****

人工智能与机器学习

神经网络实验

实 验 报 告

|  |  |
| --- | --- |
| **学 院** | 网络空间安全 |
| **专 业** | 网络空间安全 |
| **班 级** | 22270311 |
| **学 号** | 22200215 |
| **学生姓名** | 王翔宇 |
| **教师姓名** | 唐会军、赵治栋 |
| **完成日期** | 2024.06.18 |
| **成 绩** |  |
| **神经网络编程实验** | | |
| 1. **实验任务** 2. 熟练掌握反向传播算法，画出网络框架图，写出前传、后传步骤； 3. 调整参数使精确率提高； 4. 解决验证集过拟合问题； 5. 改为batch\_size=16的小批量梯度下降； 6. 完成上述改动后，写一个pytorch版本的代码，使用不同的网络结构、不同的优化器、不同的参数对比实验结果，列出实验表格； 7. 结合自己的专业，更换数据集，用神经网络做二分类或多分类应用。 | | |
| 1. **实验分析**   任务1：    任务2，任务3：  调整梯度下降过程中的学习率为0.05，确保算法在迭代过程中更加精细地搜索最优解，避免错过最低点。    增加了L2正则化，防止模型过拟合，提升了模型的泛化能力。    任务4：改为batch\_size=16的小批量梯度下降  采用小批量梯度下降，这种方法结合了梯度下降和随机梯度下降的优点，既能够降低每次迭代的计算开销，又能够利用随机性来增强泛化能力。通常比完全批量（使用全部训练数据）的梯度下降更高效。      任务5：改为pytorch版  import torch import torch.nn as nn import torch.optim as optim from sklearn.datasets import load\_breast\_cancer from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler import pandas as pd *# 加载数据集* data = load\_breast\_cancer() X, y = data.data, data.target *# 标准化特征* scaler = StandardScaler() X\_scaled = scaler.fit\_transform(X) y = y.reshape(-1, 1) *# 划分数据集* X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42) X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train, y\_train, test\_size=0.2, random\_state=42) *# 转换为PyTorch的Tensor* X\_train = torch.tensor(X\_train, dtype=torch.float32) y\_train = torch.tensor(y\_train, dtype=torch.float32) X\_val = torch.tensor(X\_val, dtype=torch.float32) y\_val = torch.tensor(y\_val, dtype=torch.float32) X\_test = torch.tensor(X\_test, dtype=torch.float32) y\_test = torch.tensor(y\_test, dtype=torch.float32) *# 定义神经网络模型* class SimpleNN(nn.Module):  def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):  super(SimpleNN, self).\_\_init\_\_()  self.fc1 = nn.Linear(input\_size, hidden\_size)  self.fc2 = nn.Linear(hidden\_size, output\_size)  self.sigmoid = nn.Sigmoid()   def forward(self, x):  x = self.sigmoid(self.fc1(x))  x = self.sigmoid(self.fc2(x))  return x *# 训练和验证函数* def train\_and\_evaluate(model, optimizer, criterion, num\_epochs=1000, batch\_size=16):  m = X\_train.shape[0]  for epoch in range(num\_epochs):  permutation = torch.randperm(m)  X\_train\_shuffled = X\_train[permutation]  y\_train\_shuffled = y\_train[permutation]   for i in range(0, m, batch\_size):  X\_batch = X\_train\_shuffled[i:i + batch\_size]  y\_batch = y\_train\_shuffled[i:i + batch\_size]   optimizer.zero\_grad()  outputs = model(X\_batch)  loss = criterion(outputs, y\_batch)  loss.backward()  optimizer.step()  if epoch % 100 == 0:  with torch.no\_grad():  val\_outputs = model(X\_val)  val\_loss = criterion(val\_outputs, y\_val)  print(f'第 {epoch}次迭代后的训练集损失：{loss.item()}, 验证集损失: {val\_loss.item()}') *# 测试函数* def test\_model(model):  with torch.no\_grad():  outputs = model(X\_test)  predictions = (outputs > 0.5).float()  accuracy = (predictions == y\_test).float().mean().item() \* 100  return accuracy *# 实验设置* input\_size = X\_train.shape[1] hidden\_size = 45 output\_size = 1 num\_epochs = 1000models\_optimizers = {  'SimpleNN\_SGD': (SimpleNN(input\_size, hidden\_size, output\_size), optim.SGD, 0.05), } criterion = nn.BCELoss() results = {} for key, (model, optimizer\_class, lr) in models\_optimizers.items():  optimizer = optimizer\_class(model.parameters(), lr=lr)  print(f'Training {key}')  train\_and\_evaluate(model, optimizer, criterion, num\_epochs)  accuracy = test\_model(model)  results[key] = accuracy *# 输出实验结果* results\_df = pd.DataFrame(results,index=['Accuracy']).T print(results\_df)  **神经网络结构：**   * 神经网络使用了一个简单的两层全连接层结构（输入层、隐藏层、输出层）。 * 输入层的大小由数据集的特征数量决定，隐藏层的大小是一个自定义的参数 hidden\_size，输出层的大小是1（因为是二分类任务）。 * 使用了 nn.Linear 定义全连接层，nn.Sigmoid 作为激活函数。   **优化器：**   * 选择了 SGD（随机梯度下降）作为优化器，学习率为 0.05。在优化器部分使用了 optim.SGD 初始化优化器，传入了模型的参数和学习率。 * 在训练过程中，每次迭代都会使用这个优化器来更新模型的参数。   **损失函数：**   * 使用了二分类问题常用的二元交叉熵损失函数 nn.BCELoss()。   与源代码实验结果对比如下：  源代码结果：  新修改  修改后代码结果：    可以发现，两者预测的准确率几乎一致。但是源代码在验证集上的损失值小于修改后的代码。  任务6：修改数据集  载入sklearn库中的load\_digits数据集，该数据集本是个多分类数据集，但是，我用此数据集进行手写数字的二分类任务（识别数字5和非数字5）。  对数据标签进行如下操作：    其余部分和上述代码一致，实验结果如下：    可以发现，在该数据集上代码的预测得分也是98%，有一个很不错的效果。 | | |
| 1. **实验过程中遇到的问题和解决方案**   问题：  实验过程中对前向传播、后向传播，以及梯度下降还是不太熟悉。  解决方法：  重新翻看了上课的相关课件进行了学习。  问题：  在写一个pytorch版本的代码任务中，对pytorch包的操作以及网络结构还有优化器的使用不太了解。  解决：  在chatgpt的帮助下，完成了该任务，加深了对pytorch的了解。 | | |