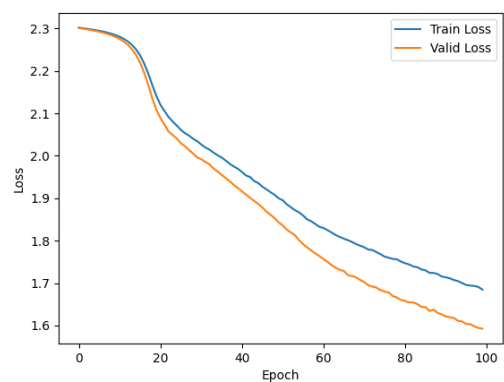
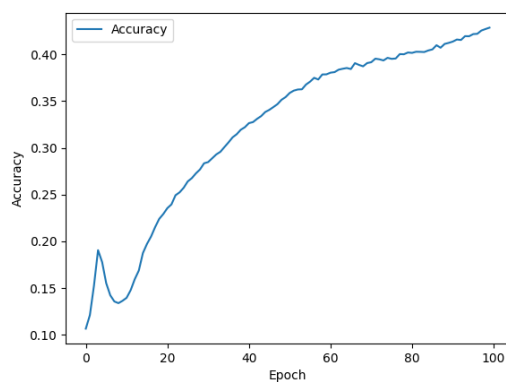


视听信息系统导论编程 1

高艺轩 毕嘉仪

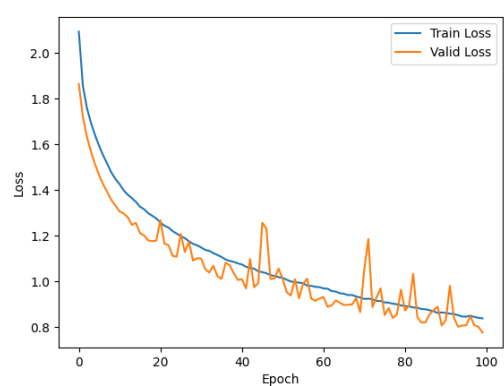
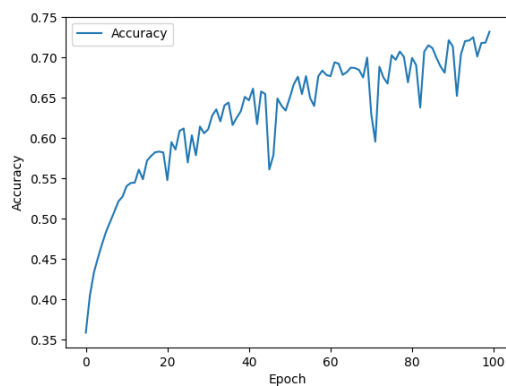
第 1 题

在 SGD 优化器下，输出的结果如下：



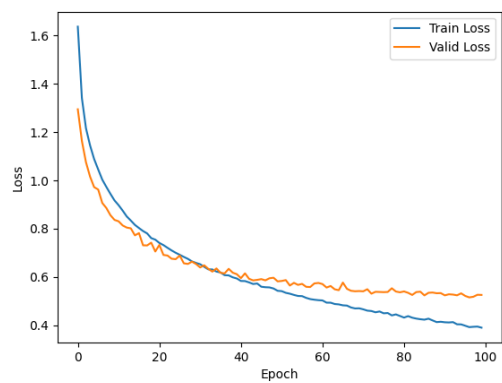
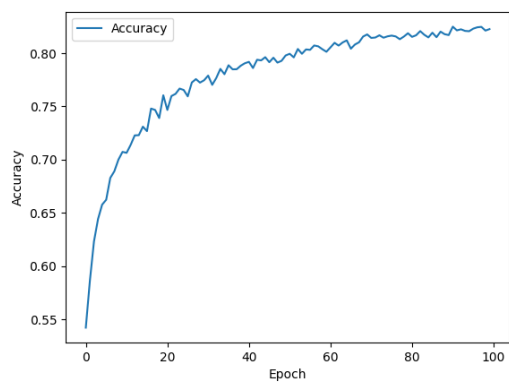
第 2 题

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

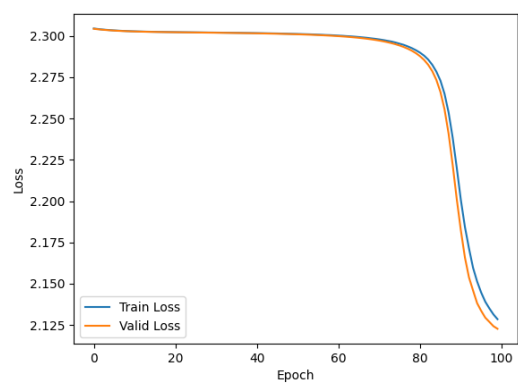
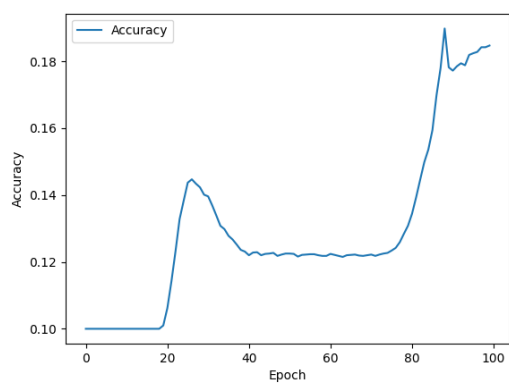
在 AdamW 优化器下，使用 ReLU 激活函数与 Batch Normalization，输出的结果如下：



第 4 题

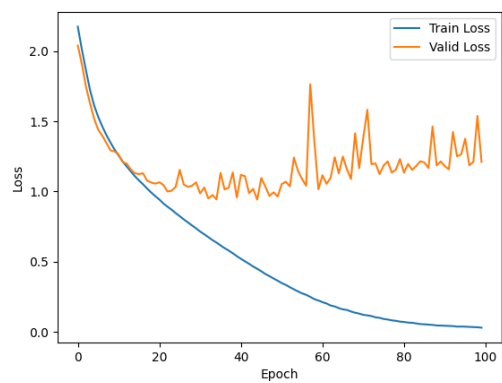
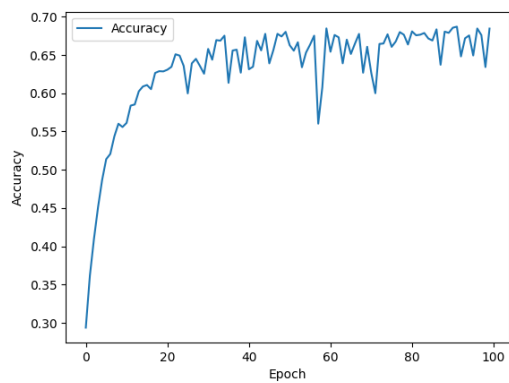
4.1

仍旧使用 \tanh 作为激活函数，SGD 优化器：



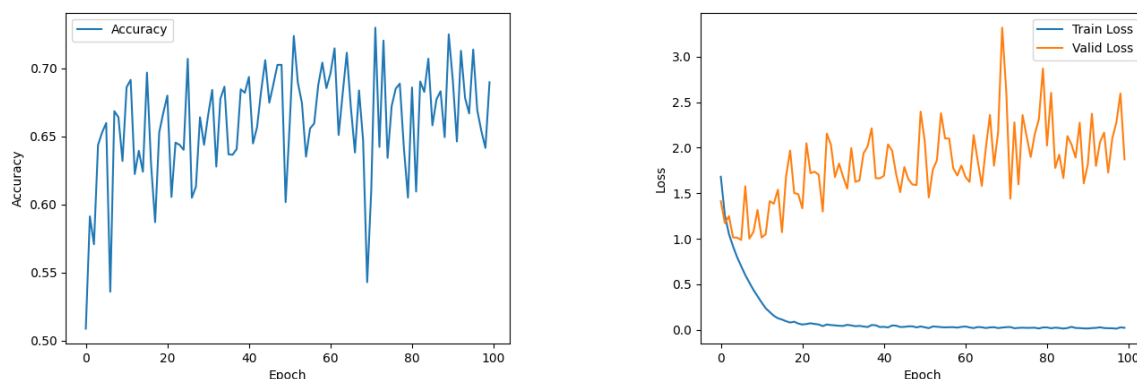
4.2

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 2, 3 部分均出现了过拟合现象，分别发生在 epoch 35 附近以及 epoch 5 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 class CustomCIFAR10(torchvision.datasets.CIFAR10):
25     def __init__(
26         self,
27         root,
28         train,
29         transform = None,
30         target_transform = None,
31         download = False,
32     ):
33         super().__init__(root, train, transform, target_transform, download)
34         new_data = []
35         new_target = []
36         for i in tqdm(range(len(self.data))):
```

```

37         img, target = self.data[i], self.targets[i]
38         # print(self.data.shape)
39         # print(img.shape)
40         img = Image.fromarray(img)
41
42         if self.transform is not None:
43             img = self.transform(img)
44
45         if self.target_transform is not None:
46             target = self.target_transform(target)
47
48         new_data.append(img)
49         new_target.append(target)
50
51         self.data = torch.Tensor(np.stack(new_data, axis=0)).to(device)
52         self.targets = torch.Tensor(new_target).to(torch.int64).to(device)
53
54     def __getitem__(self, index):
55         return self.data[index], self.targets[index]
56
57 # train_dset = torchvision.datasets.CIFAR10(root='./CIFAR10',train=True,
58 #     download=False,transform=train_transform)
59 # test_dset = torchvision.datasets.CIFAR10(root='./CIFAR10',train=False,
60 #     download=False,transform=transforms.ToTensor())
61 train_dset = CustomCIFAR10(root='./CIFAR10',train=True,download=False,
62     transform=train_transform)
63 test_dset = CustomCIFAR10(root='./CIFAR10',train=False,download=False,
64     transform=transforms.ToTensor())
65
66 train_loader = torch.utils.data.DataLoader(train_dset, batch_size=128,
67     shuffle=True, num_workers=0)
68 test_loader = torch.utils.data.DataLoader(test_dset, batch_size=128, shuffle=
69     False, num_workers=0)
70
71 #####
72
73 ##### 构建模型 #####
74
75 class Net(nn.Module):
76     def __init__(self,act):
77         super(Net, self).__init__()
78         # 卷积层 (32x32x3的图像)
79         self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
80         # 卷积层(16x16x16)
81         self.conv2 = nn.Conv2d(16, 32, 3, padding=1)

```

```

74     # 卷积层(8x8x32)
75     self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
76     # 最大池化层
77     self.pool = nn.MaxPool2d(2, 2)
78     # linear layer (64 * 4 * 4 -> 500)
79     self.fc1 = nn.Linear(64 * 4 * 4, 500)
80     # linear layer (500 -> 10)
81     self.fc2 = nn.Linear(500, 10)
82     if act == 'relu':
83         self.act = F.relu
84     elif act == 'tanh':
85         self.act = F.tanh
86     elif act == 'sigmoid':
87         self.act = F.sigmoid
88
89     def forward(self, x):
90         # add sequence of convolutional and max pooling layers
91         x = self.pool(self.act(self.conv1(x)))
92         x = self.pool(self.act(self.conv2(x)))
93         x = self.pool(self.act(self.conv3(x)))
94         # flatten image input
95         x = x.view(-1, 64 * 4 * 4)
96
97         x = self.act(self.fc1(x))
98
99         x = self.fc2(x)
100     return x
101 #####
102
103 ##### 模型加入batchnorm #####
104 class BnNet(nn.Module):
105     def __init__(self, act):
106         super(BnNet, self).__init__()
107         # 卷积层 (32x32x3的图像)
108         self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
109         self.bn1 = nn.BatchNorm2d(16)
110
111         # 卷积层(16x16x16)
112         self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
113         self.bn2 = nn.BatchNorm2d(32)
114
115         # 卷积层(8x8x32)
116         self.conv3 = nn.Conv2d(32, 64, 3, padding=1)

```

```
117     self.bn3 = nn.BatchNorm2d(64)
118
119     # 最大池化层
120     self.pool = nn.MaxPool2d(2, 2)
121
122     # linear layer (64 * 4 * 4 -> 500)
123     self.fc1 = nn.Linear(64 * 4 * 4, 500)
124     self.bn4 = nn.BatchNorm1d(500)
125
126     # linear layer (500 -> 10)
127     self.fc2 = nn.Linear(500, 10)
128
129     if act == 'relu':
130         self.act = F.relu
131     elif act == 'tanh':
132         self.act = F.tanh
133     elif act == 'sigmoid':
134         self.act = F.sigmoid
135
136     def forward(self, x):
137         # add sequence of convolutional and max pooling layers
138         x = self.pool(self.act(self.bn1(self.conv1(x))))
139         x = self.pool(self.act(self.bn2(self.conv2(x))))
140         x = self.pool(self.act(self.bn3(self.conv3(x))))
141
142         # flatten image input
143         x = x.view(-1, 64 * 4 * 4)
144
145         x = self.act(self.bn4(self.fc1(x)))
146         x = self.fc2(x)
147         return x
148 ##### 构建模型 #####
149
150
151
152 class DeepNet(nn.Module):
153     def __init__(self):
154         super(DeepNet, self).__init__()
155         ##### 代码填空: 请在此填补模型定义代码 #####
156         self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
157         self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
158         self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
159         self.conv4 = nn.Conv2d(64, 128, 3, padding=1)
```

```

160     self.conv5 = nn.Conv2d(128, 256, 3, padding=1)
161     self.conv6 = nn.Conv2d(256, 512, 1, padding=0)
162     self.fc1 = nn.Linear(512, 256)
163     self.fc2 = nn.Linear(256, 128)
164     self.fc3 = nn.Linear(128, 10)
165     self.act = F.tanh
166     self.pool = nn.MaxPool2d(2, 2)
167     self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
168     #####
169
170     def forward(self, x):
171         # convolutional layers
172         ##### 代码填空: 请在此填补前向计算代码 #####
173         x = self.act(self.conv1(x))
174         x = self.pool(self.act(self.conv2(x)))
175         x = self.act(self.conv3(x))
176         x = self.pool(self.act(self.conv4(x)))
177         x = self.act(self.conv5(x))
178         x = self.avgpool(self.act(self.conv6(x)))
179         x = x.view(-1, 512)
180
181         x = self.act(self.fc1(x))
182         x = self.act(self.fc2(x))
183         x = self.act(self.fc3(x))
184         #####
185         return x
186
187     class BnDeepNet(nn.Module):
188         def __init__(self, act):
189             super(BnDeepNet, self).__init__()
190             ##### 代码填空: 请在此填补模型定义代码 #####
191             self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
192             self.bn1 = nn.BatchNorm2d(16)
193             self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
194             self.bn2 = nn.BatchNorm2d(32)
195             self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
196             self.bn3 = nn.BatchNorm2d(64)
197             self.conv4 = nn.Conv2d(64, 128, 3, padding=1)
198             self.bn4 = nn.BatchNorm2d(128)
199             self.conv5 = nn.Conv2d(128, 256, 3, padding=1)
200             self.bn5 = nn.BatchNorm2d(256)
201             self.conv6 = nn.Conv2d(256, 512, 1, padding=0)
202             self.bn6 = nn.BatchNorm2d(512)

```



```

203     self.fc1 = nn.Linear(512, 256)
204     self.bnfc1 = nn.BatchNorm1d(256)
205     self.fc2 = nn.Linear(256, 128)
206     self.bnfc2 = nn.BatchNorm1d(128)
207     self.fc3 = nn.Linear(128, 10)
208     if act == 'relu':
209         self.act = F.relu
210     elif act == 'tanh':
211         self.act = F.tanh
212     elif act == 'sigmoid':
213         self.act = F.sigmoid
214     self.pool = nn.MaxPool2d(2, 2)
215     self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
216     #####
217
218     def forward(self, x):
219         # convolutional layers
220         ##### 代码填空: 请在此填补前向计算代码 #####
221         x = self.act(self.bn1(self.conv1(x)))
222         x = self.pool(self.act(self.bn2(self.conv2(x))))
223         x = self.act(self.bn3(self.conv3(x)))
224         x = self.pool(self.act(self.bn4(self.conv4(x))))
225         x = self.act(self.bn5(self.conv5(x)))
226         x = self.avgpool(self.act(self.bn6(self.conv6(x))))
227         x = x.view(-1, 512)
228
229         x = self.act(self.bnfc1(self.fc1(x)))
230         x = self.act(self.bnfc2(self.fc2(x)))
231         x = self.act(self.fc3(x))
232         return x
233         #####
234
235
236     ##### 训练前准备 #####
237
238
239     model = BnDeepNet('relu').to(device)
240     criterion = nn.CrossEntropyLoss().to(device)
241
242     optimizer_type = "AdamW" #或者换成AdamW
243     if optimizer_type == "SGD":
244         optimizer = optim.SGD(model.parameters(), lr=0.001)
245     elif optimizer_type == "AdamW":

```

```
246 ##### 代码填空: 请在此填补Adam优化器计算代码, lr=0.0001 #####
247 optimizer = optim.AdamW(model.parameters(), lr=0.0001)
248 #####
249 pass
250
251 n_epochs = 100
252 train_losses = []
253 valid_losses = []
254 accuracies = []
255 #####
256
257
258 ##### 训练+验证 #####
259 for epoch in range(n_epochs):
260     train_loss = 0.0
261     valid_loss = 0.0
262     model.train()
263     for idx,(img,label) in tqdm(enumerate(train_loader)):
264         # img, label = img.to(device), label.to(device)
265         optimizer.zero_grad()
266         output = model(img)
267         loss = criterion(output,label)
268         loss.backward()
269         optimizer.step()
270         train_loss += loss.item() * img.shape[0]
271
272     model.eval()
273     correct = 0
274     total = 0
275     for idx,(img,label) in tqdm(enumerate(test_loader)):
276         # img, label = img.to(device), label.to(device)
277         output = model(img)
278         loss = criterion(output, label)
279         valid_loss += loss.item() * img.shape[0]
280         _, predicted = torch.max(output.data, 1)
281         total += label.size(0)
282         correct += (predicted == label).sum().item()
283
284     train_loss = train_loss / len(train_dset)
285     valid_loss = valid_loss / len(test_dset)
286
287     train_losses.append(train_loss)
288     valid_losses.append(valid_loss)
```

```
289     accuracy = correct / total
290     accuracies.append(accuracy)
291
292     print(f"Epoch:{epoch}, Acc:{correct/total}, Train Loss:{train_loss}, Test
          Loss:{valid_loss}")
293     #####
294
295     ##### 曲线绘制 #####
296     print("MAX ACC: ", np.max(accuracies))
297     plt.plot(range(n_epochs), train_losses, label='Train Loss')
298     plt.plot(range(n_epochs), valid_losses, label='Valid Loss')
299     plt.xlabel('Epoch')
300     plt.ylabel('Loss')
301     plt.legend()
302     plt.savefig("Loss.png")
303     plt.clf()
304     # 绘制验证集准确率随epoch的变化曲线
305     plt.plot(range(n_epochs), accuracies, label='Accuracy')
306     plt.xlabel('Epoch')
307     plt.ylabel('Accuracy')
308     plt.legend()
309     plt.savefig("Acc.png")
310     #####
```