实验8 BP网络

实验思路：

BP网络是一种前馈神经网络，分为输入层和隐藏层和输出层；本次实验是2个隐藏层和一个输出层；不同层之间的连接要先经过激活函数，而激活函数一般都是非线性的，选择非线性的原因之一是如果没有非线性激活函数，那么网络就是一个线性网络，非线性可以增加网络的学习能力，另一方面如果选择线性激活函数，那么多层神将网络的学习能力与单层网络的学习能力相同，这一点很容易从数学角度说明，多个线性变化可以合并成一个线性变换；不同神经元通过不同的权重连接，复杂的神经网络要解决的一个问题就是如何训练并更新这些权重，而BP算法就是一种用于更新网络权重的高效算法。

以随机梯度下降法为例，BP算法做的事情就是通过损失函数从输出层反向逐层计算各层关于线性关系系数矩阵W和偏置b的偏导数，进而可以通过梯度下降去更新这些参数。

对于不同问题，损失函数要选择好，对于回归问题通常选取MSE作为损失函数，这个比较好理解，就是预测值和目标值的欧式距离；而在分类问题中则不能使用MSE,因为分类问题本质是学习一个条件概率分布，因此label大小在欧式空间没有意义，另一方面激活函数很多地方都是平滑的，也就是说大部分区域的导数趋于0，使用MSE时，输出层关于前一层的偏导数含激活函数导数作为乘积的一部分，这会导致梯度消失现象出现，进而导致后续参数无法更新，因此分类问题一般使用交叉熵损失，交叉熵的原理可以从最小化KL散度推出，KL散度是一种用于衡量不同概率分布之间相似性的准则函数。

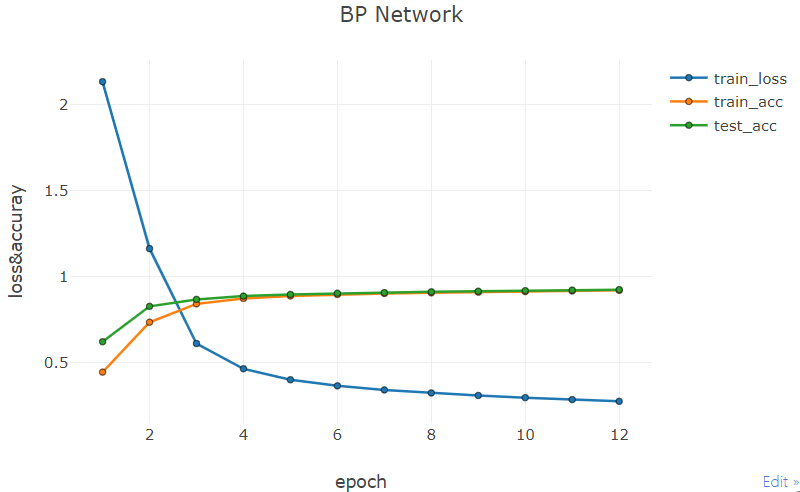
**具体实验过程：**

网络搭建使用pytorch，因为输出应该是一个关于0-9数字出现概率的概率分布，所以应该进行softmax运算。在pytorch中softmax运算在CrossEntryLoss损失函数中进行运算。

训练方面选择了SGD算法，进行12次训练，并用Visdom库绘制出每批次后的训练集准确率、训练集损失以及验证集上的损失**。**

1. 实验结果

训练曲线：



部分训练信息输出（最后准确率在93%）：

Epoch 1

-------------------------------

Train Epoch: 1 [0/60000 (0%)] Loss: 2.302205

Train Epoch: 1 [12800/60000 (21%)] Loss: 2.262058

Train Epoch: 1 [25600/60000 (43%)] Loss: 2.204235

Train Epoch: 1 [38400/60000 (64%)] Loss: 2.131383

Train Epoch: 1 [51200/60000 (85%)] Loss: 1.911251

Test set: Average loss: 1.7635, Accuracy: 6228.0/10000 (62%)

Epoch 2

-------------------------------

Train Epoch: 2 [0/60000 (0%)] Loss: 1.780298

Train Epoch: 2 [12800/60000 (21%)] Loss: 1.466152

Train Epoch: 2 [25600/60000 (43%)] Loss: 1.164305

Train Epoch: 2 [38400/60000 (64%)] Loss: 0.963359

Train Epoch: 2 [51200/60000 (85%)] Loss: 0.734639

Test set: Average loss: 0.7308, Accuracy: 8286.0/10000 (83%)

Epoch 3

-------------------------------

Train Epoch: 3 [0/60000 (0%)] Loss: 0.747539

Train Epoch: 3 [12800/60000 (21%)] Loss: 0.634958

Train Epoch: 3 [25600/60000 (43%)] Loss: 0.571571

show more (open the raw output data in a text editor) ...

Train Epoch: 12 [38400/60000 (64%)] Loss: 0.282717

Train Epoch: 12 [51200/60000 (85%)] Loss: 0.168258

Test set: Average loss: 0.2616, Accuracy: 9257.0/10000 (93%)

1. 源代码(Python)

import torch

from torch import nn

import torchvision

from torchvision import transforms

from torch.utils import data

from visdom import Visdom

viz = Visdom(env='pytorch')

# 加载mnist手写体数据集

def load\_data\_mnist(batch\_size, resize = None):

trans = [torchvision.transforms.ToTensor()]

if resize:

trans.insert(0, transforms.Resize(resize))

trans = transforms.Compose(trans)

mnist\_train = torchvision.datasets.MNIST(root = './data', train = True,

transform = trans, download = False)

mnist\_test = torchvision.datasets.MNIST(root = './data', train = False,

transform =trans, download = False)

return (data.DataLoader(mnist\_train, batch\_size, shuffle = True, num\_workers = 2),

data.DataLoader(mnist\_test, batch\_size, shuffle = False, num\_workers = 2))

# 定义三层BP网络

net = nn.Sequential(nn.Flatten(), nn.Linear(784, 512), nn.ReLU(),

nn.Linear(512, 128),

nn.ReLU(), nn.Linear(128, 10))

# 每一个训练epoch

def train\_epoch(model, train\_loader, optimizer, loss\_fn, epoch):

size = len(train\_loader.dataset)

num\_batches = len(train\_loader)

model.train() # 将模型设置为训练模式

train\_loss, train\_correct = 0, 0

for batch\_idx, (X, y) in enumerate(train\_loader):

pred = model(X)

y = y.to(pred.device)

loss = loss\_fn(pred, y)

# 梯度清零， 反向传播，更新网络参数

optimizer.zero\_grad()

loss.backward()

optimizer.step()

# 记录损失与正确率

train\_loss += loss.item()

train\_correct += (pred.argmax(1) == y).type(torch.float).sum().item()

# 每 100批输出一次

if batch\_idx % 100 == 0:

print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(

epoch, batch\_idx \* len(X), size,

100. \* batch\_idx / num\_batches, loss.item()))

return train\_loss / num\_batches, train\_correct / size

# 测试epoch

def test\_epoch(model, test\_loader, loss\_fn):

size = len(test\_loader.dataset)

num\_batches = len(test\_loader)

model.eval() # 设为评估模式

test\_loss, test\_correct = 0, 0

# 不记录梯度，节省内存

with torch.no\_grad():

for X, y in test\_loader:

pred = model(X)

y = y.to(pred.device)

loss = loss\_fn(pred, y)

test\_loss += loss.item()

test\_correct += (pred.argmax(1) == y).type(torch.float).sum().item()

test\_loss /= num\_batches

print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(

test\_loss, test\_correct, size, 100. \* test\_correct / size))

return test\_loss, test\_correct / size

# 分类问题使用交叉熵作为损失函数

loss\_fn = nn.CrossEntropyLoss()

# 使用随机梯度下降法更新

trainer = torch.optim.SGD(net.parameters(), lr = 0.01)

# 使用DP模式训练

net = nn.DataParallel(net)

# 获取训练数据集和测试数据集

train\_iter, test\_iter = load\_data\_mnist(batch\_size=128)

# 训练轮数

num\_epochs = 12

# 记录损失和正确率

train\_loss, train\_accuracy = [], []

test\_loss, test\_accuracy = [], []

for epoch in range(1, num\_epochs + 1):

print(f"Epoch {epoch}\n-------------------------------")

a, b = train\_epoch(net, train\_iter, trainer, loss\_fn, epoch)

train\_loss.append(a)

train\_accuracy.append(b)

c, d = test\_epoch(net, test\_iter, loss\_fn)

test\_loss.append(c)

test\_accuracy.append(d)

if epoch == 1:

viz.line([[a, b, d]], [epoch], win = 'live loss',

opts=dict(legend = ['train\_loss', 'train\_acc', 'test\_acc'],xlabel='epoch',

ylabel='loss&accuray', title='BP Network',

markers = True, markersize = 5))

else:

viz.line([[a, b, d]], [epoch], win = 'live loss', update = 'append')