**Homework 4-8-3**

**2.1**

1. **def** three\_layer\_convnet(x, params):
2. conv\_w1, conv\_b1, conv\_w2, conv\_b2, fc\_w, fc\_b = params
3. scores = None
4. ###############################################################################
5. # TODO: Implement the forward pass for the threelayer ConvNet.                #
6. ###############################################################################
8. channel\_1, \_, KH1, KW1 = conv\_w1.shape
9. channel\_2, \_, KH2, KW2 = conv\_w2.shape
10. x = F.relu(F.conv2d(x, conv\_w1, bias=conv\_b1, padding=2))
11. x = F.relu(F.conv2d(x, conv\_w2, bias=conv\_b2, padding=1))
12. x = flatten(x)
13. scores = x.mm(fc\_w) + fc\_b
15. ###############################################################################
16. #                                 END OF YOUR CODE                            #
17. ###############################################################################
18. **return** scores

**2.2**

1. ################################################################################
2. # TODO: Initialize the parameters of a three-layer ConvNet.                    #
3. ################################################################################
4. conv\_w1 = random\_weight((32,3,5,5))
5. conv\_b1 = zero\_weight(32)
6. conv\_w2 = random\_weight((16,32,3,3))
7. conv\_b2 = zero\_weight(16)
8. fc\_w = random\_weight((16384,10))
9. fc\_b = zero\_weight(10)
10. ################################################################################
11. #                                 END OF YOUR CODE                             #
12. ################################################################################

**3.1**

1. **class** ThreeLayerConvNet(nn.Module):
2. **def** \_\_init\_\_(self, in\_channel, channel\_1, channel\_2, num\_classes):
3. super().\_\_init\_\_()
4. ########################################################################
5. # TODO: Set up the layers you need for a three-layer ConvNet with the  #
6. # architecture defined above.                                          #
7. ########################################################################
8. self.conv\_1 = nn.Conv2d(in\_channel, channel\_1, (5,5), padding=2)
9. nn.init.kaiming\_normal\_(self.conv\_1.weight)
10. self.conv\_2 = nn.Conv2d(channel\_1, channel\_2, (3,3), padding=1)
11. nn.init.kaiming\_normal\_(self.conv\_2.weight)
12. self.fc = nn.Linear(channel\_2\*32\*32,num\_classes)
13. nn.init.kaiming\_normal\_(self.fc.weight)
14. ########################################################################
15. #                          END OF YOUR CODE                            #
16. ########################################################################
18. **def** forward(self, x):
19. scores = None
20. ########################################################################
21. # TODO: Implement the forward function for a 3-layer ConvNet. you      #
22. # should use the layers you defined in \_\_init\_\_ and specify the        #
23. # connectivity of those layers in forward()                            #
24. ########################################################################
25. x = F.relu(self.conv\_1(x))
26. x = F.relu(self.conv\_2(x))
27. x = flatten(x)
28. scores = self.fc(x)
29. ########################################################################
30. #                             END OF YOUR CODE                         #
31. ########################################################################
32. **return** scores

**3.2**

1. ################################################################################
2. # TODO: Instantiate your ThreeLayerConvNet model and a corresponding optimizer #
3. ################################################################################
4. model = nn.Sequential(
5. nn.Conv2d(3,channel\_1,(5,5),padding=2),
6. nn.ReLU(),
7. nn.Conv2d(channel\_1,channel\_2,(3,3),padding=1),
8. nn.ReLU(),
9. Flatten(),
10. nn.Linear(channel\_2\*32\*32,10)
11. )
12. optimizer = optim.SGD(model.parameters(),lr=learning\_rate,momentum=0.9,nesterov=True)
13. ################################################################################
14. #                                 END OF YOUR CODE   #
15. ################################################################################

**4**

1. ################################################################################
2. # TODO: Instantiate your ThreeLayerConvNet model and a corresponding optimizer #
3. ################################################################################
4. model = nn.Sequential(
5. nn.Conv2d(3,channel\_1,(5,5),padding=2),
6. nn.ReLU(),
7. nn.Conv2d(channel\_1,channel\_2,(3,3),padding=1),
8. nn.ReLU(),
9. Flatten(),
10. nn.Linear(channel\_2\*32\*32,10)
11. )
12. optimizer = optim.SGD(model.parameters(),lr=learning\_rate,momentum=0.9,nesterov=True)
13. ################################################################################
14. #                                 END OF YOUR CODE   #
15. ################################################################################

**5. 残差神经网络**

思路：

加深网络结构可以提高分类的准确率，但是同时也会造成退化问题，导致准确率下降，这是由于

网络的非线性转换层次过深，已经无法实现线性变换。因此，我们使用ResNet，给每个block增加一条线性通路，解决退化问题。

利用残差神经网络，我们把网络的层次进一步加深，利用SGD优化：

1. ################################################################################
2. # TODO:                                                                        #
3. # Experiment with any architectures, optimizers, and hyperparameters.          #
4. # Achieve AT LEAST 70% accuracy on the \*validation set\* within 10 epochs.      #
5. #                                                                              #
6. # Note that you can use the check\_accuracy function to evaluate on either      #
7. # the test set or the validation set, by passing either loader\_test or         #
8. # loader\_val as the second argument to check\_accuracy. You should not touch    #
9. # the test set until you have finished your architecture and  hyperparameter   #
10. # tuning, and only run the test set once at the end to report a final value.   #
11. ################################################################################
12. model = None
13. optimizer = None
14. # \*\*\*\*\* START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)\*\*\*\*\*
16. **class** Residual(nn.Module):
17. **def** \_\_init\_\_(self, in\_channel, out\_channels, use\_1x1\_conv=False, strides=1):
18. super().\_\_init\_\_()
20. self.conv\_1 = nn.Conv2d(in\_channel, out\_channels, (3,3), padding=1, stride=strides)
21. nn.init.kaiming\_normal\_(self.conv\_1.weight)
22. self.conv\_2 = nn.Conv2d(out\_channels, out\_channels, (3,3), padding=1)
23. nn.init.kaiming\_normal\_(self.conv\_2.weight)
24. **if** use\_1x1\_conv:
25. self.conv\_3 = nn.Conv2d(in\_channel,out\_channels,(1,1),stride=strides)
26. nn.init.kaiming\_normal\_(self.conv\_3.weight)
27. **else**:
28. self.conv\_3 = None
29. self.use\_1x1\_conv = use\_1x1\_conv
30. self.bn\_1 = nn.BatchNorm2d(out\_channels)
31. self.bn\_2 = nn.BatchNorm2d(out\_channels)
33. **def** forward(self, x):
34. Y = F.relu(self.bn\_1(self.conv\_1(x)))
35. Y = self.bn\_2(self.conv\_2(Y))
36. **if** self.conv\_3:
37. x = self.conv\_3(x)
38. **return** F.relu(Y+x)
40. **def** resnet\_block(nn\_module, in\_channels, out\_channels, num\_residuals, block\_index, first\_block=False):
41. **for** i **in** range(num\_residuals):
42. **if** i == 0 **and** **not** first\_block:
43. stride = 1
44. **if** block\_index > 3:
45. stride = 2
46. nn\_module.add\_module('resnet\_block\_{:d}\_{:d}'.format(block\_index,i),Residual(in\_channels, out\_channels,use\_1x1\_conv=True,strides=stride))
47. **elif** i == 0:
48. nn\_module.add\_module('resnet\_block\_{:d}\_{:d}'.format(block\_index,i),Residual(in\_channels, out\_channels))
49. **else**:
50. nn\_module.add\_module('resnet\_block\_{:d}\_{:d}'.format(block\_index,i),Residual(out\_channels, out\_channels))
52. model = nn.Sequential(
53. nn.Conv2d(3,64,(7,7),stride=1,padding=3),
54. nn.BatchNorm2d(64),
55. nn.ReLU(),
56. nn.MaxPool2d((3,3),stride=1,padding=1)
57. )
59. resnet\_block(model,64,64,2,1,first\_block=True)
60. resnet\_block(model,64,128,2,2)
61. resnet\_block(model,128,256,2,3)
62. resnet\_block(model,256,512,2,4)
63. model.add\_module('avgpool',nn.AdaptiveAvgPool2d((1,1)))
64. model.add\_module('flatten',nn.Flatten())
65. model.add\_module('linear',nn.Linear(512,10))
67. optimizer = optim.SGD(model.parameters(),lr=1e-2,momentum=0.9,nesterov=True)
69. # \*\*\*\*\*END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)\*\*\*\*\*
71. ################################################################################
72. #                                 END OF YOUR CODE   #
73. ################################################################################

使用GPU的训练时间约为20min，最终在test set上的运行结果：

Checking accuracy on test set

Got 8242 / 10000 correct (82.42)