**《最优化技术》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2019级计算机科学与技术卓越02班** | | | **姓名** | **李燕琴** |
| **实验题目** | 梯度下降法在人工神经网络的应用 | | | | | |
| **实验时间** | **2021年6月1日** | | **实验地点** | **DS3401** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 ■设计性 □综合性** | | |
| 教师评价：  □算法/实验过程正确；□源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | | |
| 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, 它包含了四个部分:   * Training set images: train-images-idx3-ubyte.gz (9.9 MB, 解压后 47 MB, 包含 60,000 个样本) * Training set labels: train-labels-idx1-ubyte.gz (29 KB, 解压后 60 KB, 包含 60,000 个标签) * Test set images: t10k-images-idx3-ubyte.gz (1.6 MB, 解压后 7.8 MB, 包含 10,000 个样本) * Test set labels: t10k-labels-idx1-ubyte.gz (5KB, 解压后 10 KB, 包含 10,000 个标签)   直接读取压缩包里的训练数据集参考代码如下，测试数据集的代码类似。  import gzip  import struct  import numpy as np  def load\_mnist\_train(path, kind='train'):  ‘’’path:数据集的路径  kind:值为train，代表读取训练集  ‘’’  labels\_path = os.path.join(path,'%s-labels-idx1-ubyte.gz'% kind)  images\_path = os.path.join(path,'%s-images-idx3-ubyte.gz'% kind)  #使用gzip打开文件  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  #这样读到的前两个数据分别是magic number和样本个数  magic, n = struct.unpack('>II',lbpath.read(8))  #使用np.fromstring读取剩下的数据，lbpath.read()表示读取所有的数据  labels = np.fromstring(lbpath.read(),dtype=np.uint8)  with gzip.open(images\_path, 'rb') as imgpath:  magic, num, rows, cols = struct.unpack('>IIII',imgpath.read(16))  images = np.fromstring(imgpath.read(),dtype=np.uint8).reshape(len(labels), 784)  return images, labels  说明：load\_mnist 函数返回两个数组, 第一个是一个 n x m 维的 NumPy array(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 10,000 样本. 在 MNIST 数据集中的每张图片由 28 x 28 个像素点构成, 每个像素点用一个灰度值表示。 load\_mnist 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).  显示数据集中的某个数字参考代码如下：  import matplotlib.pyplot as plt  X\_train,y\_train = load\_mnist(file path)  fig, ax = plt.subplots(  nrows=2,  ncols=5,  sharex=True,  sharey=True, )  ax = ax.flatten()  for i in range(10):  img = X\_train[y\_train == i][0].reshape(28, 28)  ax[i].imshow(img, cmap='Greys', interpolation='nearest')  ax[0].set\_xticks([])  ax[0].set\_yticks([])  plt.tight\_layout()  plt.show()    注意：提交本电子文档和程序，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）  import gzip  import os  import struct  import numpy as np  import time  import torch  import torchvision  from torch import nn  from torch.utils.data import Dataset, DataLoader  import torch.nn.functional as F  import matplotlib.pyplot as plt  transform = torchvision.transforms.Compose([      torchvision.transforms.ToTensor(),      torchvision.transforms.Normalize((0.1307,), (0.3081,))  ])  batch\_size = 70  input\_size = 28  output\_size = 10  num\_epochs = 50  momentum = 0.5  lr = 0.001  device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  save\_path = './checkpoint.pth'  record\_path = './train\_record.pth'  def load\_mnist(path, kind='train'):      '''      :param path: 数据集的路径 \n      :param kind:      - train，代表读取训练集      - t10k, 代表读取测试集 \n      :return:      - images: nxm-D numpy.array(images), n 样本数，m 特征数      - labels: n-D，目标变量（手写数字的类标签） \n      '''      labels\_path = os.path.join(path, '%s-labels-idx1-ubyte.gz' % kind)      images\_path = os.path.join(path, '%s-images-idx3-ubyte.gz' % kind)      # 使用gzip打开文件      with gzip.open(labels\_path, 'rb') as lbpath:  # 'rb'表示读取的是二进制          # 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          # 这样读到的前两个数据分别是 magic number 和 样本个数          magic, n = struct.unpack('>II', lbpath.read(8))          # 使用np.fromstring读取剩下的数据，lbpath.read()表示读取所有的数据          labels = np.fromstring(lbpath.read(), dtype=np.uint8)      with gzip.open(images\_path, 'rb') as imgpath:          magic, num, rows, cols = struct.unpack('>IIII', imgpath.read(16))          images = np.fromstring(imgpath.read(), dtype=np.uint8).reshape(len(labels), 28, 28)      return images, labels  class MyDataset(Dataset):      ''' 自定义数据集 '''      def \_\_init\_\_(self, folder\_path, kind='train', transform=None):          ''' 初始化一些需要传入的参数及数据集的调用 '''          super(MyDataset, self).\_\_init\_\_()  # 对继承自父类的属性进行初始化          images, labels = load\_mnist(folder\_path, kind=kind)          self.images = images  # 获取图片          self.labels = labels  # 获取标签          self.transform = transform  # 转化      def \_\_getitem\_\_(self, index):          '''          对数据进行预处理并返回想要的信息          return回哪些内容，那么在训练时循环读取每个batch时，就能获得哪些内容          '''          img = self.images[index]          label = self.labels[index]          if self.transform:              img = self.transform(img)          return img, label      def \_\_len\_\_(self):          ''' 初始化一些需要传入的参数及数据集的调用 '''          return len(self.labels)  class MyCNN(nn.Module):      ''' 自定义CNN网络结构 '''      def \_\_init\_\_(self):          super(MyCNN, self).\_\_init\_\_()          # input\_size [1,28,28]          self.conv1 = nn.Sequential(              nn.Conv2d(1, 10, kernel\_size=5),  # [10,24,24]  28-5+1=24 卷积层              nn.ReLU(),  # 激活层              nn.MaxPool2d(2),  # [10,12,12] 池化层          )          self.conv2 = nn.Sequential(              nn.Conv2d(10, 20, kernel\_size=5),  # [b,20,8,8] 12-5+1=8              nn.ReLU(),              nn.MaxPool2d(2),  # [b,20,4,4]          )          # self.conv2\_drop = nn.Dropout2d()          self.fc1 = nn.Sequential(              nn.Linear(20 \* 4 \* 4, 50),  # 全连接线性层              nn.ReLU(),  # 全连接激活层          )          self.fc2 = nn.Linear(50, output\_size)  # 输出层      def forward(self, x):          x = self.conv1(x)          x = self.conv2(x)          x = x.view(x.size(0), -1)          x = self.fc1(x)          out = self.fc2(x)          return out  def train(dataloader, model, lossfunc, optimizer, num\_epochs=25, save\_path=None, record\_path=None):      """      :param dataloader: 数据集      :param model: 网络模型      :param lossfunc: 损失函数      :param optimizer: 优化器      :param num\_epochs: 循环训练次数      :param save\_path: 保存模型路径      :param record\_path: 保存训练参数路径      :return: model 训练模型， loss\_list,acc\_list 训练记录      """      start = time.time()      loss\_list = []      acc\_list = []      initepoch = 0      # 加载保存参数      if save\_path and os.path.exists(save\_path):          print('加载保存模型...')          checkpoint = torch.load(save\_path)          model.load\_state\_dict(checkpoint['model\_state\_dict'])          optimizer.load\_state\_dict(checkpoint['optimizer\_state\_dict'])          initepoch = checkpoint['epoch']      if record\_path and os.path.exists(record\_path):          print('加载训练记录...')          record = torch.load(record\_path)          loss\_list = record['loss']          acc\_list = record['acc']      for epoch in range(initepoch, num\_epochs):          epoch\_start = time.time()          model.train()          batch\_loss = 0.0          batch\_acc = 0          for images, labels in dataloader:              images = images.to(device)              labels = labels.to(device)              labels = torch.tensor(labels, dtype=torch.long)              optimizer.zero\_grad()              with torch.set\_grad\_enabled(True):                  outputs = model(images)                  loss = lossfunc(outputs, labels)                  \_, preds = torch.max(outputs, 1)                  loss.backward()                  #梯度下降                  with torch.no\_grad():                      for param in model.parameters():                          param-=lr\*param.grad  #参数更新                  # optimizer.step()              batch\_loss += loss.item() \* images.size(0)              batch\_acc += torch.sum(preds == labels.data)          epoch\_loss = batch\_loss / len(dataloader.dataset)          epoch\_acc = batch\_acc.double() / len(dataloader.dataset)          loss\_list.append(epoch\_loss)          acc\_list.append(epoch\_acc)          print('Epoch {}/{}:\tLoss={:.4f}\tAcc={:.4f}\tTime={:.0f}s'              .format(epoch, num\_epochs - 1, epoch\_loss, epoch\_acc,time.time() - epoch\_start))          # 保存模型          torch.save({'epoch': epoch,                      'model\_state\_dict': model.state\_dict(),                      'optimizer\_state\_dict': optimizer.state\_dict()                      }, save\_path)          torch.save({              'loss': loss\_list,              'acc': acc\_list          }, record\_path)      print('Training complete in {:.0f}s'.format(time.time() - start))      return model, loss\_list, acc\_list  def test(dataloader, model, lossfunc,save\_path):      # model.eval()  # 如果没有BN层和dropout层，这个就不需要了      test\_acc = 0      test\_loss = 0.0      start = time.time()      if save\_path and os.path.exists(save\_path):          print('加载保存模型...')          checkpoint = torch.load(save\_path)          model.load\_state\_dict(checkpoint['model\_state\_dict'])      for inputs, labels in dataloader:          inputs = inputs.to(device)          labels = labels.to(device)          labels = torch.tensor(labels, dtype=torch.long)          with torch.set\_grad\_enabled(False):              outputs = model(inputs)              loss = lossfunc(outputs, labels)              \_, preds = torch.max(outputs, 1)              test\_loss += loss.item() \* inputs.size(0)              test\_acc += torch.sum(preds == labels.data)      acc = test\_acc.double() / len(dataloader.dataset)      loss = test\_loss / len(dataloader.dataset)      print('Test Result:\tLoss={:.4f}\tAcc={:.4f}\tTime={:.0f}s'          .format(loss, acc, time.time() - start))  # 模型建立  model = MyCNN()  model.to(device)  loss\_func = nn.CrossEntropyLoss()  optimizer = torch.optim.SGD(model.parameters(), lr=lr, momentum=momentum)  # 数据集加载  train\_dataset = MyDataset('./mnist', kind='train', transform=transform)  train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)  # shuffle 是否打乱顺序  # 开始训练  model, train\_loss, train\_acc = train(train\_loader, model, loss\_func,      optimizer, num\_epochs, save\_path=save\_path,record\_path=record\_path)  # 训练过程可视化  plt.figure(1)  plt.plot(range(0, len(train\_loss)), train\_loss)  plt.title('Loss Value')  plt.show()  plt.figure(2)  plt.plot(range(0, len(train\_acc)), train\_acc)  plt.title('Accuracy Value')  plt.show()  # 加载测试集  test\_dataset = MyDataset('./mnist', kind='t10k', transform=transform)  test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=True)  # 开始测试  test(test\_loader, model, loss\_func,save\_path) | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程  1、梯度下降的使用：    2、30个epoch训练过程中，损失值和准确率的变化    图 1 损失值的变化    图 2 准确率的变化  3、最终测试结果：  【Test Result: Loss=0.0916 Acc=0.9746 Time=3s】  准确率可以达到97.46%，所建立的cnn模型具有一定的实用性。 | | | | | | |

注：电子文档命名要求：学号+姓名+实验序号