**《机器学习基础》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2021级计算机科学与技术（卓越）1班** | | | **姓名** | **韩昊辰** |
| **实验题目** | **BP算法实践** | | | | | |
| **实验时间** | **2023/5/8** | | **实验地点** | **重庆大学图书馆** | | |
| **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 掌握BP算法原理并编程实践。 | | | | | | |
| 二、实验项目内容 1.理解并描述BP算法原理。  BP神经网络，也被称为误差反向传播神经网络（Backpropagation Neural Network），是一种前馈式人工神经网络。它由输入层，隐藏层和输出层组成。  首先，输入层接收外部输入，将其处理为一维向量，并通常标准化为取值范围在[0,1]之间的特征向量。然后输入层将输入传递给隐藏层，隐藏层通过激活函数（如relu函数）将加权输入进行非线性转换，并将该结果传递给输出层。输出层将加权的输入传输给输出层，输出层会定义一个误差计算函数（如MSE、交叉熵函数等）预测结果和实际结果之间的误差，通过误差反向传播算法来更新隐藏层和输出层之间的权重。该算法基于梯度下降法，通过计算误差对权重的偏导数来更新各层之间的权重。不断重复上述步骤，直到误差最小。  BP神经网络训练方法为：首先正向传播，将输入数据通过神经网络，得到预测结果。然后计算误差，将预测结果和真实结果之间的误差作为损失函数。最后反向传播，通过误差反向传播算法，根据损失函数对神经网络中所有参数进行梯度下降更新。  BP神经网络的优点是可以学习到复杂的非线性关系，并且可以逐步逼近函数的最优解。其缺点是需要较长的训练时间和大量的样本数据。  2.编程实践，将算法应用于合适的分类数据集 (如鸢尾花、UCI数据集、Kaggle数据集)，要求算法至少用于两个数据集。  详见三 | | | | | | |
| 三、实验过程或算法（源程序）模块解析及实现 大体上，本实验的BP神经网络由两大模块实现，分别为全连接神经网络搭建模块和神经网络训练寻优模块。神经网络搭建模块可细分为全连接层模块，激活函数模块以及损失计算模块；而神经网络训练寻优模块则细分为训练模块，寻优模块和模型评估模块。 神经网络搭建模块 神经网络搭建模块的功能是接收训练集属性值和标签，搭建输入层，隐藏层和输出层，计算loss，误差反向传导，梯度下降法更新层与层之间的权值，使得误差最小化。loss达到容忍值或达到迭代次数上限输出loss，样本预测值。其内部保存了神经网络所有的大量的权重，并且可以在测试集上根据输入的属性预测，和测试集标签做对比，评估其泛化能力。    神经网络搭建模块包含三种子模块，分别为全连接层模块2个，激活函数模块1个以及损失计算模块1个，如图所示。每个模块有特定的维护数据和功能实现。对于维护数据，三种模块均维护梯度这个变量，全连接层维护权值、偏置、两者的梯度以及隐藏层维度，对于实现功能，三种模块均有正向传播、反向传播和打印本模块维护变量的功能函数，而全连接层独有更新自身权重和偏置的功能函数。输入层直接接收属性值向量X，不做变换。 全连接层模块  * 全连接层维护：\_W 权值，\_b 偏置， \_grad\_W 权值梯度， \_grad\_b 偏置梯度，X本层输入 * 全连接层初始化：用户需要自定义全连接层节点个数，对于本实验的神经网络，输入节点个数为X的属性值个数，输出节点个数为Y标签值种类，隐藏层维度视问题而定，在后续寻优模块中，会测试不同隐藏层维度对结果的影响以确定最优隐藏层维度。权值和偏置使用np.random.normal函数赋正态分布的随机值，注意不能全为0。   代码：  # mem: \_W 权值，\_b 偏置， \_grad\_W 权值梯度， \_grad\_b 偏置梯度，X本层输入 def \_\_init\_\_(self, input\_D, output\_D=28): # 初始化权值、偏置以及两者的梯度，input\_D:输入节点个数，output\_D输出节点个数  self.mem = {}  self.mem["\_W"] = np.random.normal(0, 0.1, (input\_D, output\_D)) # 初始化不能为全0  self.mem["\_b"] = np.random.normal(0, 0.1, (1, output\_D))  self.mem["\_grad\_W"] = np.zeros((input\_D, output\_D))  self.mem["\_grad\_b"] = np.zeros((1, output\_D))   * 前向传播：前向实际上在前向传播中实现的就是矩阵乘法运算：（1）   通过前向传播更新模块维护数据“X”  代码：  def forward(self, X): # 输入上一层的节点值，输出H(relu激活输入值)  self.mem["X"] = X  \_W = self.mem["\_W"]  \_b = self.mem["\_b"]  H = np.matmul(X, \_W) + \_b  return H   * 在反向传播中，我们需要根据（1）式计算得到 X 和 W 的梯度：，通过反向传播更新模块维护数据梯度   代码：  def backward(self, grad\_H): # 由上一层节点值和下一层的loss关于output的梯度共同决定本层梯度，输出给上一层的梯度  X = self.mem["X"]  \_W = self.mem["\_W"]  self.mem["\_grad\_W"] = np.matmul(X.T, grad\_H)  self.mem["\_grad\_b"] = np.matmul(grad\_H.T, np.ones(X.shape[0]))  # print("hi",grad\_H)  # print(\_W.T)  grad\_X = np.matmul(grad\_H, \_W.T)  return grad\_X   * 更新权重代码：   def update(self, learn\_rate):  self.mem["\_W"] = self.mem["\_W"] - self.mem["\_grad\_W"] \* learn\_rate  self.mem["\_b"] = self.mem["\_b"] - self.mem["\_grad\_b"] \* learn\_rate   * 打印mem信息：   def Print(self):  print(self.mem) 激活函数模块 激活函数使用relu函数，它是目前深度神经网络中经常使用的激活函数，ReLU 实际上是一个斜坡（ramp）函数，定义为：    在反向传播中进行求梯度得：    激活函数模块维护输入输出和梯度三个变量。  模仿全连接模块，实现激活函数模块代码如下：  class Relu():  # mem: X 输入，OUT 输出，grad 梯度  def \_\_init\_\_(self):  self.mem = {}   def forward(self, X):  self.mem["X"] = X  self.mem["OUT"] = np.where(X < 0, np.zeros\_like(X), X)  return self.mem["OUT"]   def backward(self, grad\_H):  X = self.mem["X"]  self.mem["grad"] = (X > 0).astype(np.float32) \* grad\_H  return self.mem["grad"]   def Print(self):  print(self.mem) 损失计算模块 损失函数使用MSE函数，即均方误差损失。它是机器学习中一个常用的回归问题的损失函数，用于度量模型输出和真实标签之间的差异程度。计算方法是将每个预测值与真实值之差的平方求和，然后求平均数，用python自定义函数实现如下：  # 编写MSE损失函数 def mse\_loss(y\_true, y\_pred, size\_average=True, reduce=True):  # Compute the difference between the predicted values and the true values  diff = y\_pred - y\_true   # Compute the squared error for each sample  sq\_error = np.square(diff)   # Compute the mean squared error across all samples  if size\_average and reduce:  mse\_loss = np.mean(sq\_error)  elif reduce:  mse\_loss = np.sum(sq\_error)  else:  mse\_loss = sq\_error  return mse\_loss  损失计算模块基于MSE损失函数，分析全连接层输出的样本预测值、样本真实标记，在前向传播过程中给出loss值，在方向传播过程中计算loss关于样本预测值的梯度传导给前向模块，代码实现如下：  # 实现MSE损失函数 class MSEloss():  # mem: predict\_y 样本预测值，label\_Y 样本真是标记，grad 梯度  def \_\_init\_\_(self):  self.mem = {}   def forward(self, predict\_y, label\_y):  self.mem["predict\_y"] = predict\_y  self.mem["label\_y"] = label\_y  loss = mse\_loss(label\_y,predict\_y)  return loss   def backward(self):  train\_y = self.mem["label\_y"]  y = self.mem["predict\_y"]  self.mem["grad"] = (y - train\_y).reshape(y.shape[0],len(train\_y[0]))  return self.mem["grad"]   def Print(self):  print(self.mem) 1.1.4搭建全连接神经网络模型   如图，用户指定输入层、隐藏层、输出层维度，初始化2个全连接模块，1个relu激活模块，1个mseloss模块，使用临时变量连接模块，实现神经网络模型的搭建。  代码实现如下：  # 搭建全连接神经网络模型 class FullConnectionModel():  # mem: input\_dims 输入层维度，latent\_dims隐藏层维度，output\_dims 输出层维度，predict\_y 样本预测值，loss loss值  # 1个隐藏层，1个relu激活层，1个输出层，1个损失计算层  def \_\_init\_\_(self, input\_D, latent\_dims, output\_D):  self.mem = {}  self.Mul\_H1 = LinearLayer(input\_D, latent\_dims)  self.relu = Relu()  self.Mul\_H2 = LinearLayer(latent\_dims, output\_D)  self.MSE\_Loss = MSEloss()  self.mem["input\_dims"] = input\_D  self.mem["latent\_dims"] = latent\_dims  self.mem["output\_dims"] = output\_D   def forward(self, X, y\_labels):  o0 = X  a1 = self.Mul\_H1.forward(o0)  o1 = self.relu.forward(a1)  a2 = self.Mul\_H2.forward(o1)  o2 = a2  # print("o2:",o2)  # print(o2.shape)  self.mem["predict\_y"] = y = o2.reshape(y\_labels.shape)  self.mem["loss"] = self.MSE\_Loss.forward(predict\_y=y, label\_y=y\_labels)   def backward(self):  grad\_loss\_to\_h2 = self.MSE\_Loss.backward()  grad\_h2\_to\_relu = self.Mul\_H2.backward(grad\_loss\_to\_h2)  grad\_relu\_to\_h1 = self.relu.backward(grad\_h2\_to\_relu)  grad\_X = self.Mul\_H1.backward(grad\_relu\_to\_h1)   def upgrade(self, learn\_rate):  self.Mul\_H1.update(learn\_rate)  self.Mul\_H2.update(learn\_rate)   def Print(self):  print("predict\_y:",self.mem["predict\_y"])  print("loss:", self.mem["loss"])  调用时，用户只需要输入三个层的维度即可创建神经网络模型。 1.2 神经网络训练寻优模块 到上一步整个全连接神经网络框架基本已经搭建完成了，在这一步我们就可以对模型进行训练和评估，根据划分好的训练集、验证集和测试集，我们可以借助验证集进行参数寻优，找出最好的参数组合，然后将由这组参数训练来的模型在测试集上进行测试。 1.2.1 模型训练模块 模型训练模块通过将训练一次模型封装在类中实现。模块接收训练集的属性值和标签，指定学习率，调用神经网络模型的前向传播，反向传播和更新功能，对模型参数进行调整，并送入模型评估模块报告模型损失值和精度。  代码：  # 训练一次模型 def trainOneStep(model, x\_train, y\_train, learning\_rate=1e-5):  model.forward(x\_train, y\_train)  model.backward()  model.upgrade(learning\_rate)  loss = model.mem["loss"]  accuracy = computeAccuracy(model.mem["predict\_y"], y\_train)  return loss, accuracy 1.2.2 模型评估模块 模型评估模块要做的事就是报告模型损失值和精度。损失值就是MSELoss的函数结果，被保存在神经网络模块的维护数据中，精度的计算方法是计算样本预测种类（预测值最大的输出节点对应的种类）和真实种类如果相同，计数+1，最后计数/总样本数=精度。计算实现如下：  # 计算精确度 def computeAccuracy(prob, labels):  predicitions = np.argmax(prob, axis=1)  truth = np.argmax(labels, axis=1)  return np.mean(predicitions == truth)  装进模块：  # 评估模型 def evaluate(model, x, y):  model.forward(x, y)  loss = model.mem["loss"]  accuracy = computeAccuracy(model.mem["predict\_y"], y)  return loss, accuracy 1.2.3 模型寻优模块 模型寻优模块其实也包含对模型不断迭代求解的训练过程。但是模块会先让用户选择一组隐藏层维度测试值，默认为[100 200 300]，指定迭代次数（默认10000），学习率（默认1e-5），传入训练集、验证集，开始训练。  训练和寻优过程为：对于每一个隐藏层维度，用训练集不断训练神经网络模块，在loss降到容忍值或达到迭代次数上限时停止训练，送进模型评估模块，记录返回结果。最后，在所有隐藏层维度对应结果中挑选最好的一个，作为模型参数传出来。  代码如下：  # 模型寻优 def train(x\_train, y\_train, x\_validation, y\_validation,epochs = 10000, latent\_dims\_list = [100, 200, 300],learning\_rate = 1e-5):  best\_accuracy = 0  best\_latent\_dims = 0  output\_D = len(y\_train[0])  # 在验证集上寻优  print("Start seaching the best parameter...\n")  for latent\_dims in latent\_dims\_list:  model = FullConnectionModel(len(x\_train[0]),latent\_dims,output\_D)   bar = trange(20) # 使用 tqdm 第三方库，调用 tqdm.std.trange 方法给循环加个进度条  for epoch in bar:  loss, accuracy = trainOneStep(model, x\_train, y\_train, learning\_rate)  bar.set\_description(f'Parameter latent\_dims={latent\_dims: <3}, epoch={epoch + 1: <3}, loss={loss: <10.8}, accuracy={accuracy: <8.6}') # 给进度条加个描述  bar.close()   validation\_loss, validation\_accuracy = evaluate(model, x\_validation, y\_validation)  print(f"Parameter latent\_dims={latent\_dims: <3}, validation\_loss={validation\_loss}, validation\_accuracy={validation\_accuracy}.\n")   if validation\_accuracy > best\_accuracy:  best\_accuracy = validation\_accuracy  best\_latent\_dims = latent\_dims   # 得到最好的参数组合，训练最好的模型  print(f"The best parameter is {best\_latent\_dims}.\n")  print("Start training the best model...")  best\_model = FullConnectionModel(len(x\_train[0]),best\_latent\_dims,output\_D)  x = np.concatenate([x\_train, x\_validation], axis=0)  y = np.concatenate([y\_train, y\_validation], axis=0)  bar = trange(epochs)  for epoch in bar:  loss, accuracy = trainOneStep(best\_model, x, y, learning\_rate)  bar.set\_description(f'Training the best model, epoch={epoch + 1: <3}, loss={loss: <10.8}, accuracy={accuracy: <8.6}') # 给进度条加个描述  bar.close()   return best\_model  这里我们贴心地使用 tqdm 第三方库，调用 tqdm.std.trange 方法给循环加个进度条，因为神经网络训练通常耗时巨大，更不用说我们寻优要训练好几个模型，所以给用户一个可视化的进度让用户看着放心，等得舒心。  以上就是两大模块的实现代码，在main函数中只需要调用模型寻优也就是train函数，就可以直接用一个module接收最优模型，然后既可以用模型去预测测试集，也可以送到模型评估模块里报告精度和loss。 2 模型在数据集上的效果 本实验将bp神经网络用于一个小实验和两个数据集。小实验为经典的异或分类问题，两个数据集分别为iris鸢尾花分类数据集和MNIST手写数字分类数据集 2.1 异或分类问题 这个小实验的目的是对神经网络模型做一个初步测试，看其能否正常运行输出。  代码：  def Yi\_huo\_train():  # 训练数据：经典的异或分类问题  train\_X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  train\_y = np.array([0, 1, 1, 0])   # 初始化网络，总共2层，输入数据是2维，第一层3个节点，第二层1个节点作为输出层，激活函数使用Relu  model = FullConnectionModel(2, 3, 1)   # 训练网络  for i in range(1):  loss, accuracy = trainOneStep(model, train\_X, train\_y, 1e-2)   # 判断学习是否完成  if i % 200 == 0:  print("loss:", loss, ",accuracy:", accuracy)  if loss < 0.001:  print("训练完成！ 第%d次迭代" % (i))  break   # 测试结果：  model.forward(train\_X, train\_y)  model.Print()  print("y\_label:", train\_y) 2.2 iris鸢尾花分类数据集 数据集来源于sklearn库的datasets中的load\_iris，其标签值为0，1，2，因为神经网络的label值必须是输出节点的one-hot编码的向量形式，所以我们对iris数据集的标签做初步处理：  iris = load\_iris() x = iris.data y = iris.target tmp = [] T = [[1, 0, 0], [0, 1, 0], [0, 0, 1]] for i in y:  tmp.append(T[i]) y = np.array(tmp)  然后分割数据集：  # 将数据集的 70%作为训练集,30%作为测试集,检验模型在测试集上的分类正确率 x\_train, x\_validation, y\_train, y\_validation = train\_test\_split(x, y, test\_size=0.3, random\_state=0)  送入寻优模块，iris数据集不大，不到200个数据，迭代次数可设置为默认的10000，隐藏层维度测试也默认为[100,200,300]。之后评估loss和精度  model = train(x\_train,y\_train,x\_validation,y\_validation) loss, accuracy = evaluate(model, x, y) print(f'Evaluate the best model, test loss={loss:0<10.8}, accuracy={accuracy:0<8.6}.') 2.3 MNIST手写数字数据集 MNIST 数据集是一个非常有名的手写体数字识别数据集，MNIST 数据集是 NIST 数据集的一个子集，它包含了 60000 张图片作为训练数据，10000 张图片作为测试数据，在 MNIST 数据集中的每一张图片都代表了 0~9 中的一个数字，图片的大小都为 28 × 28，且数字都会出现在图片的正中间。  首先进行数据集下载和处理。MNIST 数据集提供了 4 个下载文件，分别为训练数据图片、训练数据答案、测试数据图片、测试数据答案。  下载后采用TensorFlow 提供的一个工具 input\_data 来处理 MNIST 数据集，实现对数据自动识别并读取。  from tensorflow.examples.tutorials.mnist import input\_data  mnist = input\_data.read\_data\_sets("dataset/", one\_hot=True)  实验过程中，发现python自带包tensorflow找不到example模块，在博客帮助下于gitub下载tensorflow文件，其中包含的example文件夹复制粘贴到python的tensorflow安装目录下    但发现example文件夹里又没有tutorials，于是又在百度云上下载tutorials粘贴进example文件夹    解决报错问题，然后将数据送入寻优模块解析  model = train(mnist.train.images, mnist.train.labels, mnist.validation.images, mnist.validation.labels,200)  loss, accuracy = evaluate(model, mnist.test.images, mnist.test.labels) print(f'Evaluate the best model, test loss={loss:0<10.8}, accuracy={accuracy:0<8.6}.') | | | | | | |
| 四、实验结果及分析 1 亦或分类问题结果分析   以上为两次不同运行产生的结果，结果将预测样本值和真实样本标签做了对比，两次结果存在差异，可以看出不同的初始化权重将会得到不同的结果。loss值很大，因为数据集只有4个数据，欠拟合。但该小实验说明模型可以正常输入输出，为后续在大数据集上的实验做了铺垫。 2 Iris数据集结果分析 迭代200次结果：    迭代10000次结果：    迭代200次，loss为0.1086，精度为0.6867，最优隐藏层维度为200；迭代10000次，loss为0.0289，精度为0.98，最优隐藏层维度为200，可以看出迭代次数对模型拟合能力有影响，对模型外部参数影响不大。 3 MNIST手写数字数据集结果分析   迭代次数设置为200，若设置为10000，训练时间过长（超过20分钟），且对精度提升不大，因此设置为200。loss值为0.056，精度0.49，最优隐藏层维度为100.模型拟合度不高，可能原因是对于图像数据的处理，只有一个隐藏层无法很好地对属性值进行变换，可以利用卷积神经网络，增加隐藏层个数来对属性值做更多的升维变换，以此提高模型精度，卷积神经网络这也是图像处理经常使用的学习模型。 对本次实验的思考 神经网络具有很突出的特点，其采用分布式表示法，可自动进行特征提取和选择，不需要人工设计特征，具有良好的适应性和扩展性，可以处理复杂的非线性关系，有利于对复杂的输入输出进行建模，具有强大的拟合能力，可以从大规模数据中学习并发现模式和规律，有效处理高维、大样本的数据。  与传统的线性分类器相比，神经网络可以更好地处理非线性问题，并且具有更高的准确性。神经网络可以自动进行特征提取和选择，无需人工干预，使得建模变得更为简单。但是，神经网络通常需要较大的计算资源和时间进行训练和调优，其计算开销较大。并且它的可解释性较差，人们无法深入理解神经网络中各个隐藏层的具体含义。神经网络的结果与输入数据密切相关，对数据质量的要求较高，而且容易受到噪声的影响。 | | | | | | |