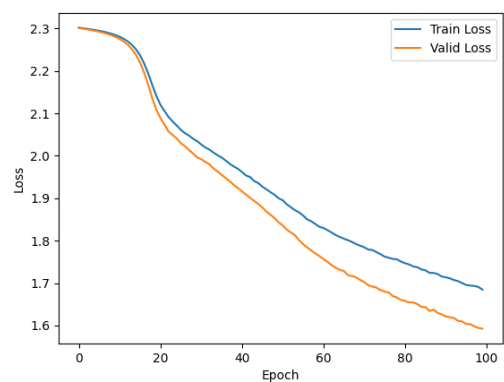
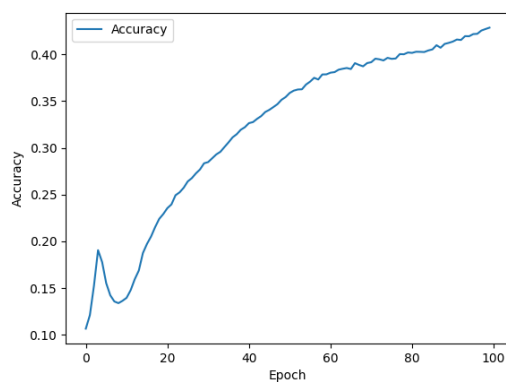


视听信息系统导论编程 1

高艺轩 毕嘉仪

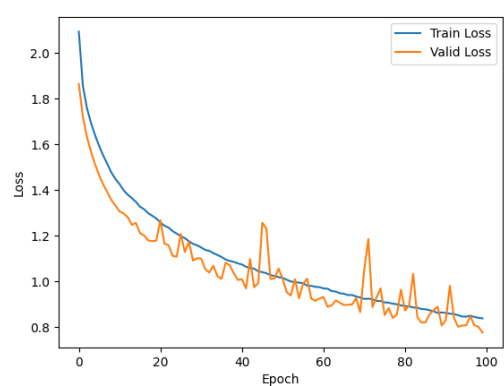
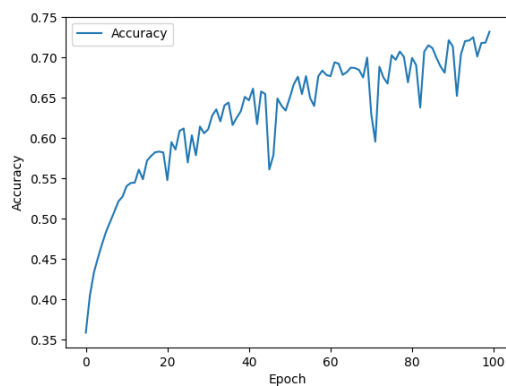
第 1 题

在 SGD 优化器下，输出的结果如下：



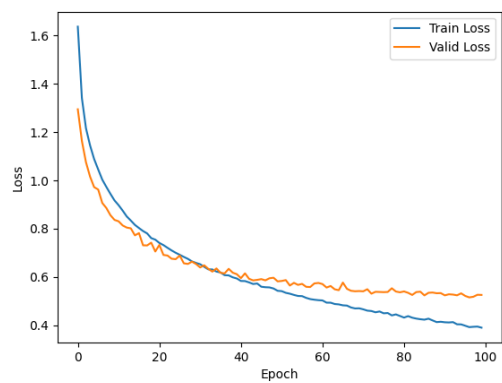
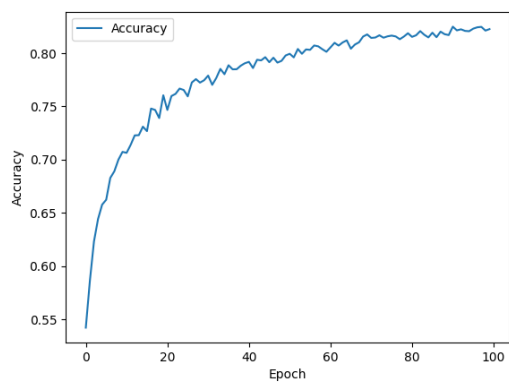
第 2 题

在 SGD 优化器下，使用 ReLU 激活函数与 Batch Normalization，输出的结果如下：



第 3 题

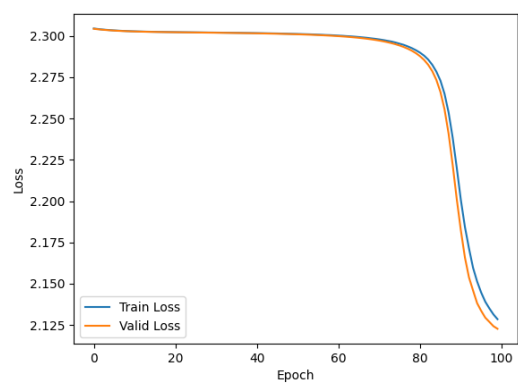
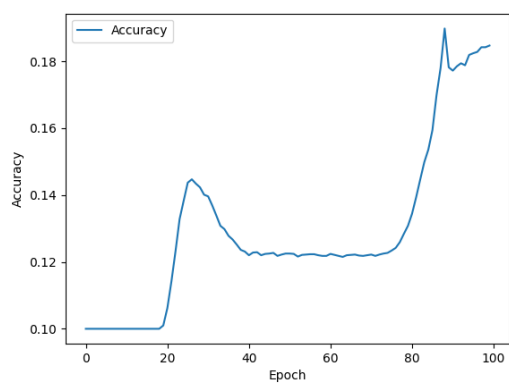
在 AdamW 优化器下，使用 ReLU 激活函数与 Batch Normalization，输出的结果如下：



第 4 题

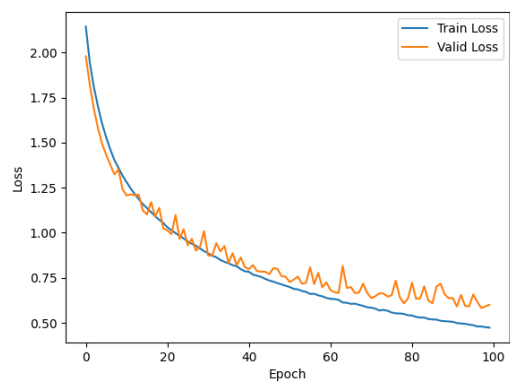
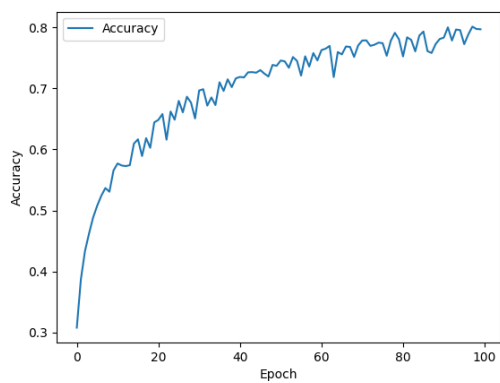
4.1

仍旧使用 \tanh 作为激活函数，SGD 优化器：



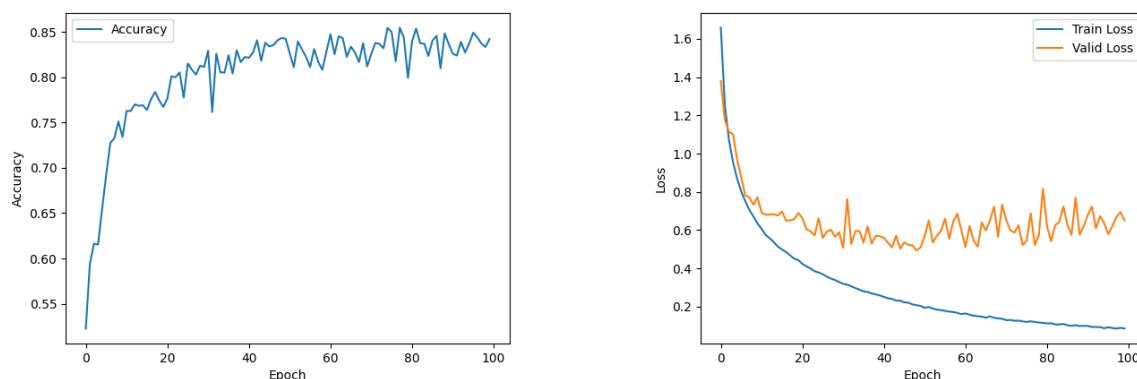
4.2

将激活函数更改为 ReLU 函数，并使用 BatchNorm，使用 SGD：



4.3

在 AdamW 优化器下，使用 ReLU 激活函数与 Batch Normalization，输出的结果如下：



第 5 题

1. 请根据结果分析 ReLU 和 tanh 激活函数的表现.

由于实验 1 与实验 2 同时更改了激活函数与是否增加 Batch Normalization，我们无法直接从实验结果中通过对比得出 ReLU 与 tanh 到底哪个表现更好。不过从理论上进行分析，ReLU 激活函数在当前任务中的表现应当会优于 tanh 激活函数。这是由于 ReLU 激活函数不存在梯度消失问题，能够使网络快速收敛，无论网络深浅；而 tanh 函数容易陷入饱和区、导致梯度消失，使得网络收敛速度大幅降低，特别是在深层次网络中这一现象更加明显；因此，在 CNN 架构中，ReLU 激活函数表现应该会优于 tanh。

2. 请根据结果分析 BatchNorm 的作用.

由于实验 1 与实验 2 同时更改了激活函数与是否增加 Batch Normalization，我们无法直接从实验结果中通过对比看出 BatchNorm 的作用。不过从理论上进行分析，BatchNorm 可以使网络中每一层的数据分布均值为 0、方差为 1，这样神经元输出值不会太大，加强了网络稳定性，同时防止出现梯度爆炸（或梯度消失），使网络能够较快收敛；此外，Batch Normalization 可以使相同 batch 中的所有样本相互关联，加强了泛化性，避免过拟合。

3. 请根据结果分析更换优化器的效果.

从实验 2、3 的结果对比中可以看出，使用 SGD 优化器会导致测试准确率与训练、测试损失出现明显震荡，而更换了 AdamW 优化器以后准确率曲线与损失曲线都更加平滑。SGD 优化器容易出现振荡是由于它会导致梯度值相差较大的方向上优化步长差异也大，致使梯度值大的方向上产生振荡；而 Adam 优化器可以根据不同参数梯度值自适应地调整各个方向的优化步长大小，以使各个方向都能同时得到有效且合理的优化。

4. 请根据你的结果分析模型是否出现了过拟合，如有，请在图像中指出在哪里出现了过拟合。如无，请给出你判断的原因.

通过训练/测试损失曲线以及测试准确率曲线可知，实验 4 的第 3 部分出现了过拟合现象，发生在 epoch 35 附近。判断依据：训练损失持续降低，而测试损失不再降低甚至开始升高且出现巨幅振荡，测试准确率也不再增加。

程序源代码

```
1 import os
2 import torch
3 import torchvision
4 import numpy as np
5 from tqdm import tqdm
6 import torch.nn as nn
7 import torch.nn.functional as F
8 import torch.optim as optim
9 import torchvision.transforms as transforms
10 import matplotlib.pyplot as plt
11
12 from PIL import Image
13 from typing import *
14
15 device = 'cuda' if torch.cuda.is_available() else 'cpu'
16
17 ##### 数据集初始化与读入 #####
18 train_transform = transforms.Compose([
19     transforms.RandomHorizontalFlip(),
20     transforms.RandomCrop(32, padding=4),
21     transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
22     transforms.ToTensor()
23 ])
24
25 train_dset = torchvision.datasets.CIFAR10(root='./CIFAR10', train=True, download=
    False, transform=train_transform)
26 test_dset = torchvision.datasets.CIFAR10(root='./CIFAR10', train=False, download=
    False, transform=transforms.ToTensor())
27 train_loader = torch.utils.data.DataLoader(train_dset, batch_size=128, shuffle=
    True, num_workers=0)
28 test_loader = torch.utils.data.DataLoader(test_dset, batch_size=128, shuffle=
    False, num_workers=0)
29 #####
30
31
32 ##### 构建模型 #####
```

```

33 class Net(nn.Module):
34     def __init__(self,act):
35         super(Net, self).__init__()
36         # 卷积层 (32x32x3的图像)
37         self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
38         # 卷积层(16x16x16)
39         self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
40         # 卷积层(8x8x32)
41         self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
42         # 最大池化层
43         self.pool = nn.MaxPool2d(2, 2)
44         # linear layer (64 * 4 * 4 -> 500)
45         self.fc1 = nn.Linear(64 * 4 * 4, 500)
46         # linear layer (500 -> 10)
47         self.fc2 = nn.Linear(500, 10)
48         if act == 'relu':
49             self.act = F.relu
50         elif act == 'tanh':
51             self.act = F.tanh
52         elif act == 'sigmoid':
53             self.act = F.sigmoid
54
55     def forward(self, x):
56         # add sequence of convolutional and max pooling layers
57         x = self.pool(self.act(self.conv1(x)))
58         x = self.pool(self.act(self.conv2(x)))
59         x = self.pool(self.act(self.conv3(x)))
60         # flatten image input
61         x = x.view(-1, 64 * 4 * 4)
62
63         x = self.act(self.fc1(x))
64
65         x = self.fc2(x)
66         return x
67 #####
68
69 ##### 模型加入batchnorm #####
70 class BnNet(nn.Module):
71     def __init__(self, act):
72         super(BnNet, self).__init__()
73         # 卷积层 (32x32x3的图像)
74         self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
75         self.bn1 = nn.BatchNorm2d(16)

```

```
76
77     # 卷积层(16x16x16)
78     self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
79     self.bn2 = nn.BatchNorm2d(32)
80
81     # 卷积层(8x8x32)
82     self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
83     self.bn3 = nn.BatchNorm2d(64)
84
85     # 最大池化层
86     self.pool = nn.MaxPool2d(2, 2)
87
88     # linear layer (64 * 4 * 4 -> 500)
89     self.fc1 = nn.Linear(64 * 4 * 4, 500)
90     self.bn4 = nn.BatchNorm1d(500)
91
92     # linear layer (500 -> 10)
93     self.fc2 = nn.Linear(500, 10)
94
95     if act == 'relu':
96         self.act = F.relu
97     elif act == 'tanh':
98         self.act = F.tanh
99     elif act == 'sigmoid':
100         self.act = F.sigmoid
101
102     def forward(self, x):
103         # add sequence of convolutional and max pooling layers
104         x = self.pool(self.act(self.bn1(self.conv1(x))))
105         x = self.pool(self.act(self.bn2(self.conv2(x))))
106         x = self.pool(self.act(self.bn3(self.conv3(x))))
107
108         # flatten image input
109         x = x.view(-1, 64 * 4 * 4)
110
111         x = self.act(self.bn4(self.fc1(x)))
112         x = self.fc2(x)
113         return x
114     ##### 构建模型 #####
115
116
117
118     class DeepNet(nn.Module):
```

```
119 def __init__(self):
120     super(DeepNet, self).__init__()
121     ##### 代码填空: 请在此填补模型定义代码 #####
122     self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
123     self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
124     self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
125     self.conv4 = nn.Conv2d(64, 128, 3, padding=1)
126     self.conv5 = nn.Conv2d(128, 256, 3, padding=1)
127     self.conv6 = nn.Conv2d(256, 512, 1, padding=0)
128     self.fc1 = nn.Linear(512, 256)
129     self.fc2 = nn.Linear(256, 128)
130     self.fc3 = nn.Linear(128, 10)
131     self.act = F.tanh
132     self.pool = nn.MaxPool2d(2, 2)
133     self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
134     #####
135
136 def forward(self, x):
137     # convolutional layers
138     ##### 代码填空: 请在此填补前向计算代码 #####
139     x = self.act(self.conv1(x))
140     x = self.pool(self.act(self.conv2(x)))
141     x = self.act(self.conv3(x))
142     x = self.pool(self.act(self.conv4(x)))
143     x = self.act(self.conv5(x))
144     x = self.avgpool(self.act(self.conv6(x)))
145     x = x.view(-1, 512)
146
147     x = self.act(self.fc1(x))
148     x = self.act(self.fc2(x))
149     x = self.act(self.fc3(x))
150     #####
151     return x
152
153 class BnDeepNet(nn.Module):
154     def __init__(self, act):
155         super(BnDeepNet, self).__init__()
156         ##### 代码填空: 请在此填补模型定义代码 #####
157         self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
158         self.bn1 = nn.BatchNorm2d(16)
159         self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
160         self.bn2 = nn.BatchNorm2d(32)
161         self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
```

```

162     self.bn3 = nn.BatchNorm2d(64)
163     self.conv4 = nn.Conv2d(64, 128, 3, padding=1)
164     self.bn4 = nn.BatchNorm2d(128)
165     self.conv5 = nn.Conv2d(128, 256, 3, padding=1)
166     self.bn5 = nn.BatchNorm2d(256)
167     self.conv6 = nn.Conv2d(256, 512, 1, padding=0)
168     self.bn6 = nn.BatchNorm2d(512)
169     self.fc1 = nn.Linear(512, 256)
170     self.bnfc1 = nn.BatchNorm1d(256)
171     self.fc2 = nn.Linear(256, 128)
172     self.bnfc2 = nn.BatchNorm1d(128)
173     self.fc3 = nn.Linear(128, 10)
174     if act == 'relu':
175         self.act = F.relu
176     elif act == 'tanh':
177         self.act = F.tanh
178     elif act == 'sigmoid':
179         self.act = F.sigmoid
180     self.pool = nn.MaxPool2d(2, 2)
181     self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
182     #####
183
184     def forward(self, x):
185         # convolutional layers
186         ##### 代码填空: 请在此填补前向计算代码 #####
187         x = self.act(self.bn1(self.conv1(x)))
188         x = self.pool(self.act(self.bn2(self.conv2(x))))
189         x = self.act(self.bn3(self.conv3(x)))
190         x = self.pool(self.act(self.bn4(self.conv4(x))))
191         x = self.act(self.bn5(self.conv5(x)))
192         x = self.avgpool(self.act(self.bn6(self.conv6(x))))
193         x = x.view(-1, 512)
194
195         x = self.act(self.bnfc1(self.fc1(x)))
196         x = self.act(self.bnfc2(self.fc2(x)))
197         x = self.act(self.fc3(x))
198         return x
199         #####
200
201
202     ##### 训练前准备 #####
203
204

```



```

205 model = BnDeepNet('relu').to(device)
206 criterion = nn.CrossEntropyLoss().to(device)
207
208 optimizer_type = "AdamW" #或者换成AdamW
209 if optimizer_type == "SGD":
210     optimizer = optim.SGD(model.parameters(), lr=0.001)
211 elif optimizer_type == "AdamW":
212     ##### 代码填空: 请在此填补Adam优化器计算代码, lr=0.0001 #####
213     optimizer = optim.AdamW(model.parameters(), lr=0.0001)
214     #####
215     pass
216
217 n_epochs = 100
218 train_losses = []
219 valid_losses = []
220 accuracies = []
221 #####
222
223
224 ##### 训练+验证 #####
225 for epoch in range(n_epochs):
226     train_loss = 0.0
227     valid_loss = 0.0
228     model.train()
229     for idx, (img, label) in tqdm(enumerate(train_loader)):
230         img, label = img.to(device), label.to(device)
231         optimizer.zero_grad()
232         output = model(img)
233         loss = criterion(output, label)
234         loss.backward()
235         optimizer.step()
236         train_loss += loss.item() * img.shape[0]
237
238     model.eval()
239     correct = 0
240     total = 0
241     for idx, (img, label) in tqdm(enumerate(test_loader)):
242         img, label = img.to(device), label.to(device)
243         output = model(img)
244         loss = criterion(output, label)
245         valid_loss += loss.item() * img.shape[0]
246         _, predicted = torch.max(output.data, 1)
247         total += label.size(0)

```

```
248     correct += (predicted == label).sum().item()
249
250     train_loss = train_loss / len(train_dset)
251     valid_loss = valid_loss / len(test_dset)
252
253     train_losses.append(train_loss)
254     valid_losses.append(valid_loss)
255     accuracy = correct / total
256     accuracies.append(accuracy)
257
258     print(f"Epoch:{epoch}, Acc:{correct/total}, Train Loss:{train_loss}, Test Loss
          :{valid_loss}")
259 #####
260
261 ##### 曲线绘制 #####
262 print("MAX ACC: ", np.max(accuracies))
263 plt.plot(range(n_epochs), train_losses, label='Train Loss')
264 plt.plot(range(n_epochs), valid_losses, label='Valid Loss')
265 plt.xlabel('Epoch')
266 plt.ylabel('Loss')
267 plt.legend()
268 plt.savefig("Loss.png")
269 plt.clf()
270 # 绘制验证集准确率随epoch的变化曲线
271 plt.plot(range(n_epochs), accuracies, label='Accuracy')
272 plt.xlabel('Epoch')
273 plt.ylabel('Accuracy')
274 plt.legend()
275 plt.savefig("Acc.png")
276 #####
```