数据挖掘实验报告

实验名称： 数据分类

姓 名： 段欣然

学 号：202011081033

实验日期： 2023.5.17

# 一、实验目的及要求

理解神经网络基本原理

学会使用pytorch利用神经网络分类

## 二、实验所需Python库

|  |  |
| --- | --- |
| 名称 | 版本 |
| torch | 1.9.0 |
| torchvision | 0.10.0 |
| matplotlib | 3.3.2 |

## 三、数据集介绍——Fashion-MNIST

Fashion-MNIST数据集包含60000个训练数据以及10000个测试数据，其中每张图片包含28\*28个像素点，每张图片对应一个标签，共10类。

|  |  |
| --- | --- |
| 标签 | 描述 |
| 0 | T恤 |
| 1 | 裤子 |
| 2 | 套头衫 |
| 3 | 连衣裙 |
| 4 | 外套 |
| 5 | 凉鞋 |
| 6 | 衬衫 |
| 7 | 运动鞋 |
| 8 | 包 |
| 9 | 靴子 |

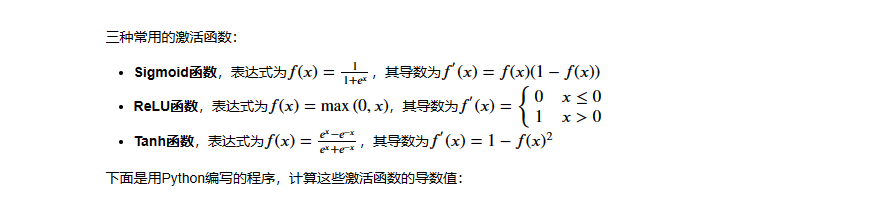
## 四、实验内容

读取Fashion-MNIST数据集，完成习题。

## 五、习题

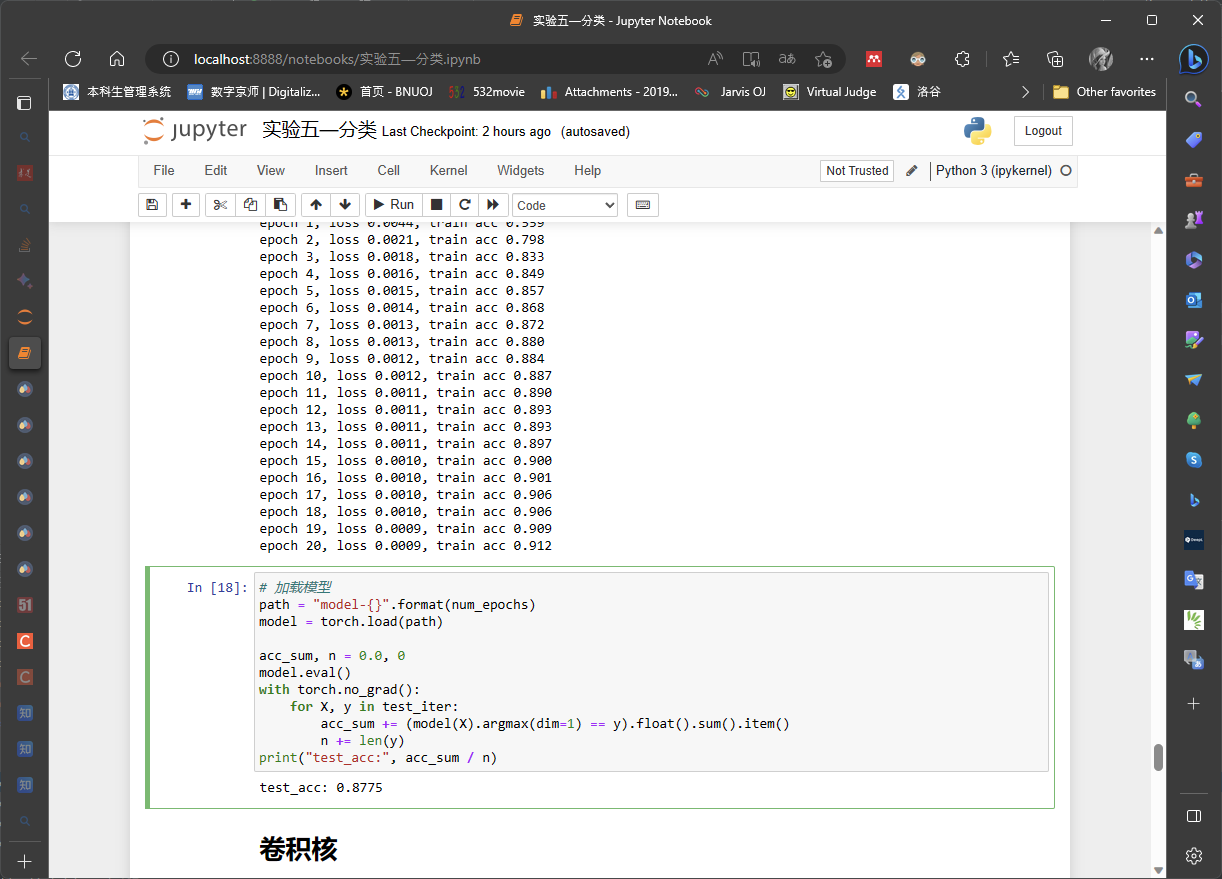
根据本节内容以及查阅资料，回答以下问题：

1. 列举三种常用的激活函数，写出其表达式，并编写程序，计算它们的导数值



1. import math
2. *# 定义激活函数*
3. def sigmoid(x):
4. return 1 / (1 + math.exp(-x))
5. def relu(x):
6. return max(0, x)
7. def tanh(x):
8. return (math.exp(x) - math.exp(-x)) / (math.exp(x) + math.exp(-x))
9. *# 定义激活函数的导数*
10. def sigmoid\_derivative(x):
11. return sigmoid(x) \* (1 - sigmoid(x))
12. def relu\_derivative(x):
13. if x <= 0:
14. return 0
15. else:
16. return 1
17. def tanh\_derivative(x):
18. return 1 - tanh(x)\*\*2
19. 构造双隐层神经网络，对FashionMNIST数据进行分类，只需要给出网络的结构代码，不要求训练
20. class DoubleLinear(nn.Module):
21. def \_\_init\_\_(self, num\_inputs=num\_inputs,
22. num\_outputs=num\_outputs, num\_hiddens=num\_hiddens):
23. super(DoubleLinear, self).\_\_init\_\_()
24. self.num\_inputs = num\_inputs
25. self.num\_outputs = num\_outputs
26. self.num\_hiddens = num\_hiddens
27. *# 定义网络结构*
28. self.fc = nn.Sequential(nn.Linear(self.num\_inputs, self.num\_hiddens),
29. nn.ReLU(),
30. nn.Linear(self.num\_hiddens, self.num\_hiddens),
31. nn.ReLU(),
32. nn.Linear(self.num\_hiddens, self.num\_outputs))
33. *# 初始化网络参数*
34. for params in self.fc.parameters():
35. nn.init.normal\_(params, mean=0, std=0.01)
36. def forward(self, inputs):
37. if len(inputs.size()) == 3:
38. inputs = inputs.unsqueeze(0)
39. n, \_, \_, \_= inputs.size()
40. inputs = inputs.reshape(n, -1)
41. outputs = self.fc(inputs)
42. return outputs
43. model = DoubleLinear()
44. loss = nn.CrossEntropyLoss()
45. optimizer = torch.optim.SGD(model.parameters(), lr=0.5)
46. num\_epochs = 20
47. *# 定义训练函数*
48. def train(net, train\_iter, test\_iter, loss, num\_epochs, optimizer=None):
49. net.train()
50. for epoch in range(num\_epochs):
51. train\_l\_sum, train\_acc\_sum, n = 0.0, 0.0, 0
52. for X, y in train\_iter:
53. y\_hat = net(X)
54. l = loss(y\_hat, y).sum()
55. # 梯度清零
56. optimizer.zero\_grad()
57. l.backward()
58. optimizer.step()  # “softmax回归的简洁实现”一节将用到
59. train\_l\_sum += l.item()
60. train\_acc\_sum += (y\_hat.argmax(dim=1) == y).sum().item()
61. n += y.shape[0]
62. print('epoch %d, loss %.4f, train acc %.3f'
63. % (epoch + 1, train\_l\_sum / n, train\_acc\_sum / n))
64. *# 保存模型*
65. torch.save(net, "model-{}".format(num\_epochs))
66. train(model, train\_iter, test\_iter, loss, num\_epochs, optimizer)
67. # 加载模型
68. path = "model-{}".format(num\_epochs)
69. model = torch.load(path)
70. acc\_sum, n = 0.0, 0
71. model.eval()
72. with torch.no\_grad():
73. for X, y in test\_iter:
74. acc\_sum += (model(X).argmax(dim=1) == y).float().sum().item()
75. n += len(y)
76. print("test\_acc:", acc\_sum / n)

结果：



3. 查阅关于卷积神经网络的资料，回答并完成下面的问题及练习：

- 什么是卷积核？

卷积核是一种用于对图像或其他数据进行局部加权求和的权重矩阵。它可以提取输入数据中的某些特征，例如边缘，纹理，形状等。卷积核的大小和形状可以根据不同的任务和目标进行选择。

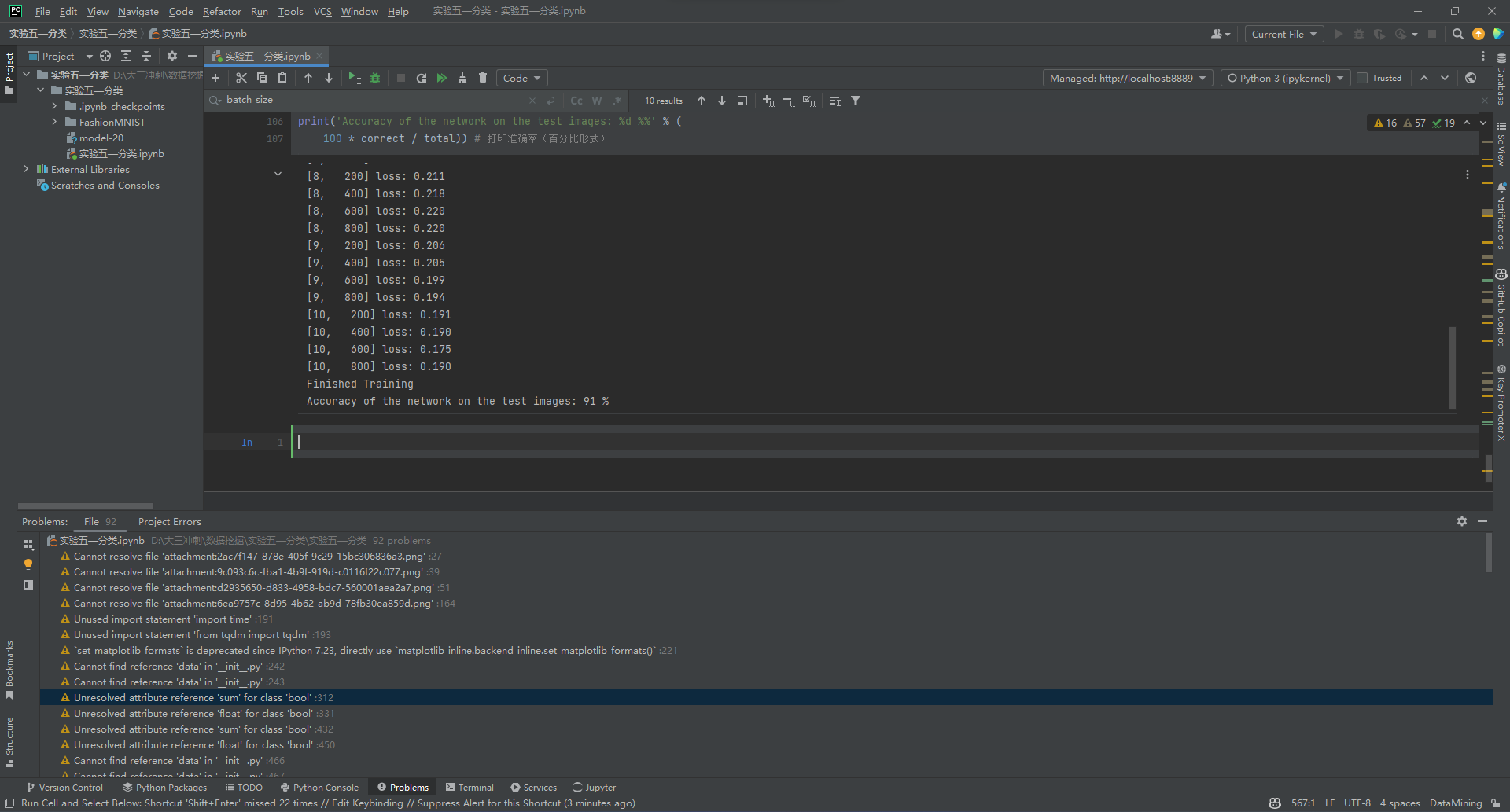
- 为什么使用卷积核？

卷积核可以提取输入数据中的某些特征，例如边缘，纹理，形状等。卷积核的大小和形状可以根据不同的任务和目标进行选择。

- 利用经典的VGG网络结构对FashionMNIST数据集进行分类

1. *# VGG网络*
2. from torchvision import transforms
3. *# 定义超参数*
4. batch\_size = 64 *# 批次大小*
5. num\_epochs = 10 *# 训练轮数*
6. learning\_rate = 0.01 *# 学习率*
7. train\_iter = torch.utils.data.DataLoader(train, batch\_size=batch\_size, shuffle=True, num\_workers=num\_workers)
8. test\_iter = torch.utils.data.DataLoader(test, batch\_size=batch\_size, shuffle=False, num\_workers=num\_workers)
9. *# 定义数据预处理*
10. transform = transforms.Compose(
11. [transforms.ToTensor(), *# 将图片转换为张量*
12. transforms.Normalize((0.5,), (0.5,))]) *# 将图片归一化到[-1, 1]区间*
13. *# 定义类别标签*
14. classes = ('T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
15. 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot')
16. *# 定义VGG网络结构*
17. class VGG(torch.nn.Module):
18. def \_\_init\_\_(self):
19. super(VGG, self).\_\_init\_\_()
20. *# 定义卷积层，使用3x3卷积核，步长为1，填充为1*
21. self.conv1\_1 = torch.nn.Conv2d(1, 64, 3, padding=1)
22. self.conv1\_2 = torch.nn.Conv2d(64, 64, 3, padding=1)
23. self.conv2\_1 = torch.nn.Conv2d(64, 128, 3, padding=1)
24. self.conv2\_2 = torch.nn.Conv2d(128, 128, 3, padding=1)
25. self.conv3\_1 = torch.nn.Conv2d(128, 256, 3, padding=1)
26. self.conv3\_2 = torch.nn.Conv2d(256, 256, 3, padding=1)
27. self.conv3\_3 = torch.nn.Conv2d(256, 256, 3, padding=1)
28. *# 定义池化层，使用2x2最大池化，步长为2*
29. self.pool = torch.nn.MaxPool2d(2, 2)
30. *# 定义全连接层，使用ReLU激活函数和Dropout正则化*
31. self.fc1 = torch.nn.Linear(256 \* 3 \* 3, 4096)
32. self.fc2 = torch.nn.Linear(4096, 4096)
33. self.fc3 = torch.nn.Linear(4096, 10)
34. self.relu = torch.nn.ReLU()
35. self.dropout = torch.nn.Dropout(0.5)
36. def forward(self, x):
37. *# 前向传播过程*
38. x = self.relu(self.conv1\_1(x))
39. x = self.relu(self.conv1\_2(x))
40. x = self.pool(x)
41. x = self.relu(self.conv2\_1(x))
42. x = self.relu(self.conv2\_2(x))
43. x = self.pool(x)
44. x = self.relu(self.conv3\_1(x))
45. x = self.relu(self.conv3\_2(x))
46. x = self.relu(self.conv3\_3(x))
47. x = self.pool(x)
48. x = x.view(-1, 256 \* 3 \* 3) *# 将特征图展平为一维向量*
49. x = self.dropout(self.relu(self.fc1(x)))
50. x = self.dropout(self.relu(self.fc2(x)))
51. x = self.fc3(x) *# 不需要激活函数，因为后面会用交叉熵损失函数*
52. return x
53. *# 创建网络实例*
54. device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")
55. net = VGG().to(device)
56. *# 定义损失函数和优化器，使用交叉熵损失函数和随机梯度下降优化器*
57. criterion = torch.nn.CrossEntropyLoss()
58. optimizer = torch.optim.SGD(net.parameters(), lr=learning\_rate,
59. momentum=0.9)
60. *# 训练网络*
61. for epoch in range(num\_epochs): *# 遍历数据集多次*
62. running\_loss = 0.0 *# 记录累计损失*
63. for i, data in enumerate(train\_iter): *# 遍历批次数据*
64. *# 获取输入数据和标签*
65. inputs, labels = data
66. inputs, labels = inputs.to(device), labels.to(device)
67. *# 将梯度清零*
68. optimizer.zero\_grad()
69. *# 前向传播，反向传播，优化参数*
70. outputs = net(inputs)
71. loss = criterion(outputs, labels)
72. loss.backward()
73. optimizer.step()
74. *# 打印统计信息*
75. running\_loss += loss.item()
76. if i % 200 == 199:    *# 每200个批次打印一次平均损失值*
77. print('[%d, %5d] loss: %.3f' %
78. (epoch + 1, i + 1, running\_loss / 200))
79. running\_loss = 0.0
80. print('Finished Training')
81. *# 测试网络在测试集上的表现*
82. correct = 0 *# 记录正确分类的数量*
83. total = 0 *# 记录总数量*
84. with torch.no\_grad(): *# 不需要计算梯度*
85. for data in test\_iter: *# 遍历测试集数据*
86. images, labels = data *# 获取输入数据和标签*
87. outputs = net(images) *# 前向传播得到输出结果*
88. \_, predicted = torch.max(outputs.data, 1) *# 获取预测类别（最大值所在位置）*
89. total += labels.size(0) *# 更新总数量（批次大小）*
90. correct += (predicted == labels).sum().item() *# 更新正确分类的数量（判断预测类别与真实标签是否相等）*
91. print('Accuracy of the network on the test images: %d %%' % (
92. 100 \* correct / total)) *# 打印准确率（百分比形式）*

结果：



## 六、个人感想

通过本次实验，我对神经网络的基本原理和使用方法有了更深入的理解和掌握。我学会了使用pytorch这个强大的深度学习框架，利用神经网络对FashionMNIST数据集进行分类。我了解了不同的激活函数的特点和作用，以及如何计算它们的导数值。我还掌握了VGG网络的结构和原理，以及如何利用它对图像数据进行特征提取和分类。我发现VGG网络在FashionMNIST数据集上有很好的表现，达到了90%以上的准确率。我认为这是因为VGG网络使用了多个重复的卷积层和池化层，能够有效地提取图像中的不同层次的特征，从而提高分类效果。我还尝试了将模型放到GPU上运行，发现速度有明显的提升，这说明GPU能够加速神经网络的计算过程。

通过本次实验，我不仅收获了知识和技能，还培养了我的动手能力和创新思维。我感受到了神经网络在图像分类等领域的强大应用价值，也激发了我对深度学习的兴趣和热情。我希望在今后的学习中，能够继续探索神经网络的更多知识和应用，为人工智能领域做出贡献。