**《最优化技术》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **20194196、计算机科学与技术、01班** | | | **姓名** | **谢双骏** |
| **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4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：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.基本思想**  （1）首先是神经元的定义：  0](2{U93WJKL4J7WPL]IORA  假设,,，为外界的输入，那么输出为：  （2）激活函数,输入层不会对数据进行处理。隐藏层和输出层均采用Sigmoid函数作为激活函数：  把神经元的输出，放在激活函数里面，就形成了标准化之后的值  （3）接下来是人工神经网络的定义：  T}]SNGX3HL$H9B]9(AXC(YT   1. 工作信号正向传播   W28BG1A5B])3S3HRW$53J}9  人工神经网络一共有三层，一个输入层，一个隐含层，一个输出层。最初我们只知道输入层的初始数据，我们需要将输入一层一层的往前推进，最后得到输出层的值。首先由输入层得到隐含层：  其次由隐含层得到输出层:   1. 误差信号反向传播   最优化技术+9-ann_20最优化技术+9-ann_21最优化技术+9-ann_22最优化技术+9-ann_23最优化技术+9-ann_24  其中权重的变化值的实际来源其实就是误差的负梯度方向乘上学习率也就是步长，本质其实就是梯度下降法。   1. 迭代更新部分   根据第五步求出的权重变化值，不断更新隐含层和输出层的权重和偏置：  隐含层权重=隐含层权重+隐含层权重变化  隐含层偏置=隐含层偏置+隐含层偏置变化  输出层权重=输出层权重+输出层权重变化  输出层偏置=输出层偏置+输出层偏置变化   1. 得到训练好的网络之后，对测试集进行测试，并记录下准确率，以便绘制曲线 2. 总结   人工神经网络其实类似于数学模型的层次分析法（个人观点，除了激活函数的部分），只是我们运用了训练的样本，去不断更新不同层次之间的权重和偏置。  **2.代码部分**  # coding:utf-8 import os import gzip import struct import numpy as np import matplotlib.pyplot as plt plt.rcParams['font.sans-serif']=['SimHei'] #图中文字体设置为黑体 plt.rcParams['axes.unicode\_minus']=False #负值显示 import time #实验报告所给函数，此处不添加注释了 """ 加载mnist数据集为numpy数组 path: 数据集的路径 kind: train=训练集; t10k=测试集 """ def load\_mnist(path, kind='train'):  labels\_path = os.path.join(path, '%s-labels-idx1-ubyte.gz' % kind)  images\_path = os.path.join(path, '%s-images-idx3-ubyte.gz' % kind)  with gzip.open(labels\_path, 'rb') as lbpath:  struct.unpack('>II', lbpath.read(8))  labels = np.frombuffer(lbpath.read(), dtype=np.uint8)  with gzip.open(images\_path, 'rb') as imgpath:  struct.unpack('>IIII', imgpath.read(16))  images = np.frombuffer(imgpath.read(), dtype=np.uint8)  images = images.reshape(len(labels), 784)  return images, labels """ 把输入（图片的像素灰度值）的定义域从[0,255]映射到[0,1] 把标签，也就是是什么数字转换成one-hot编码 """ def normalize(images, labels):  return images/255, np.eye(10)[labels] #人工神经网络类 class ANN:  #输入数据，隐藏层大小，输出大小，学习率  def \_\_init\_\_(self,input\_number,hide\_number,output\_number,learn\_rate):  # 设置激活函数 sigmoid  self.activation = lambda x: 1.0 / (1.0 + np.exp(-x))  #设置输入层大小  self.input\_number=input\_number  #设置隐藏层大小  self.hide\_number = hide\_number  #设置输出层大小  self.output\_number = output\_number  #设置学习率  self.learn\_rate = learn\_rate  #设置输入层和隐藏层之间的权重  self.weight\_input\_hide=np.random.normal(0.0,1e-2,(hide\_number,input\_number))  #设置隐藏层和输出层之间的权重  self.weight\_hide\_output=np.random.normal(0.0,1e-2,(output\_number,hide\_number))  # 设置输入层和隐藏层之间的偏置  self.bias\_input\_hide=np.zeros((hide\_number,1))  # 设置隐藏层和输出层之间的偏置  self.bias\_hide\_output=np.zeros((output\_number,1))  #工作信号向前传播  def forward\_spread(self):  #隐藏层的输出  self.hide\_output=self.activation(np.dot(self.weight\_input\_hide,self.sample)+self.bias\_input\_hide)  #输出层的输出  self.out\_output=self.activation(np.dot(self.weight\_hide\_output,self.hide\_output)+self.bias\_hide\_output)   #误差信号反向传播  def reverse\_spread(self):  #输出层的误差  self.output\_error=(self.label-self.out\_output)\*self.out\_output\*(1-self.out\_output)  #隐藏层的误差  self.hide\_error=np.dot(self.weight\_hide\_output.T,self.output\_error)\*self.hide\_output\*(1-self.hide\_output)   #梯度的变化值  def grad\_iteration(self):  #输出层的权重的梯度的变化值  self.grad\_output\_weight=self.learn\_rate\*np.dot(self.output\_error,self.hide\_output.T)  # 输出层的偏置的梯度的变化值  self.grad\_output\_bias=self.learn\_rate\*self.output\_error  # 隐藏层的权重的梯度的变化值  self.grad\_hide\_weight=self.learn\_rate\*np.dot(self.hide\_error,self.sample.T)  # 隐藏层的偏置的梯度的变化值  self.grad\_hide\_bias=self.learn\_rate\*self.hide\_error   #迭代更新  def iteration(self):  #隐藏层的权重  self.weight\_input\_hide=self.weight\_input\_hide+self.grad\_hide\_weight  # 隐藏层的偏置  self.bias\_input\_hide=self.bias\_input\_hide+self.grad\_hide\_bias  # 输出层的权重  self.weight\_hide\_output=self.weight\_hide\_output+self.grad\_output\_weight  # 输出层的偏置  self.bias\_hide\_output=self.bias\_hide\_output+self.grad\_output\_bias   #训练  def train(self,sample,label):  #初始化输入  self.sample=np.array(sample,ndmin=2).T  #初始化正确输出  self.label=np.array(label,ndmin=2).T  #工作信号向前传播  self.forward\_spread()  #误差信号向后传播  self.reverse\_spread()  #根据梯度求权值和偏置的迭代更新变化值  self.grad\_iteration()  #迭代更新  self.iteration()  #记录准确率  def learn\_degree(self, samples, labels):  # 初始化输入  self.sample = np.array(samples, ndmin=2).T  # 初始化正确输出  self.forward\_spread()  """  相当于找出向量中哪个下标是1，因为label的向量是one-hot形式  然后比较下标是否一致，就可以看出来预测是否正确  如果真实值是9，那么真实的label向量应该是[0,0,0,0,0,0,0,0,1,0]  如果预测值是9，那么预测的label向量应该是[0,0,0,0,0,0,0,0,1,0]  返回的下标是9，9，比较相同就表示预测正确，反之就表示预测错误  """  label1 = np.argmax(self.out\_output.T, axis=1)  label2 = np.argmax(labels, axis=1)  return sum(label1 == label2) / labels.shape[0] #main 函数 if \_\_name\_\_ == '\_\_main\_\_':  #初始化参数  input\_number=784  hide\_number=200  output\_number=10  learn\_rate=0.4  #实例化模型  ann=ANN(input\_number,hide\_number,output\_number,learn\_rate)  #处理数据  picture\_train, label\_train = load\_mnist('./mnist', kind='train')  picture\_train, label\_train = normalize(picture\_train, label\_train)  picture\_test, label\_test = load\_mnist('./mnist', kind='t10k')  picture\_test, label\_test = normalize(picture\_test, label\_test)  #学习程度记录  learn\_degree = []  #训练  print('开始训练')  print('-----------------------------------')  time\_start = time.time()  for i in range(len(picture\_train)):  if (i + 1) % 500 == 0:  print('训练到第%d张...' %(i + 1))  #每多训练500次，就做一次测试，并记录学习率  learn\_degree.append(ann.learn\_degree(picture\_test, label\_test))   ann.train(picture\_train[i], label\_train[i])  time\_end = time.time()  print('训练结束')  print('-----------------------------------')  print('学习率: '+str(ann.learn\_rate))  print('隐藏层大小: '+str(ann.hide\_number))  print('准确率: ', learn\_degree[-1])  print('花费时间: ', time\_end - time\_start, 's')  #画图  x=np.arange(500,60001,500)  plt.plot(x,learn\_degree,color='orange',linestyle='--')  plt.xlabel("训练量")  plt.ylabel("准确率")  plt.title("ANN识别手写数字")  plt.show() | | | | | | |
| **四、实验结果及分析和（或）源程序调试过程**  **（一）结果**  **1.控制台**  开始训练  -----------------------------------  训练到第500张...  （篇幅太长，省略了）  训练到第60000张...  训练结束  -----------------------------------  学习率: 0.4  隐藏层大小: 200  准确率: 0.9571  花费时间: 105.70022583007812 s  进程已结束，退出代码 0  **2.图像**     1. **调试过程**   1.最开始的时候，对于权值的随机数生成，一点限制都没有采用，发现学习的准确率非常低，在0.1-0.3左右。后来在查阅网上文献后，把随机数的生成更改为标准正态分布，发现准确率虽然有上升，但是还是不高，大概维持在0.5-0.6左右，后来继续更改随机数的生成，使用正态分布，把标准差调整为0.01，发现学习的准确率提升到了0.95左右，还是达到了预期的效果  **2.调试参数**   |  |  |  | | --- | --- | --- | | 学习率 | 隐含层神经元个数 | 准确率 | | 0.1 | 100 | 0.9268 | | 0.2 | 100 | 0.9435 | | 0.3 | 100 | 0.9468 | | 0.4 | 100 | 0.9523 | | 0.1 | 200 | 0.9267 | | 0.2 | 200 | 0.9467 | | 0.3 | 200 | 0.9507 | | 0.4 | 200 | 0.9571 | | 0.1 | 300 | 0.9281 | | 0.2 | 300 | 0.9449 | | 0.3 | 300 | 0.9549 | | 0.4 | 300 | 0.9550 | | 0.1 | 400 | 0.9244 | | 0.2 | 400 | 0.9455 | | 0.3 | 400 | 0.9545 | | 0.4 | 400 | 0.9569 |   最后发现，其他条件不变的情况下，选取0.4的学习率，200的隐含层神经元个数是最好 | | | | | | |