人工智能理论与实践

第五章 人工神经网络与深度学习 实验

信息学部 信息与通信工程学院

主要内容

01 实验一 BP神经网络

02 实验二 LeNet网络

主要内容

01 实验一 BP神经网络

02 实验二 LeNet网络

1.1 实验目的

1）理解BP神经网络的基本原理

2）理解数据正向传播和误差的反向传递过程

3）自定义BP神经网络的隐藏层神经元的个数，学习率等参数，观察参数的变化对网络精度的影响

4）使用训练成功的网络，对手写数字图像进行离线测试，测试出网络的识别正确率

1.2 实验环境

1） 操作系统：Windows 10 x64

2）开发环境：PyCharm Community Edition 2022.1 x64 + Python3.6

1.3 实验内容

1）下载并解析MNIST数据集

2）使用Python搭建BP神经网络

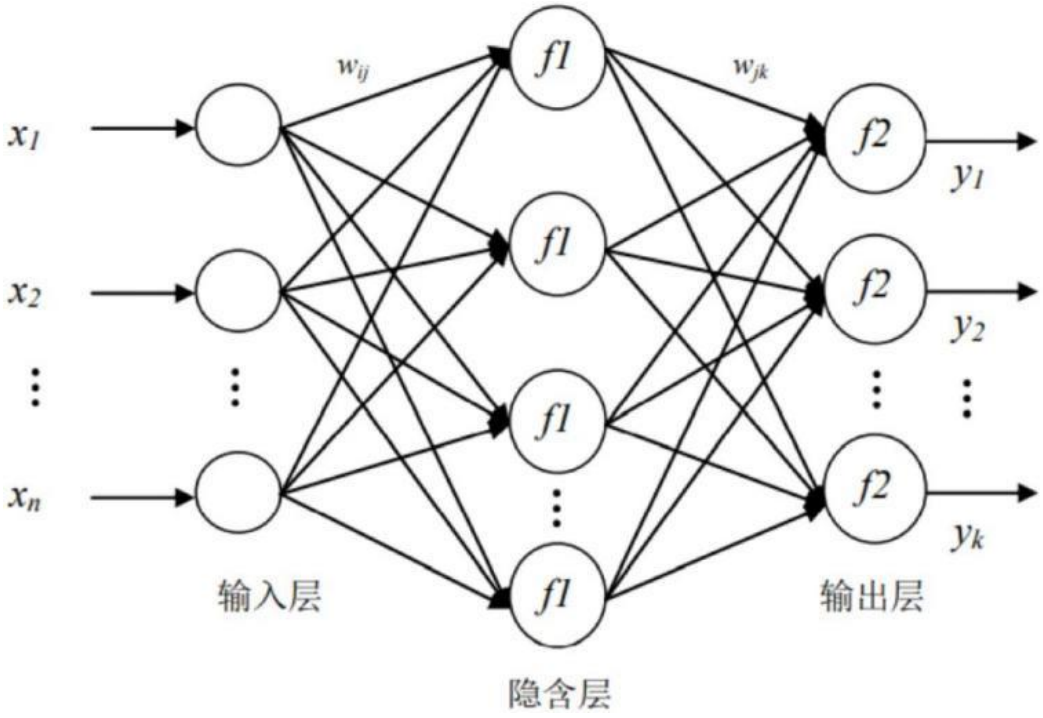
3）使用MNIST数据集完成对BP神经网络的训练和测试，并对实验结果进行分析

1.4 原理说明

1）BP神经网络

基本思想是梯度下降法，利用梯度搜索技术，以期使网络的实际输出值和期望输出值的误差均方差为最小。BP神经网络模型拓扑结构包括输入层（input）、隐藏层(hide layer)和输出层(output layer)。

BP 网络的典型结构如右图所示：



- BP神经网络的学习过程由信号的正向传播和误差的反向传播两个过程组成。
  - 正向传播时，输入样本由输入层传入，经过隐藏层处理后，传入输出层，输出层计算输出预测的结果，正向传播过程到此结束。
  - 当预测的结果与期望存在误差时，则会将误差进行反向传播，通过计算出的误差来反向依次调整隐藏层到输出层的权重和偏置、输入层到隐藏侧的权重和偏置。
  - 如此循环两个过程，直至满足停止条件，如权重的更新低于某个域值，预测的错误率低于某个域值，达到一定的迭代次数等。
- 在本次实验中的神经网络，我们对节点的激励函数采用常用的sigmoid函数。

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

2）MNIST数据集

- MNIST是一个手写体数字的图片数据集，该数据集来由美国国家标准与技术研究所发起整理，一共统计了来自250个不同的人手写数字图片。该数据集主要由一些手写数字的图片和相应的标签组成，图片是28*28的像素矩阵，一共分为10类，图片标签分别对应从0～9，共10个阿拉伯数字。此数据集中，训练样本共60000个，测试样本共10000个。
- MNIST数据集包含了以下四个部分，从上至下分别为：训练集图像（Training set images: train-images-idx3-ubyte.gz）、训练集标签（Training set labels: train-labels-idx1-ubyte.gz）、测试集图像（Test set images: t10k-images-idx3-ubyte.gz）、测试集标签（Test set labels: t10k-labels-idx1-ubyte.gz）。

1.5 实验步骤

第一步，使用Python IDE（本实验中使用的是PyCharm）打开，对数据集进行解析。参考示例代码（decodeMNIST.py）

decodeMinist.py

```python
import numpy as np
import struct
from PIL import Image
# 训练集文件
train_images_idx3_ubyte_file = './datafile/train-images-idx3-ubyte/train-images.idx3-ubyte'
# 训练集标签文件
train_labels_idx1_ubyte_file = './datafile/train-labels-idx1-ubyte/train-labels.idx1-ubyte'

# 测试集文件
test_images_idx3_ubyte_file = './datafile/t10k-images-idx3-ubyte/t10k-images.idx3-ubyte'
# 测试集标签文件
test_labels_idx1_ubyte_file = './datafile/t10k-labels-idx1-ubyte/t10k-labels.idx1-ubyte'


def decode_idx3_ubyte(idx3_ubyte_file):
    """
    解析idx3文件的通用函数
    :param idx3_ubyte_file: idx3文件路径
    :return: 数据集
    """
    # 读取二进制数据
    bin_data = open(idx3_ubyte_file, 'rb').read()  #以二进制格式打开，用于只读，返回整个文件

    # 解析文件头信息，依次为魔数、图片数量、每张图片高、每张图片宽
    offset = 0
    fmt_header = '>iiii'  #四个维度的大小都是四字节整型
    #按照fmt_header指定的格式，从偏移位置offset开始，对bin_data解包
    magic_number, num_images, num_rows, num_cols = struct.unpack_from(fmt_header, bin_data, offset)
    print('      图片数量: %d张, 图片大小: %d*%d' % (num_images, num_rows, num_cols))

```

第二步，搭建BP神经网络。参考示例代码（nueralnet.py）

```python
# coding:utf-8
import numpy as np


class NueraLNet(object):
    def __init__(self, sizes):  # 对类进行初始化
        self.num_layers = len(sizes)
        self.sizes = sizes
        self.bias = [np.random.randn(1, y) for y in sizes[1:]]  # bias中的是数组除去输入层，随机产生每层中y个神经元的bias值0-1
        # weights中的是二维数组，其中行代表前一层的节点位置，列代表后一层的节点位置，随机产生每条连线的weight值（0-1）
        self.weights = [np.random.randn(x, y) for x, y in zip(sizes[:-1], sizes[1:])]

    def get_result(self, images):  # 神经网络的前向传播部分
        result = images
        for b, w in zip(self.bias, self.weights):
            result = sigmoid(np.dot(result, w) + b)  # 加权求和以及加上bias
        return result  # 计算后每个神经元的值

    # 训练神经网络
    # trainimage表示训练集的图片，trainresult表示训练集的结果，epoch表示使用次训练集训练的次数
    # rate是学习率，默认为1，minbatch为小批梯度下降中，每一批量的样本个数，默认为10，test_image为验证集图片，test_result为验证集结果
    def train_net(self, trainimage, trainresult, epoch, rate=1, minibatch=10, test_image=None, test_result=None):
        for i in range(epoch):
            minibatchimage = [trainimage[k:k + minibatch] for k in range(0, len(trainimage), minibatch)]  # 按照小样本数量划分训练集
            minibatchresult = [trainresult[k:k + minibatch] for k in range(0, len(trainimage), minibatch)]
            for image, result in zip(minibatchimage, minibatchresult):
                self.update_net(image, result, rate)  # 根据每个小样本调用update_net函数更新网络参数
            print("epoch: {0}".format(i + 1))  # 输出迭代次数
            if test_image and test_result:  # 判断有没有验证集，有的话调用test_net函数，并查看这一次训练后神经网络监测字体的正确率
                self.test_net(test_image, test_result)
```

第三步，编写主函数，实现网络的训练，测试，保存，读入以及离线测试的功能。

参考代码（main.py）

main.py

```python
# coding: utf-8
from decodeMinist import *
from nueralnet import *
import random

# 解析Mnist训练集
print('训练集：')
train_images = decode_idx3_ubyte(train_images_idx3_ubyte_file)  # 解析Mnist训练集图像
train_labels = decode_idx1_ubyte(train_labels_idx1_ubyte_file)  # 解析Mnist训练集标签
print('测试集：')
test_images = decode_idx3_ubyte(test_images_idx3_ubyte_file)  # 解析Mnist测试集图像
test_labels = decode_idx1_ubyte(test_labels_idx1_ubyte_file)  # 解析Mnist测试集标签

# 归一化+维度转换
# #将0-255的像素值转化为0.0-1.0范围内的实数，将原有数组转化为一个1x784的新数组
trainingimages = [(im / 255).reshape(1, 784) for im in train_images]
traininglabels = [vectorized_result(int(i)) for i in train_labels]  # 训练集标签做维度转换
testimages = [(im / 255).reshape(1, 784) for im in test_images]  # 将原有数组转化为一个1x784的新数组
testlabels = [l for l in test_labels]

# 搭建BP神经网络并进行训练
net = NueraLNet([28 * 28, 30, 10])  # 输入层28*28=784个输入数据，中间层30，输出层10
net.train_net(trainingimages, traininglabels, 3, 5, 10, testimages, testlabels)  # 训练次数30次，学习率为5，每一批量的样本个数为10

# 测试
net.save_training()  # 将神经网络的权重和偏移量保存到本地
net.read_training()  # 将本地的参数读取到神经网络中
net.test_net(testimages, testlabels)  # 测试神经网络正确率
print("\n")
```

第四步，调整网络的隐藏层神经元个数，学习率等参数，观察参数的变化对网络精度的影响，记录实验结果。

要求：每组报告中至少对一个参数进行调整，取值3-5个，做表进行性能对比，分析参数对性能的影响。可选参数包括但不限于迭代次数、学习率、隐藏层神经元个数。

1.6 实验报告

1）题目：利用BP神经网络实现手写数字识别

2）总结程序运行中的报错原因并记录。

3）运行结果截图。

4）完成调整参数的性能对比，观察思考不同参数取值对识别正确率的影响。

5） 总结由本实验所获得心得体会。

主要内容

01 实验一 BP神经网络

02 实验二 LeNet网络

2.1 实验目的

1. 掌握LeNet-5网络的搭建。

2. 利用LeNet-5网络实现手写数字识别。

3. 了解影响识别正确率的因素。

2.2 实验环境

1. Windows 10

2. Python 3.6

3. Pytorch 1.0.1

2.3 实验内容

1. LeNet-5网络的介绍及搭建。

2. MNIST数据集介绍及处理。

3. 网络训练及测试。

2.4 实验原理和步骤

1） LeNet-5网络的介绍及搭建

(1) LeNet-5网络结构与原理

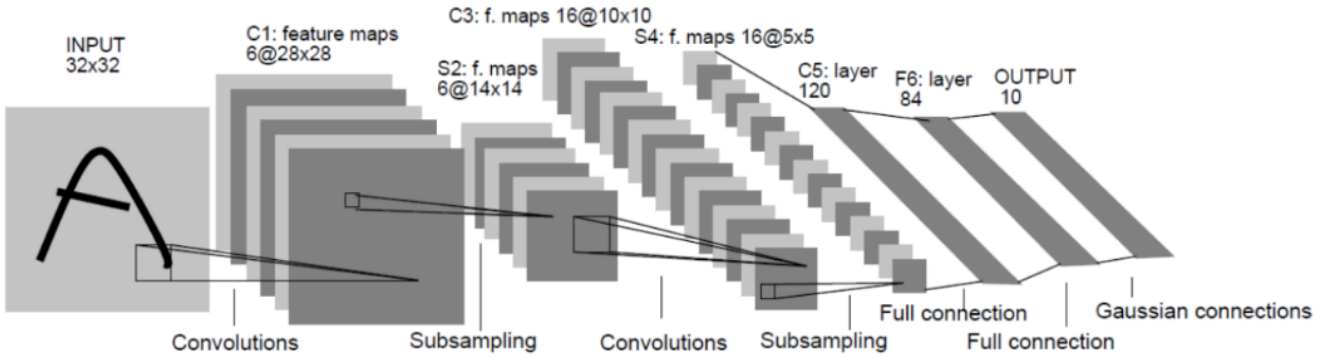


图1： LeNet-5网络的结构图

LeNet-5神经网络出自论文《Gradient-Based Learning Applied to Document Recognition》，是一种用于手写体字符识别的非常高效的卷积神经网络。LeNet-5神经网络一共包含7层（输入层不作为网络结构），分别由2个卷积层、2个池化层和3个连接层组成每层包含不同数量的训练参数。将一批数据输入进神经网络，经过卷积，激活，池化，全连接和Softmax回归等操作，最终返回一个概率数组，从而达到识别图片的目的。LeNet-5网络的结构如图1所示。

(2) LeNet-5网络的搭建

①在项目根目录下新建一个python文件，命名为“model.py”。

②导入构建网络模型所需要的包：

```
import torch
import torch.nn as nn
```

③ 定义LeNet-5网络的卷积层、池化层、全连接层：

```
# 定义LeNet类型
class LeNet5(nn.Module):

    # 初始化构造函数
    def __init__(self):
        super(LeNet5, self).__init__()
        #定义卷积层、池化层
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        #定义全连接层
        self.fc3 = nn.Linear(in_features=256, out_features=120)
        self.fc4 = nn.Linear(in_features=120, out_features=84)
        self.fc5 = nn.Linear(in_features=84, out_features=10)
```

④ 定义LeNet-5网络的前向传播：

```
# 定义前向传播
def forward(self, x):
    x = self.pool1(torch.relu(self.conv1(x)))
    x = self.pool2(torch.relu(self.conv2(x)))
    x = x.view(x.size(0), -1)
    x = torch.relu(self.fc3(x))
    x = torch.relu(self.fc4(x))
    x = self.fc5(x)
    return x
```

⑤ 运行model.py文件，一个LeNet-5网络模型就搭建成功。

2）MNIST数据集介绍及处理

(1) MNIST数据集的来源及数据组成

MNIST数据集来自美国国家标准与技术研究所，National Institute of Standards and Technology (NIST)。训练集 (training set) 由来自250个不同人手写的数字构成，其中50%是高中学生，50%来自人口普查局(the Census Bureau)的工作人员。测试集(test set)也是同样比例的手写数字数据。训练集中包含60000个样本，测试集中包含10000个样本。MNIST数据集可由官网http://yann.lecun.com/exdb/mnist/直接下载使用。

MNIST数据集的一般使用方法是，先用训练图像进行学习，再用学习到的模型对测试图像进行正确的分类。

(2) MNIST数据集的加载

① 在项目根目录下新建文件夹，命名为“data”，进入data文件夹，新建文件夹，命名为“raw”，将下载的MNIST数据集放入raw文件夹。

② 在项目根目录下新建一个python文件，命名为“data.py”。

③ 导入加载MNIST数据集需要的包：

```
from torch.utils.data import Dataset
import gzip
import os
import numpy as np
```

④ 自定义数据集类型：

```
# 自定义数据集类型
def load_data(data_folder, data_name, label_name):
    with gzip.open(os.path.join(data_folder, label_name), 'rb') as lbpath:  # rb表示的是读取二进制数据
        y_train = np.frombuffer(lbpath.read(), np.uint8, offset=8)

    with gzip.open(os.path.join(data_folder, data_name), 'rb') as imgpath:
        x_train = np.frombuffer(
            imgpath.read(), np.uint8, offset=16).reshape(len(y_train), 28, 28)
    return x_train, y_train

class MyDataset(Dataset):
    def __init__(self, folder, data_name, label_name, transform=None):
        (data_set, labels) = load_data(folder, data_name, label_name)
        self.data_set = data_set
        self.labels = labels
        self.transform = transform

    def __getitem__(self, index):
        img, target = self.data_set[index], int(self.labels[index])
        if self.transform is not None:
            img = self.transform(img)
        return img, target

    def __len__(self):
        return len(self.data_set)
```

⑤ 运行data.py文件，此时MNIST已经被加载。

3）网络训练及测试

(1) LeNet-5网络的训练

① 在项目根目录下新建一个python文件，命名为“trian.py”。

② 导入训练需要的包：

```
import torch
import torch.nn as nn
from model import LeNet5
from torch.utils.data import DataLoader
from data import MyDataset
import torchvision.transforms as transforms
```

③ 将自定义的数据集类型和网络模型进行实例化：

```
# 实例化数据集类型
trainDataset = MyDataset('data/MNIST/raw', 'train-images-idx3-ubyte.gz', 'train-labels-idx1-ubyte.gz',
                         transform=transforms.ToTensor())
# 加载训练集
data_train_loader = DataLoader(trainDataset, batch_size=256, shuffle=True, num_workers=0)

# 实例化模型
model = LeNet5()
device = torch.device("cpu") #使用电脑的cpu进行训练
print(device) #打印所用设备名称
model = model.to(device) #将模型搬移到cpu上
model.train() # 切换到训练模式
```

④ 定义训练所用优化器和交叉熵损失函数：

```
# 交叉熵损失函数
criterion = nn.CrossEntropyLoss()
# 定义优化器
lr = 0.01
optimize = torch.optim.SGD(model.parameters(), lr=lr, momentum=0.9, weight_decay=5e-4)
```

⑤ 进行迭代训练，此处设置迭代次数为5，输出训练结果，并保存模型。

```
#开始迭代
epochs = 5   #设置迭代次数
train_loss = 0
correct = 0
total = 0
for epoch in range(epochs):
    for batch_idx, (inputs, targets) in enumerate(data_train_loader):

        optimize.zero_grad() #梯度值为零
        inputs = inputs.to('cpu')
        outputs = model(inputs) #传入模型
        outputs = outputs.to(device)
        targets = targets.to(device)
        loss = criterion(outputs, targets) #计算损失
        loss.backward() #反向传播
        optimize.step() #优化器优化
        #计算训练损失
        train_loss += loss.item()
        _, predicted = outputs.max(1) #选取预测值最大的一类
        total += targets.size(0) #统计类别
        correct += predicted.eq(targets).sum().item() #计算正确个数
    #打印结果
    print('epoch:{}'.format(epoch), '------ loss:%.3f | acc:%.3f%%(%d/%d)'%(train_loss/(batch_idx+1),
                                                                            100.*correct/total, correct, total))
# 保存模型
torch.save(model, 'weight/LeNet.pkl')
```

```
cpu
epoch:0 ------ loss:1.449 | acc:53.848%(32309/60000)
epoch:1 ------ loss:1.652 | acc:73.833%(88600/120000)
epoch:2 ------ loss:1.775 | acc:81.297%(146335/180000)
epoch:3 ------ loss:1.870 | acc:85.246%(204590/240000)
epoch:4 ------ loss:1.946 | acc:87.732%(263197/300000)
```

⑥ 运行train.py文件，可以看到五次训练的结果：

(2) LeNet-5网络的测试

① 在项目根目录下新建一个python文件，命名为“test.py”。

② 导入测试需要的包：

```
import torch
import torch.nn as nn
from data import MyDataset
from torchvision.transforms import transforms
from torch.utils.data import DataLoader
```

③ 实例化测试数据集并加载模型：

```
# 实例化测试数据集
# 实例化数据集类型
testDataset = MyDataset('data/MNIST/raw', 't10k-images-idx3-ubyte.gz', 't10k-labels-idx1-ubyte.gz',
                        transform=transforms.ToTensor())
data_test_loader = DataLoader(testDataset, batch_size=256, shuffle=True, num_workers=0)

# 加载模型
model = torch.load('./weight/LeNet.pkl')
device = torch.device("cpu")# 检查cpu可用性
model = model.to(device)  # 在cpu上进行推理
model.eval()  # 切换到评估模式

criterion = nn.CrossEntropyLoss()  # 交叉熵损失函数
```

④ 测试手写数字识别准确率，打印测试结果：

```
# 开始测试
train_loss = 0
correct = 0
total = 0
with torch.no_grad():
    for batch_idx, (inputs, targets) in enumerate(data_test_loader):
        inputs = inputs.to('cpu')
        outputs = model(inputs)  # 传入模型
        outputs = outputs.to(device)
        targets = targets.to(device)
        loss = criterion(outputs, targets)  # 计算损失

        # 计算训练损失
        train_loss += loss.item()
        _, predicted = outputs.max(1)  # 选取预测值最大的一类
        total += targets.size(0)  # 统计类别
        correct += predicted.eq(targets).sum().item()  # 计算正确个数

    # 打印结果
    print(
        '------ loss:%.3f | acc:%.3f%%(%d/%d)' % (train_loss / (batch_idx + 1), 100. * correct / total, correct, total))
```

⑤ 运行test.py文件，可以看到输出训练后识别的正确率：

```
------ loss:0.075 | acc:97.770%(9777/10000)
```

(3) 训练结果的可视化展示

① 在项目根目录下新建一个python文件，命名为“show.py”。

② 导入展示识别结果需要的包：

```
import torch
import random
from torch.autograd import Variable
from torchvision import datasets, transforms
from torchvision.transforms import ToPILImage
from torch.utils.data import DataLoader
```

③ 加载测试数据集，将构建的模型放到CPU环境：

```
# 数据转化为tensor格式
data_transform = transforms.Compose([transforms.ToTensor()])

# 加载测试数据集
show_dataset = datasets.MNIST(root='./data', train=False, transform=data_transform, download=True)

# 调用net里定义的网络模型LeNet5 将数据模型转到cpu
device = 'cpu'
model = torch.load('./weight/LeNet.pkl')
model = model.to(device)  # 在cpu上进行推理
model.eval()  # 切换到评估模式
```

④ 定义存放输出结果的类：

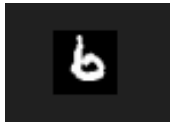
```
# 获取结果
classes = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
```

⑤ 随机加载手写数字图片进行识别：

```
# 随机加载手写字符识别
# 把tensor转化为图片，方便可视化
show = ToPILImage()

i = random.randint(0, len(show_dataset))
X, y = show_dataset[i][0], show_dataset[i][1]
show(X).show()
X = Variable(torch.unsqueeze(X, dim=0).float(), requires_grad=False).to(device)
with torch.no_grad():
    pred = model(X)
    predicted, actual = classes[torch.argmax(pred[0])], classes[y]
    print(f'figure {str(i)}: predicted: "{predicted}", actual: "{actual}"')
```

⑥ 运行show.py文件，可以得到随机的一张图像和它的识别结果：



```
figure 587: predicted: "6", actual: "6"
```

⑦ 在之前的实验中，我们设置了迭代训练的次数为5，得到识别的准确率为97.770%，现在我们更改迭代次数，观察不同的训练次数对识别正确率的影响。

2.5 实验报告要求

（1）题目：利用LeNet5神经网络实现手写数字识别

（2）总结程序运行中的报错原因并记录。

（3）运行结果截图。

（4）完成调整参数的性能对比，观察思考不同参数取值对识别正确率的影响。

（5） 总结由本实验所获得心得体会。

人工智能理论与实践

第五章 人工神经网络与深度学习 实验

信息学部 信息与通信工程学院