计算机视觉实验一报告
计算机视觉实验一报告

任务要求：

设计一个前馈神经网络，对一组数据实现分类任务。

下载“dataset.csv”数据集，其中包含四类二维高斯数据和它们的标签。设计至少含有一层隐藏层的前馈神经网络来预测二维高斯样本$(data1, data2)$所属的分类$label$。这个数据集需要先进行随机排序，然后选取90%用于训练，剩下的10%用于测试

整体设计

数据预处理

根据要求：我们需要数据经过随机打乱后，按 90% 和 10% 的比例划分为训练集和测试集。

我们使用pandas读取csv文件具体数据

这里用X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1, random_state=42)实现数据打乱特征、标签提取、并划分训练集和测试集，最后需要转换为张量。

最后创建 `TensorDataset` 和 `DataLoader` 用于数据加载

```
train_dataset = TensorDataset(X_train, Y_train)
test_dataset = TensorDataset(X_test, Y_test)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

这里需要注意！！！
pytorch的限制所以我们的label需要取值为0开始的离散值，那么就需要修改Y的值自减一。

模型构建与训练

这里使用了普通的三层，也就是只有一个隐藏层的模型
输入层定义维度为2，因为输入是二维的高斯分布数据。
隐藏层定义需要神经元个数，这是自由的，可以多加尝试。
输出层定义维度为4，因为这个问题是一个四分类问题。
我们需要如下
- 构建前馈神经网络模型，尝试不同的网络层数、以及不同的激活函数（ReLU、Sigmoid、Tanh）。
- 定义损失函数（CrossEntropyLoss）和优化器（AdamW）。

- 在训练集上进行多轮训练，记录每个轮次的损失值。
- 最后使用测试集测试我们的模型结果如何。

有关优化器

这里给出一个其他任务的log如下

```
500_SGD_epoch
Epoch [10/500], Loss: 0.9952073693275452
Epoch [30/500], Loss: 0.9172471165657043
Epoch [50/500], Loss: 0.8393732905387878
Epoch [70/500], Loss: 0.7775803208351135
Epoch [90/500], Loss: 0.7203820943832397
Epoch [110/500], Loss: 0.6697395443916321
Epoch [130/500], Loss: 0.6274657845497131
Epoch [150/500], Loss: 0.5908138751983643
Epoch [170/500], Loss: 0.559145987033844
Epoch [190/500], Loss: 0.5319651961326599
Epoch [210/500], Loss: 0.5083927512168884
Epoch [230/500], Loss: 0.4876689612865448
Epoch [250/500], Loss: 0.46924394369125366
Epoch [270/500], Loss: 0.45263543725013733
Epoch [290/500], Loss: 0.43745434284210205
Epoch [310/500], Loss: 0.42346206307411194
Epoch [330/500], Loss: 0.4104464650154114
Epoch [350/500], Loss: 0.39825204014778137
Epoch [370/500], Loss: 0.3867553770542145
Epoch [390/500], Loss: 0.3758520185947418
Epoch [410/500], Loss: 0.36545705795288086
Epoch [430/500], Loss: 0.3555046319961548
Epoch [450/500], Loss: 0.34595391154289246
Epoch [470/500], Loss: 0.336782306432724
Epoch [490/500], Loss: 0.3279631733894348
Accuracy of the model on the test set: 100.0%

500_AdamW_epoch
Epoch [10/500], Loss: 0.9914829134941101
Epoch [20/500], Loss: 0.7175624966621399
Epoch [30/500], Loss: 0.5416432023048401
Epoch [40/500], Loss: 0.44182807207107544
Epoch [50/500], Loss: 0.3729058504104614
Epoch [60/500], Loss: 0.31072142720222473
Epoch [70/500], Loss: 0.25461503863334656
Epoch [80/500], Loss: 0.20773328840732574
```

```
Epoch [90/500], Loss: 0.17133553326129913
Epoch [100/500], Loss: 0.14437882602214813
Epoch [110/500], Loss: 0.12482898682355881
Epoch [120/500], Loss: 0.11068034917116165
Epoch [130/500], Loss: 0.10029398649930954
Epoch [140/500], Loss: 0.09250326454639435
Epoch [150/500], Loss: 0.08650895953178406
Epoch [160/500], Loss: 0.08178768306970596
Epoch [170/500], Loss: 0.0780133455991745
Epoch [190/500], Loss: 0.07238556444644928
...
Epoch [460/500], Loss: 0.054268285632133484
Epoch [470/500], Loss: 0.05406449735164642
Epoch [480/500], Loss: 0.053868409246206284
Epoch [490/500], Loss: 0.05367942154407501
Epoch [500/500], Loss: 0.05349693074822426
Accuracy of the model on the test set: 100.0%
```

我们可以很明显的发现SGD和AdamW在训练上面的区别，由于AdamW是考虑了物理惯性等优化结构后的成果，它的收敛速度比SGD快的多，当然效果也有可能是因为学习率的影响。但是总的来说，AdamW和Adam是改进后的优化器，可以使用Adam而不是SGD可以得到更好的效果。

有关激活函数

```
50_relu_epoch
Epoch [1/50], Loss: 0.7081524133682251
Epoch [5/50], Loss: 0.04637599736452103
Epoch [9/50], Loss: 0.018905771896243095
Epoch [13/50], Loss: 0.005938094574958086
Epoch [17/50], Loss: 0.051892317831516266
Epoch [21/50], Loss: 0.007578755728900433
Epoch [25/50], Loss: 0.0016273203073069453
Epoch [29/50], Loss: 0.0005599295836873353
Epoch [33/50], Loss: 0.016461113467812538
Epoch [37/50], Loss: 0.02073649875819683
Epoch [41/50], Loss: 0.03722808137536049
Epoch [45/50], Loss: 0.19947202503681183
Epoch [49/50], Loss: 0.0007041222997941077
Accuracy on the test set: 99.5%

50_sigmoid_epoch
Epoch [1/50], Loss: 1.1432151794433594
Epoch [5/50], Loss: 0.424778550863266
```

```
Epoch [9/50], Loss: 0.15931442379951477
Epoch [13/50], Loss: 0.07751275599002838
Epoch [17/50], Loss: 0.0729590654373169
Epoch [21/50], Loss: 0.026640450581908226
Epoch [25/50], Loss: 0.027256734669208527
Epoch [29/50], Loss: 0.06364866346120834
Epoch [33/50], Loss: 0.01086816843599081
Epoch [37/50], Loss: 0.014782466925680637
Epoch [41/50], Loss: 0.013351364061236382
Epoch [45/50], Loss: 0.028768200427293777
Epoch [49/50], Loss: 0.040534261614084244
Accuracy on the test set: 98.5%
```

可以发现二者有一些区别，但是本身这个问题比较简单，体现不出来区别，可能需要具体的输出才能比较二者区别。不过还是可以看出来如果用sigmoid整体loss还是会更高一些。

总的来说，还是以下的比较：

1. Sigmoid ：
- 输出值在 0 到 1 之间，具有平滑的曲线。
- 在输入值较大或较小时，梯度趋近于 0，容易导致梯度消失问题，影响训练效率。

2. ReLU ：
- 计算简单，梯度在正数部分恒为 1，有效缓解了梯度消失问题。
- 缺点是当输入为负数时，输出为 0，可能导致一些神经元永远不被激活。

3. Tanh ：
- 输出值在 -1 到 1 之间，是 Sigmoid 函数的一种改进，中心在 0 点。
- 仍然存在梯度消失问题，但程度比 Sigmoid 函数轻。

4. Leaky ReLU ：
- 对 ReLU 函数的改进，当输入为负数时，不是输出 0，而是有一个小的斜率（如 0.01）。
- 有助于缓解 ReLU 函数中神经元死亡的问题。

总结

实验一只是一个简单的引入，能够初步了解一下前馈神经网络，或者应该叫mlp比较合适。
经过没有特别设计的网络实现的效果还是可以的，这个问题本身比较简单。