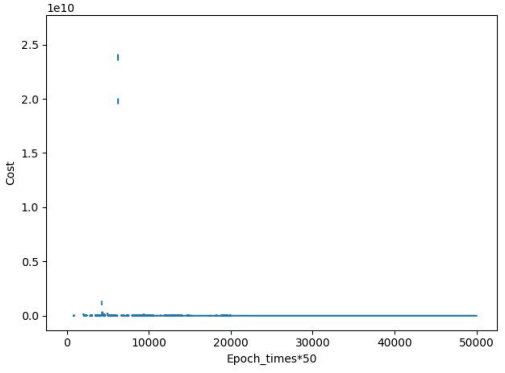


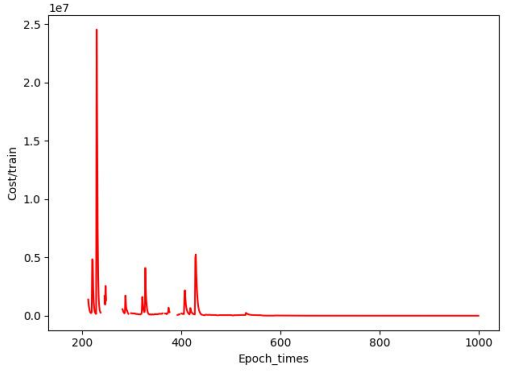


每次训练完成后将样本随机打乱，以防止程序陷入死循环。本实验中是否打乱对结果并无影响。

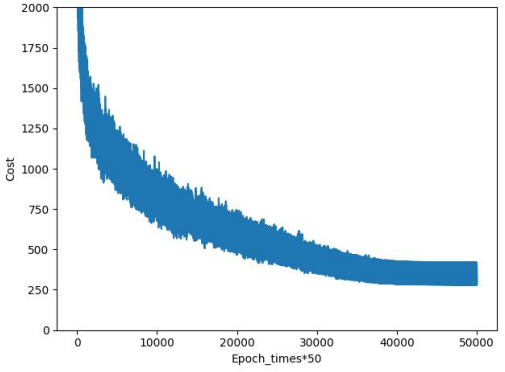
（7）学习率

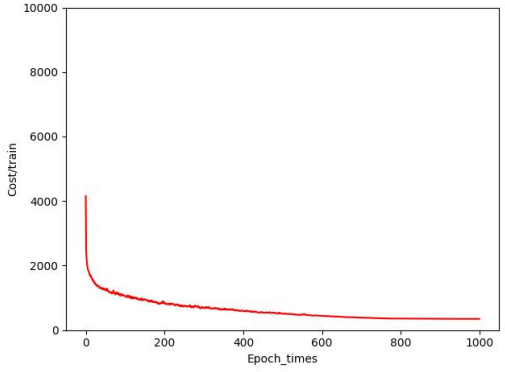
Eta=0.1





Eta=0.001





学习率设置过大使得梯度下降过程**越过最优解**，降低准确度。

4. 调试过程

（1）用实验报告模板中 `load_mnist` 函数读取文件时找不到相应压缩包：

```
Traceback (most recent call last):
  File "d:\PyLearn\2.py", line 265, in <module>
    X_train, y_train = load_mnist(path, kind='train') # X_train : 60000*784
  File "d:\PyLearn\2.py", line 37, in load_mnist
    with gzip.open(labels_path,'rb') as lbpath:
  File "C:\Users\Jackson1125\AppData\Local\Programs\Python\Python38\lib\gzip.py", line 58, in open
    binary_file = GzipFile(filename, gz_mode, compresslevel)
  File "C:\Users\Jackson1125\AppData\Local\Programs\Python\Python38\lib\gzip.py", line 173, in __init__
    fileobj = self.myfileobj = builtins.open(filename, mode or 'rb')
FileNotFoundError: [Errno 2] No such file or directory: 'OptimizationTechnology\\实验4\\MNIST\\train-labels-idx1-ubyte'
```

解决办法：因为本机的文件名显示后缀，因此需要**加上后缀名“.gz”**

（2）使用实验报告模板中给定函数解码时 np.fromstring 解二进制图像出现问题：

```
d:\PyLearn\OptimizationTechnology\实验4\Neural_nets.py:25: DeprecationWarning: The binary mode of fromstring is
 deprecated, as it behaves surprisingly on unicode inputs. Use frombuffer instead
  labels=np.fromstring(lbpath.read(),dtype=np.uint8)
d:\PyLearn\OptimizationTechnology\实验4\Neural_nets.py:29: DeprecationWarning: The binary mode of fromstring is
 deprecated, as it behaves surprisingly on unicode inputs. Use frombuffer instead
  images=np.fromstring(imgpath.read(),dtype=np.uint8).reshape(len(labels),784)
```

解决办法：改为 frombuffer 即可

（3）shuffle 随机打乱样本后 image 和 label 不匹配

解决办法：随机生成 index，然后同时改变 image 和 label

```
if self.shuffle:              #打乱训练集的顺序以防陷入死循环
    #np.random.permutation(n)对[0,3)范围内的数字随机排序
    idx=np.random.permutation(y_data.shape[0])  #y_data的行之间随机打乱
    X_data,y_data=X_data[idx],y_data[idx]
```

（4）提取所有错误样本数量未知，无法传入 fig 和 ax 进行绘图

解决办法：截取前 30 个样本展示即可

```
#前30个错误样本
miscl_img=X_test[y_test!=test_ans][:30]      #截取前30个
correct_lab=y_test[y_test!=test_ans][:30]
miscl_lab=test_ans[y_test!=test_ans][:30]
```

（5）实例化对象时报错：

```
  File "d:\PyLearn\OptimizationTechnology\实验4\Neural_nets.py", line 68
    def __init__(self,n_input,n_output,n_hidden,
                 ^
SyntaxError: non-default argument follows default argument
```

解决办法：将没有默认值的参数放在有默认值的参数前面

（6）Test 函数正向传播后没有直接返回结果

解决办法：识别出来的数字即为 1，其他为 0。用提取最大值方法返回坐标：

```
def Test(self,X):       #训练结束后,输入图像集X得到每张图像的识别结果
    a1,z2,a2,z3,a3=self.Forward_prop(X,self.w1,self.w2)
    #返回每一列最大元素的行号(由于z3中每列只有1个1其他9个0,因此提取出的就是每个训练图像的真实值)
    ans=np.argmax(z3,axis=0)
    return ans
```

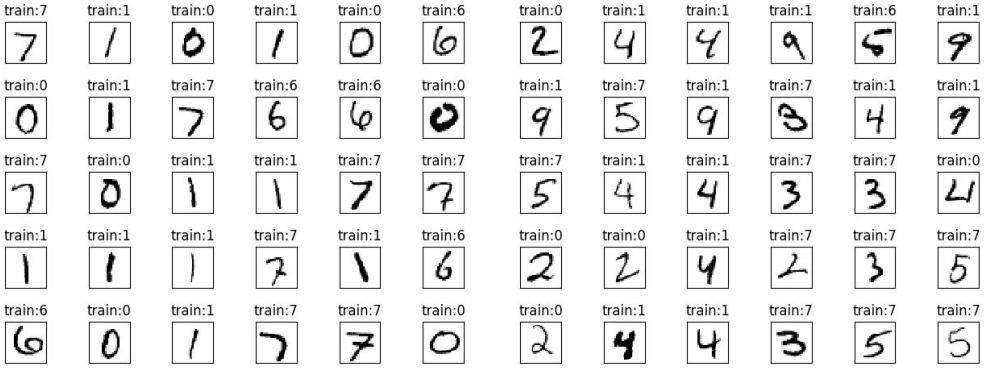
5. 总结

人工神经网络是人工神经元组成的**并行自适应网络**，目标是**对人类神经系统的某个功能进行抽象和建模**。主要就是根据给定的学习策略**正向传播**，**计算损失函数**，然后**反向传播**并**通过梯度下降法优化参数**。

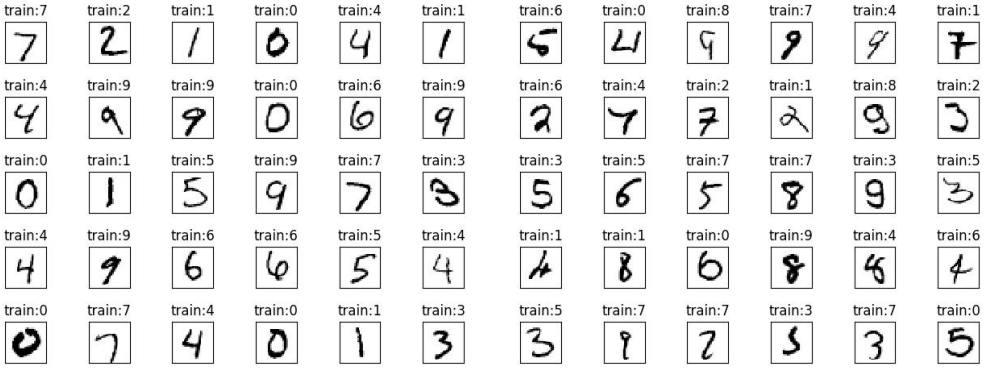
人工神经网络中用到的参数来源于人为经验，在结果的准确度上占有重要作用。

附录 A 参数分析中的正确与错误样本

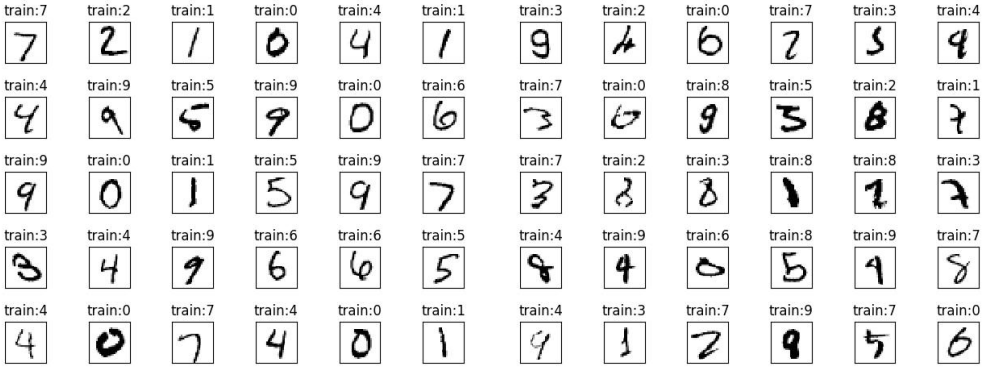
n_hidden=2:



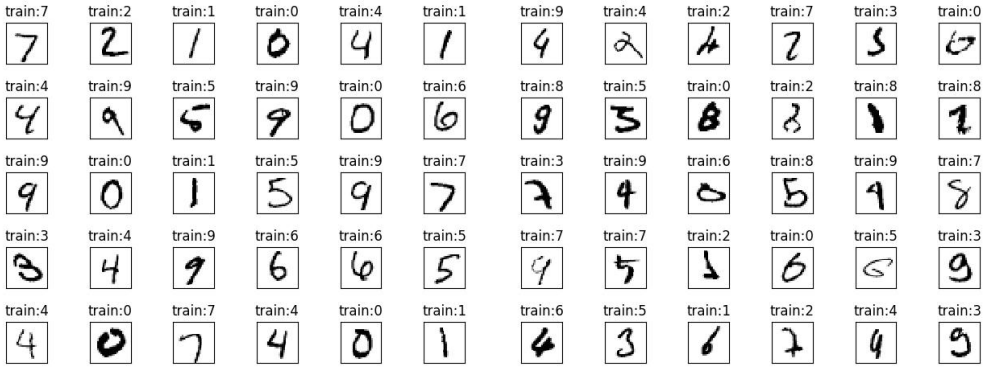
n_hidden=10:



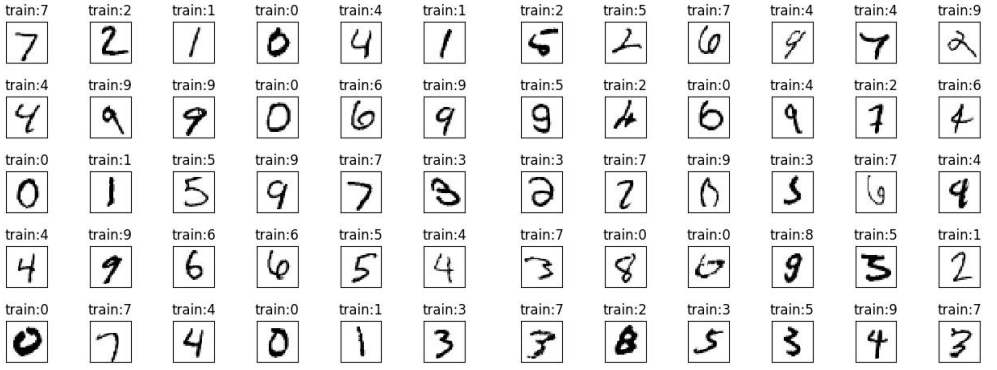
n_hidden=200:



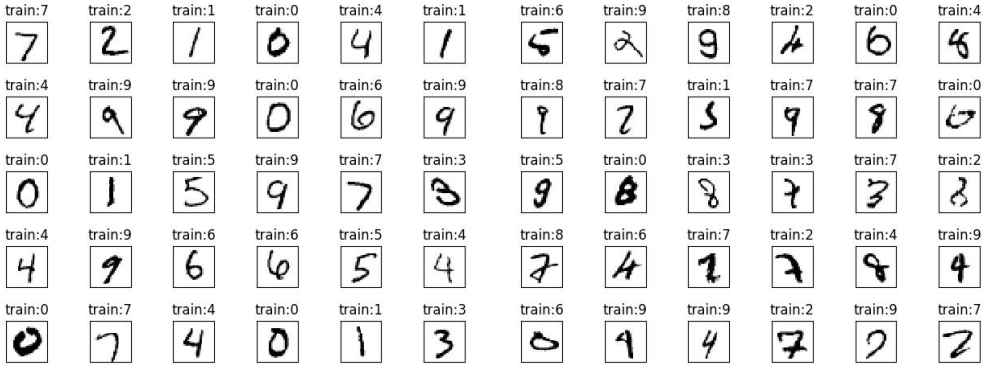
n_hidden=500:



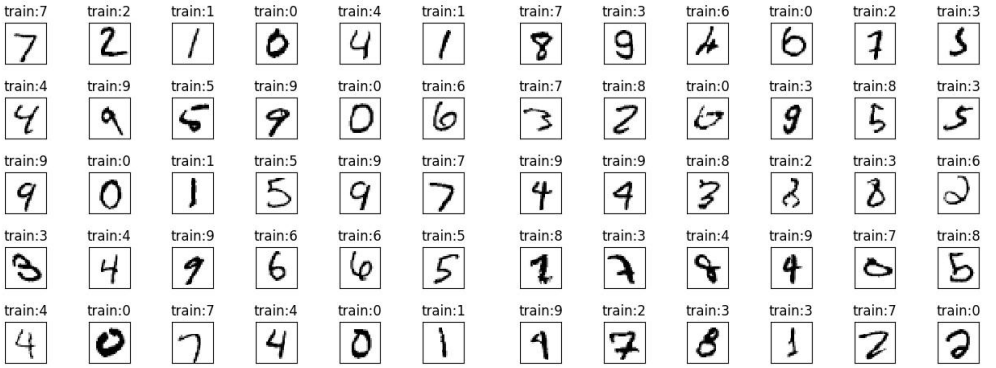
L2=0：



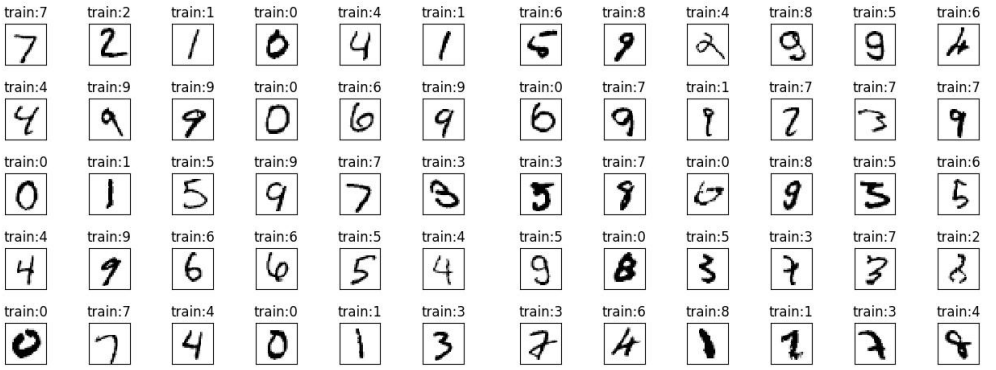
L1=0.1：



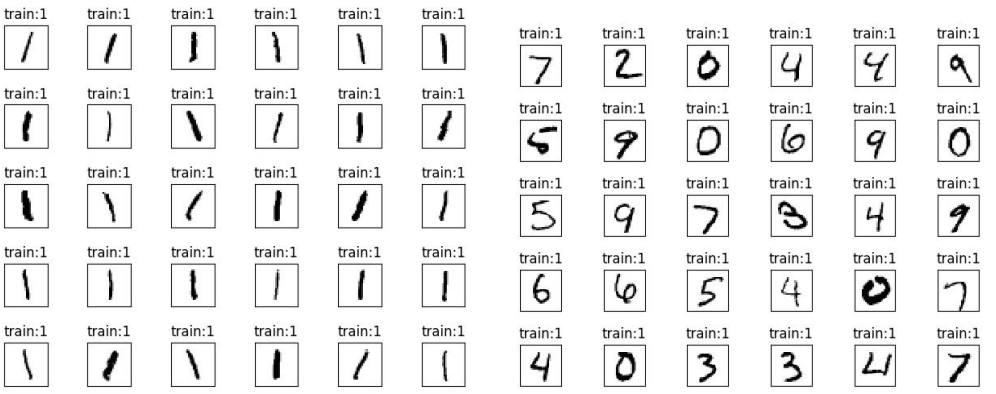
L2=0.2：



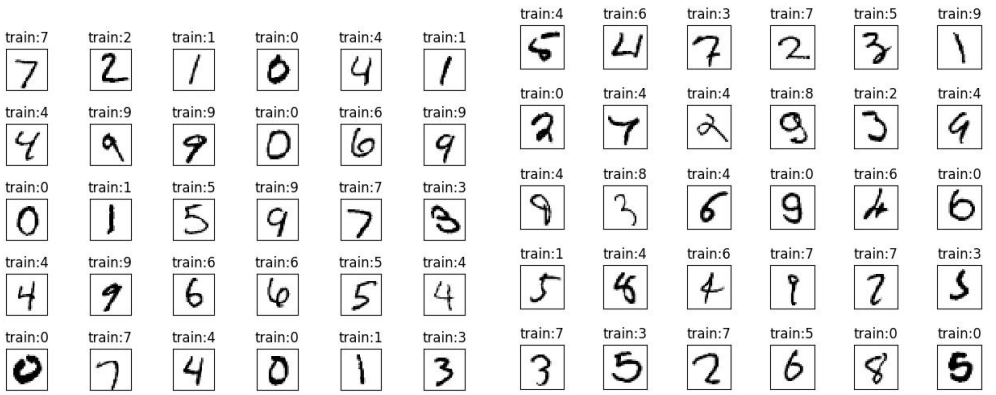
L1=0.1，L2=0.2：



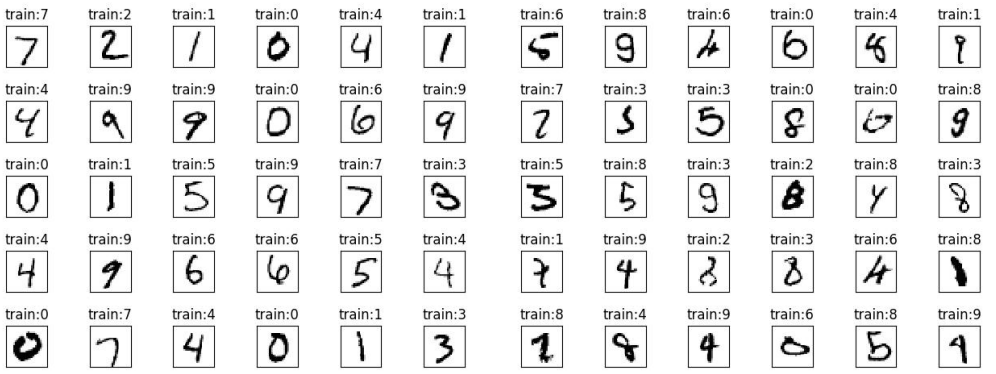
minibatch=1：



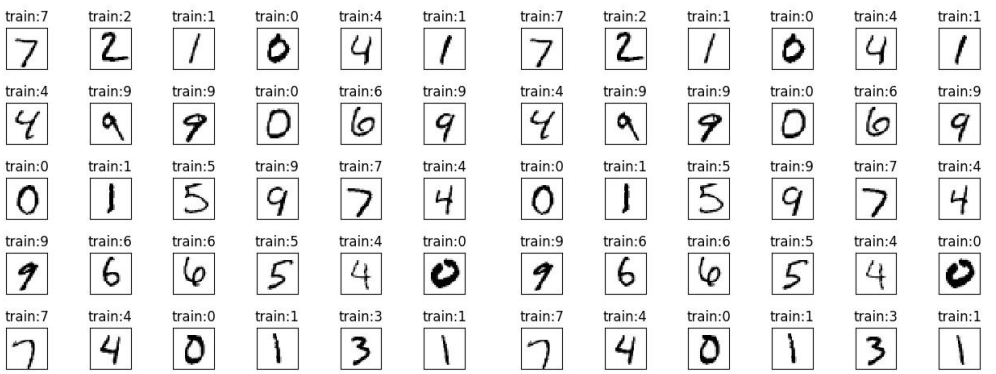
minibatch=10：



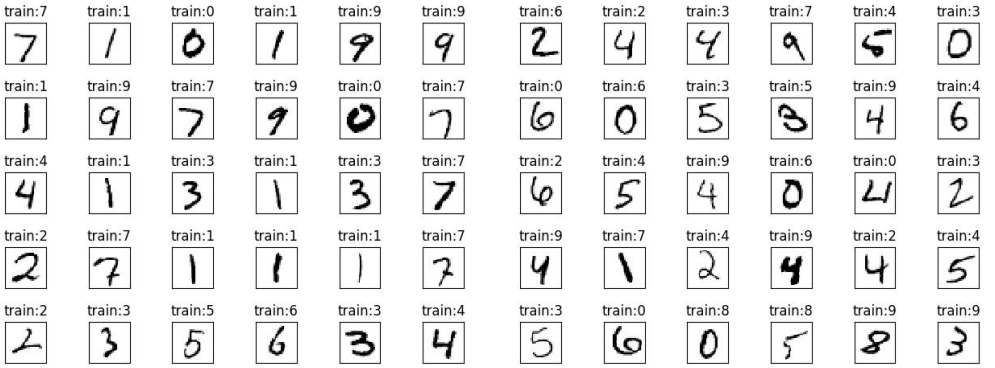
minibatch=100：



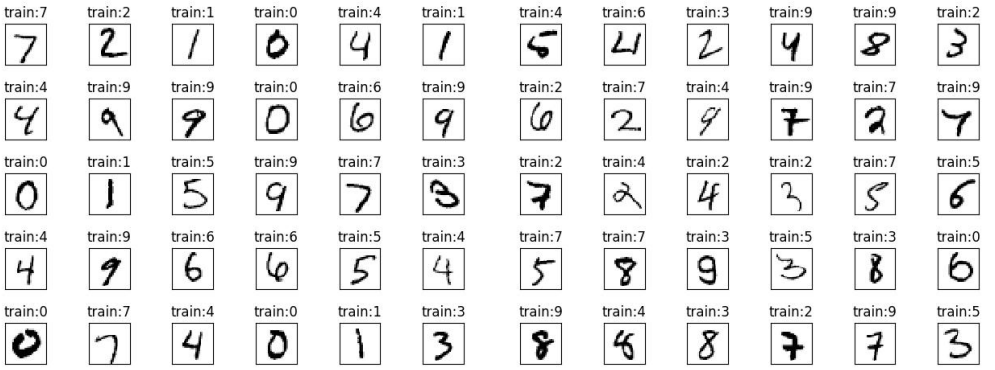
minibatch=n：



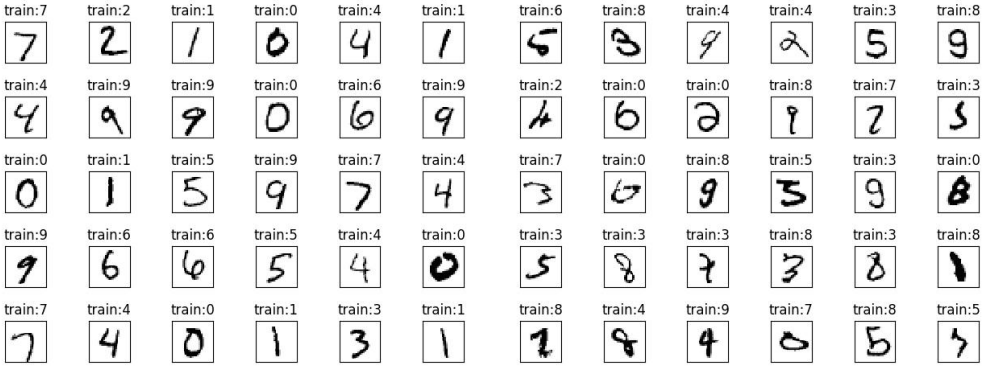
epoch_times=1：



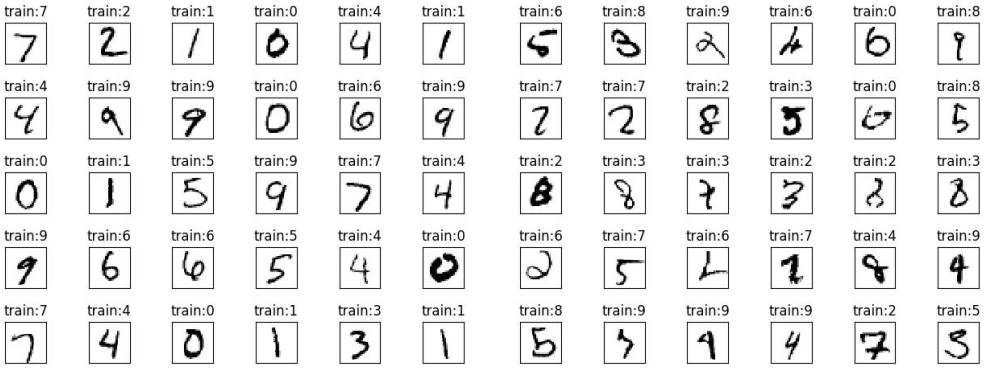
epoch_times=20：



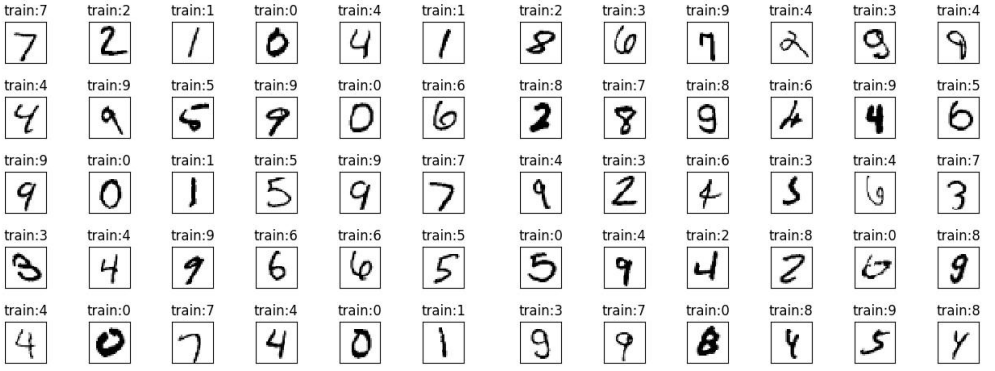
epoch_times=100：



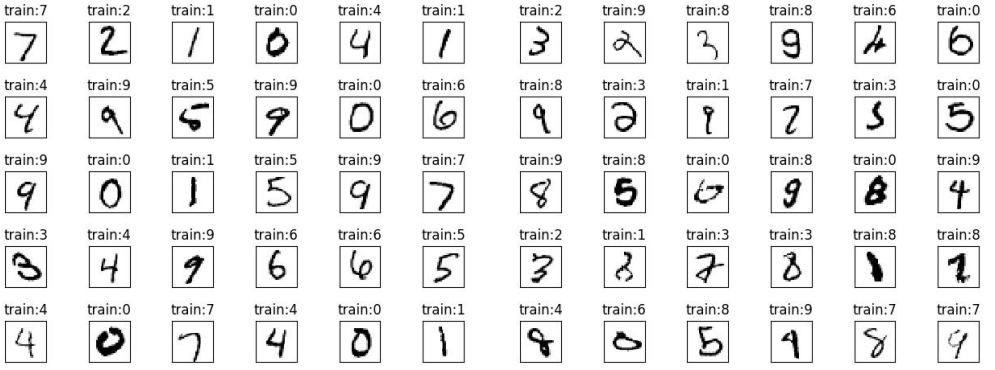
epoch_times=3000：



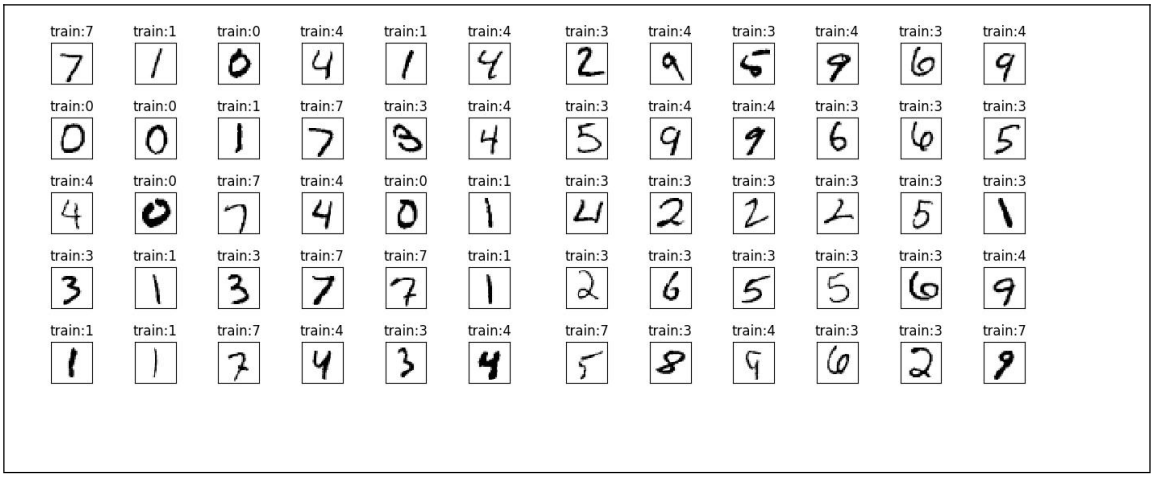
de Eta=0:



shuffle=false：



Eta=0.1



注：电子文档命名要求：学号+姓名+实验序号