1. The codes are showed in the spiral.py
2. I tried to decrease the value of the number of hidden nodes from 10. The details are as follows.

|  |  |
| --- | --- |
| The number of Hidden nodes | Details |
| 10 | ep: 3400 loss: 0.0072 acc: 100.00 |
| 9 | ep: 2700 loss: 0.0245 acc: 100.00 |
| 8 | ep: 2500 loss: 0.0226 acc: 100.00 |
| 7 | ep:10800 loss: 0.0599 acc: 100.00 |
| 6 | ep: 6700 loss: 0.0366 acc: 100.00(Sometime) |
| 5 | ep:99900 loss: 0.0508 acc: 91.24 |

In the table, we could see that when the number of hidden nodes is 6, it could achieve 100% accuracy. However, when I tried it again, I found that it could not reach 100% accuracy sometimes. Therefore, the minimum number of hidden nodes if 7.

1. The codes are showed in the spiral.py
2. I tried different combinations of the value of the number of hidden nodes in [10, 9, 8, 7, 6, 5] and the initial weight in [0.001, 0.01, 0.1, 0.2, 0.3] The details are as follows.

|  |  |  |
| --- | --- | --- |
| The number of Hidden nodes | Init weight | Details(The best reasult) |
| 10 | 0.2 | ep: 5500 loss: 0.0948 acc: 100.00 |
| 9 | 0.1 | ep: 9300 loss: 0.0407 acc: 100.00 |
| 8 |  | Can not reach 100% with 20000 epochs |
| 7 |  | Can not reach 100% with 20000 epochs |
| 6 |  | Can not reach 100% with 20000 epochs |
| 5 |  | Can not reach 100% with 20000 epochs |

When the number of hidden nodes is 10, if the initial weight is 0.1, 0.2 and 0.3, it could achieve 100% accuracy. But if I set 0.1 or 0.3, it could not achieve 100% accuracy within 20000 epochs. When the number of hidden nodes is 9, if the initial weight is 0.1, it could achieve 100% accuracy before 200000 epochs. When the number of hidden nodes is 8, it can not achieve 100% accuracy within 2000 epochs when the initial weight is in [0.001, 0.01, 0.1, 0.2, 0.3]. When the number of hidden nodes is 7, it can not achieve 100% accuracy within 2000 epochs when the initial weight is in [0.001, 0.01, 0.1, 0.2, 0.3]. When the number of hidden nodes is 6, it can not achieve 100% accuracy within 2000 epochs when the initial weight is in [0.001, 0.01, 0.1, 0.2, 0.3]. When the number of hidden nodes is 5, it can not achieve 100% accuracy within 2000 epochs when the initial weight is in [0.001, 0.01, 0.1, 0.2, 0.3].

5.

(1) PolarNet

python spiral\_main.py --net polar --hid 7

ep: 5100 loss: 0.0194 acc: 100.00

polar1\_0.png

polar1\_1.png

图片包含 形状

描述已自动生成

polar1\_2.png

图片包含 背景图案

描述已自动生成

polar1\_3.png

图片包含 示意图

描述已自动生成

polar1\_4.png

图片包含 圆圈

描述已自动生成

polar1\_5.png

图片包含 形状

描述已自动生成

polar1\_6.png

图片包含 示意图

描述已自动生成

(2)

python spiral\_main.py --net raw --hid 10 --init 0.2

ep:13400 loss: 0.0168 acc: 100.00

raw1\_0.png

图片包含 背景图案

描述已自动生成

raw1\_1.png

图片包含 背景图案

描述已自动生成

raw1\_2.png

图片包含 背景图案

描述已自动生成

raw1\_3.png

图片包含 背景图案

描述已自动生成

raw1\_4.png 图片包含 背景图案

描述已自动生成

raw1\_5.png

图片包含 背景图案

描述已自动生成

raw1\_6.png 图片包含 背景图案

描述已自动生成

raw1\_7.png 图片包含 背景图案

描述已自动生成

raw1\_8.png 图片包含 背景图案

描述已自动生成

raw1\_9.png 图片包含 背景图案

描述已自动生成

raw2\_0.png 图片包含 图示

描述已自动生成

raw2\_1.png 图片包含 背景图案

描述已自动生成

raw2\_2.png 图片包含 示意图

描述已自动生成

raw2\_3.png 图片包含 背景图案

描述已自动生成

raw2\_4.png 图片包含 示意图

描述已自动生成

raw2\_5.png 图片包含 图表

描述已自动生成

raw2\_6.png 图片包含 图示

描述已自动生成

raw2\_7.png 图片包含 图示

描述已自动生成

raw2\_8.png 图片包含 示意图

描述已自动生成

raw2\_9.png 图片包含 形状

描述已自动生成

6.

1. For the PolarNet, we could see that the function generated non-linear features in the hidden layer from the pictures of the hidden layer in question 5. For the RawNet, we could see that the function generated linear features in the first hidden layer and generated non-linear features in the second hidden layer from the pictures in question 5. From these pictures, we also could get that every hidden node could only get a part of information of the classification boundary. The values of the hidden nodes multiply the weights plus the bias and then using the activation function is the result of the function. Therefore, the net need to use all the hidden nodes and the weights to achieve the classification.
2. In this part, I tried different values of initial weights in [0.001, 0.01, 0.1, 0.2, 0.3] in RawNet, which is recommended by the teacher. I found that if the value of the initial weight is 0.1 or 0.2, it is more likely to succeed. However, when I used 0.001 and 0.1, it always ended in failure. When I used 0.3, it could succeed sometimes. Besides, talking about the speed, in the question 2, I tried to set the initial weight as 0.1 and 0.2 to compare the speed when the number of hidden nodes is 10, I got the result that if the weight is 0.2, it used 5900 epochs to achieve 100% accuracy. However, when the initial weight is 0.1, it used more than 20000 epochs to achieve 100% accuracy sometimes. Therefore, I want to conclude that if the number of the initial weight is too larger or too small, it is very hard to reach the best point. So we need to consider and try to find the most suitable value of the initial weight in order to get good results basing on different conditions of the task.
4. Changing batch size

I tried to change the batch size from 97 to 194. In the PolarNet, it could achieve 100% accuracy in 3400 epochs if the number of hidden nodes is 10 and the batch size is 97. However, if I set the batch is 194, the number of epochs is only 900.

*ep: 900 loss: 0.1756 acc: 100.00*

Therefore, I conclude that if the batch size is larger, the speed of learning of some tasks is faster.

1. I tried the SGD optimizer instead of Adam optimizer.

Optimizer = torch.optim.SGD(net.parameters(), lr=args.lr, momentum=0.9,weight\_decay=0.0001)

It does not perform well as Adam optimizer.

*ep:12900 loss: 0.0130 acc: 100.00*

I think the reason is that Adam is an extended SGD optimizer. Adam optimizer uses momentum and adaptive learning rate to speed up. Therefore, Adam optimizer is better than SGD optimizer.

1. I tried to change the activation from tanh to relu.

*self.hidden\_layer\_1 = torch.rule(self.linear1(input))*

However, the model performed worse. I think the reason is that when the x is smaller than 0, the output is 0. So, in the learning process, zero output will make the gradient become zero. It is difficult for the model to learn.

1. I tried to add the third hidden layer in the RawNet. It also has a good performance.

*ep: 5700 loss: 0.0063 acc: 100.00*

I think that the hidden layer is useful to fit data to some extent. If the model has more hidden layer, the performance of representing data is better and the data is more abstract. So, it could perform better in the training set. However, if the number of hidden layers is too lager, it will over-fitting and does not perform well in testing set.