模型简介

1. 以nn.Linear作全连接层，F.relu作激活函数，全连接一次激活一次，最后再做一次全连接

超参数

网络规模：各线性层规模：1*20，（relu,）20*20，（relu,）20*30，（relu,）30*1

样本数量=5000（4000 for train，1000 for test)

train cycle num=150

learning rate=0.01(*=0.5 every 5 cycles)

optimizer=optim.SGD(model.parameters(),lr=0.01,momentum=0.5)

loss function=nn.MSELoss()

2. 对每个隐藏层，作层归一化(一个维度为1，经尝试，不能用BatchNorm1d，**因为一批只有一个样本~~**)

```
x=(x-torch.mean(x))/torch.var(x)
```

3. 采用SGD优化器，正则防过拟合，并设置momentum，也可这里设置衰减率作L2正则化

```
optimizer=optim.SGD(model.parameters(),lr=0.01,momentum=0.5)
```

4. 对损失函数采用L2正则化，

```
l2_regularization = sum(torch.sum(torch.pow(param, 2)) for param in
model.parameters())
loss=criterion(out,y)+l2_lambda * l2_regularization
```

5. L2正则化后出现梯度爆炸，loss.backward()计算出梯度后用梯度裁剪

```
loss.backward()     #loss梯度下降求导
nn.utils.clip_grad_norm_(model.parameters(), 1)
```

6. 采用较大规模的权重矩阵而非一味加深网络（好像可以获得更好的性能）,还可大幅提升训练速度

Confronted problem&solution

1. 不要把带含导数的tensor的变量随便用
2. tensor.item()将单元素tensor变量转为标量，多元素用.tolist()
3. nn.Linear,F.relu()之类的输入都为.float(),不要看报错信息改成.double()
4. 输出一开始全一样，后面好了，不知为啥，

   即使学习率不变，权重也会按反向传播方法Gradient decline，

   让学习率随时间变化，可以避免梯度爆炸或太慢
5. loss=-0，把损失函数调为MSELoss()好了，
6. test acc还有>1的

7. batch_size的处理——如果DataLoader() set batch_size，每次只取一个批量吗，嗯嗯

```
import torch
import torch.utils.data as data

# 定义数据集
class MyDataset(torch.utils.data.Dataset):
    def __init__(self, data, target):
        self.data = torch.tensor(data, dtype=torch.float32)
        self.target = torch.tensor(target, dtype=torch.long)

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        return self.data[idx], self.target[idx]

# 定义超参数
batch_size = 32

# 准备数据
data = torch.randn(1000, 10)
target = torch.randint(0, 2, (1000,))
train_dataset = MyDataset(data, target)

# 创建数据加载器
train_loader = data.DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)

# 迭代数据加载器
for batch_idx, (data, target) in enumerate(train_loader):
    # 在这里进行模型训练
    print('Batch %d, data shape: %s, target shape: %s' % (batch_idx,
str(data.shape), str(target.shape)))
```

7. 加载到GPU上报错：

- 是x=x.to(device),不是x.to(device)

8. 把dataset加载到GPU上，

- ~~DataSet不支持.to()方法~~，MydataSet初始化数据时，直接to(device)
- 对net，net.to(device)就行，other parameter?

9. 可以调学习率

10. 在本地CPU上，Test_acc>>Train_acc，想把损失函数也正则化，

    ~~不过为啥本地CPU和云上GPU训练的结果差这么大，云上GPU test result挺好的~~

11. apply nn.Normalization1d时，报错

    ```
    ValueError: expected 2D or 3D input (got 1D input)
    ```

    是输入维度不正确

- nn.Normalization1d和nn.Normalization2d的区别，有nn.Normalization3d吗?

> `nn.Normalization1d`和`nn.Normalization2d`是PyTorch中常用的标准化层。它们都是对输入的特征进行标准化处理，使其具有零均值和单位方差。其中，`nn.Normalization1d`用于处理一维特征（例如时间序列数据、语音信号），`nn.Normalization2d`用于处理二维特征（例如图像数据）。
>
> 具体而言，`nn.Normalization1d`对于一个大小为 (batch_size, num_features) 的输入，会计算出所有样本在每个特征上的均值和标准差，并用这些值对输入数据进行标准化。而`nn.Normalization2d`则会计算每个通道在所有样本上的均值和标准差，并用这些值对输入数据进行标准化。
>
> 在PyTorch中也有`nn.Normalization3d`，它用于处理三维特征（例如视频数据）。与`nn.Normalization2d`类似，它会计算每个通道在所有样本上的均值和标准差，并对输入数据进行标准化。

- 经测试，Normalization1d的input不能由维度为1

  > 在值=num_featues的维度进行规范化，
  > dim1宽度=2，按dim1规范化

  为啥呢，若维度=1，directly：x=(x-torch.mean(x))/torch.var(x)

```python
import torch.nn as nn
import torch

y=torch.randn(2,5)
print(y)
bn = nn.BatchNorm1d(5)
z=bn(y)
print(z)

x = torch.tensor([[1,2],[1,0],[1,1]],dtype=torch.float32)
bnx = nn.BatchNorm1d(2)
print(bnx(x))
#output = bn(x)

x=torch.arange(10,dtype=torch.float32)
x=(x-torch.mean(x))/torch.var(x)
print(x)
print((x*x).sum())
```

✓ 0.0s

```
tensor([[-0.0077,  1.0932,  0.3640,  0.6439,  0.6342],
        [ 0.1458, -0.6529, -0.8407,  0.4843, -0.5998]])
tensor([[-0.9992,  1.0000,  1.0000,  0.9992,  1.0000],
        [ 0.9992, -1.0000, -1.0000, -0.9992, -1.0000]],
       grad_fn=<NativeBatchNormBackward0>)
tensor([[ 0.0000,  1.2247],
        [ 0.0000, -1.2247],
        [ 0.0000,  0.0000]], grad_fn=<NativeBatchNormBackward0>)
tensor([-0.4909, -0.3818, -0.2727, -0.1636, -0.0545,  0.0545,  0.1636,  0.2727,
         0.3818,  0.4909])
tensor(0.9818)
```

12. nn.Flatten()，批中各对象展平

```
import torch.nn as nn
import torch

x=torch.randn(2,3,4)
#tensor.size()=(batch_size,C,H,W,...)
fla=nn.Flatten()
#return.size()=(batch_size,C*H*W*...)
y=fla(x)
print(y.size())
```

✓ 0.0s

```
torch.Size([2, 12])
```

13. 损失函数添加L1,L2正则化项，特别当训练误差很小，test error很大

> 在使用PyTorch训练模型时，可使用两种方式添加L1/L2正则化：一种是添加正则化项到损失函数中，另一种是在 backward () 之后，添加正则化项到参数变量的梯度中，然后再进行 step ()。12

```
def train(model, iterator, optimizer, criteon):
    avg_acc, avg_loss = [], []
    model.train()

    for batch in tqdm(iterator):
        text, label = batch[0].cuda(), batch[1].cuda()

        pred = model(text)
        l1_penalty = L1_Weight * sum([p.abs().sum() for p in
model.fc.parameters()])
        #l2_penalty = L2_Weight * sum([(p*p).sum() for p in
model.fc.parameters()])
        loss = criteon(pred, label.long())
        loss_with_penalty = loss + l1_penalty
        #loss_with_penalty = loss + l2_penalty

        acc = utils.binary_acc(torch.argmax(pred.cpu(), dim=1),
label.cpu().long())
        avg_acc.append(acc)
        avg_loss.append(loss.item())

        optimizer.zero_grad()
        loss_with_penalty.backward()
        #loss.backward()
        optimizer.step()

    avg_acc = np.array(avg_acc).mean()
    avg_loss = np.array(avg_loss).mean()
    train_metrics = {'train_acc': avg_acc,
                     'train_loss': avg_loss
                     }
    logging.info(train_metrics)
    return avg_acc, avg_loss
```

- 加正则化项失败*1

```python
for epoch in range(num_epochs):
    train_loss=0
    train_acc=0
    if epoch%5==0:
        optimizer.param_groups[0]['lr']*=0.5
    for x,y in train_dataloader:
        out=model(x)
        loss=criterion(out,y)
        l2_penalty = L2_Weight * sum([(p**2).sum() for p in model.parameters()])
        loss_l2=loss+l2_penalty
        You, 27秒钟前 • Uncommitted changes
        optimizer.zero_grad()
        loss_l2.backward()    #loss梯度下降求导
        optimizer.step()    #利用这个导数迭代
        train_loss+=loss_l2.item()
#         print(loss.item())
        train_acc+=(abs(out*1.0/y)).item()  #转为标量
#         print(type(train_loss))
```

问题 2 输出 调试控制台 终端 GITLENS JUPYTER powershell - lab1_fnn

```
    func(params,
  File "C:\software\Python_3_10\lib\site-packages\torch\optim\sgd.py", line 238, in _single_tensor_sgd
    buf.mul_(momentum).add_(d_p, alpha=1 - dampening)
KeyboardInterrupt
PS C:\Code\dlcode\Code_d21\lab1_fnn> python lab1_v2.py
cpu
cpu
8000
<class 'int'> <class 'float'>
epoch:0,Train Loss: nan,Train acc: nan,Test Loss: nan,Test acc: nan
<class 'int'> <class 'float'>
epoch:1,Train Loss: nan,Train acc: nan,Test Loss: nan,Test acc: nan
<class 'int'> <class 'float'>
epoch:2,Train Loss: nan,Train acc: nan,Test Loss: nan,Test acc: nan
Traceback (most recent call last):
  File "C:\Code\dlcode\Code_d21\lab1_fnn\lab1_v2.py", line 76, in <module>
    l2_penalty = L2_Weight * sum([(p**2).sum() for p in model.parameters()])
  File "C:\Code\dlcode\Code_d21\lab1_fnn\lab1_v2.py", line 76, in <listcomp>
    l2_penalty = L2_Weight * sum([(p**2).sum() for p in model.parameters()])
  File "C:\software\Python_3_10\lib\site-packages\torch\nn\modules\module.py", line 1710, in parameters
    for name, param in self.named_parameters(recurse=recurse):
  File "C:\software\Python_3_10\lib\site-packages\torch\nn\modules\module.py", line 1737, in named_parameters
    for elem in gen:
  File "C:\software\Python_3_10\lib\site-packages\torch\nn\modules\module.py", line 1679, in _named_members
    for module_prefix, module in modules:
  File "C:\software\Python_3_10\lib\site-packages\torch\nn\modules\module.py", line 1883, in named_modules
    memo.add(self)
KeyboardInterrupt
```

- 加入正则化项后

```python
for epoch in range(num_epochs):
    train_loss=0
    train_acc=0
    if epoch%5==0:
        optimizer.param_groups[0]['lr']*=0.5
    for x,y in train_dataloader:
        out=model(x)
        l2_regularization = sum([(p**2).sum() for p in model.layer1.parameters()])+sum([(p**2).sum() for p in model.layer
        sum([(p**2).sum() for p in model.layer3.parameters()])+sum([(p**2).sum() for p in model.layer4.parameters()])
#         print(l2_regularization.size())
        loss=criterion(out,y)+l2_lambda * l2_regularization

        optimizer.zero_grad()
        loss.backward()    #loss梯度下降求导
        optimizer.step()    #利用这个导数迭代
        train_loss+=loss.item()
#         print(loss.item())
```

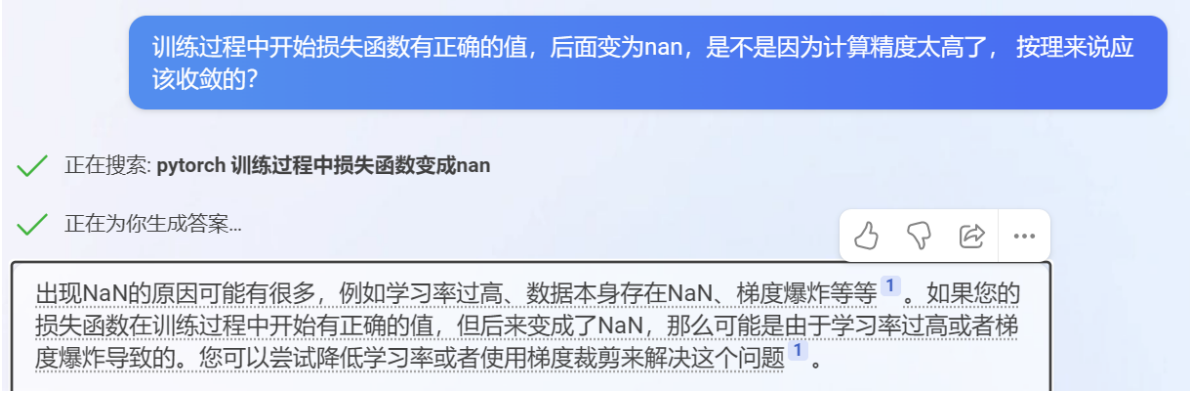
问题 3 输出 调试控制台 终端 GITLENS JUPYTER python - lab1_fnn

```
KeyboardInterrupt
PS C:\Code\dlcode\Code_d21\lab1_fnn> python lab1_v2.py
cpu
cpu
8000
<class 'int'> <class 'float'>
epoch:0,Train Loss:0.734296,Train acc:2.264269,Test Loss:0.416949,Test acc:2.549321
<class 'int'> <class 'float'>
epoch:1,Train Loss:0.631529,Train acc:2.458284,Test Loss:0.610638,Test acc:1.990503
<class 'int'> <class 'float'>
epoch:2,Train Loss: nan,Train acc: nan,Test Loss: nan,Test acc: nan
<class 'int'> <class 'float'>
```

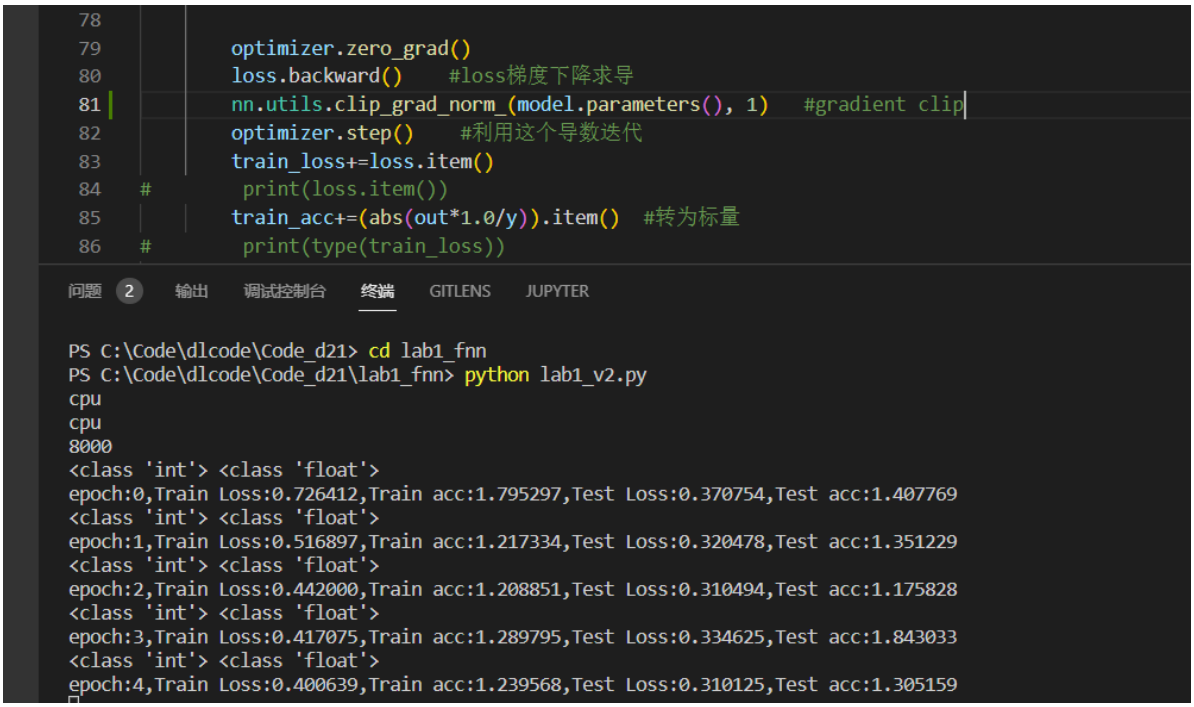
- 更标准的L2正则化项添加方法

```
        l2_regularization = sum(torch.sum(torch.pow(param, 2)) for param in model.parameters())
```

- confront gradient boom



- Adopt gradient clip

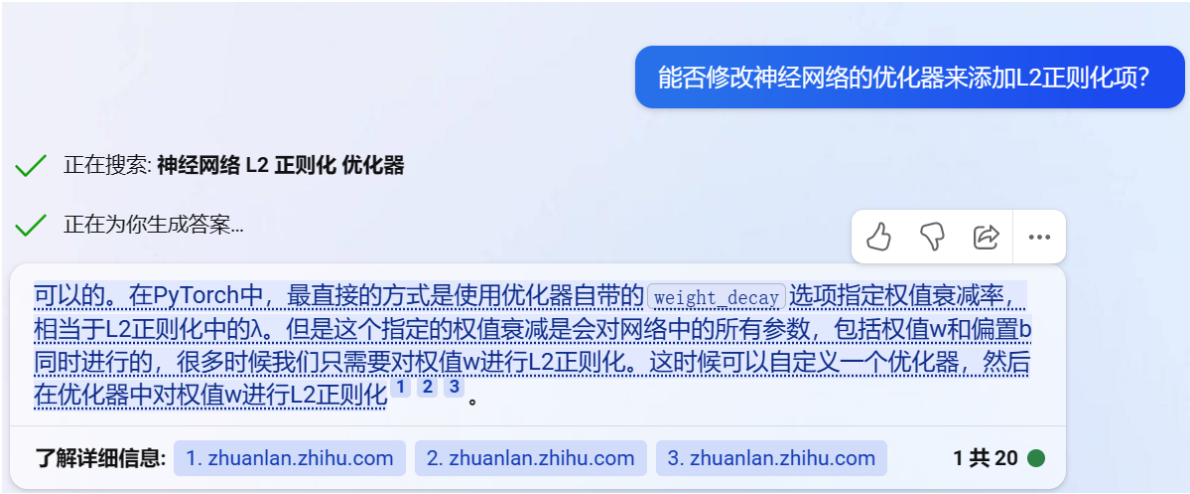


```
78
79          optimizer.zero_grad()
80          loss.backward()    #loss梯度下降求导
81          nn.utils.clip_grad_norm_(model.parameters(), 1)   #gradient clip
82          optimizer.step()    #利用这个导数迭代
83          train_loss+=loss.item()
84  #        print(loss.item())
85          train_acc+=(abs(out*1.0/y)).item()  #转为标量
86  #        print(type(train_loss))
```

问题 2　输出　调试控制台　终端　GITLENS　JUPYTER

```
PS C:\Code\dlcode\Code_d21> cd lab1_fnn
PS C:\Code\dlcode\Code_d21\lab1_fnn> python lab1_v2.py
cpu
cpu
8000
<class 'int'> <class 'float'>
epoch:0,Train Loss:0.726412,Train acc:1.795297,Test Loss:0.370754,Test acc:1.407769
<class 'int'> <class 'float'>
epoch:1,Train Loss:0.516897,Train acc:1.217334,Test Loss:0.320478,Test acc:1.351229
<class 'int'> <class 'float'>
epoch:2,Train Loss:0.442000,Train acc:1.208851,Test Loss:0.310494,Test acc:1.175828
<class 'int'> <class 'float'>
epoch:3,Train Loss:0.417075,Train acc:1.289795,Test Loss:0.334625,Test acc:1.843033
<class 'int'> <class 'float'>
epoch:4,Train Loss:0.400639,Train acc:1.239568,Test Loss:0.310125,Test acc:1.305159
```

修改优化器来添加L2正则化项



示例

以下是一个在PyTorch中修改神经网络的优化器来添加L2正则化项的示例代码： 1 2 3

```python
import torch.nn as nn
import torch.optim as optim

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(10, 10)
        self.fc2 = nn.Linear(10, 1)

    def forward(self, x):
        x = self.fc1(x)
        x = self.fc2(x)
        return x

net = Net()
optimizer = optim.SGD(net.parameters(), lr=0.01)

# 自定义优化器，对权值w进行L2正则化
class L2Regularization(optim.Optimizer):
    def __init__(self, optimizer, weight_decay):
        self.optimizer = optimizer
        self.weight_decay = weight_decay

    def step(self):
        for group in self.optimizer.param_groups:
            for param in group['params']:
                param.data -= self.weight_decay * param.data
        return self.optimizer.step()

optimizer = L2Regularization(optimizer, weight_decay=0.01)
```

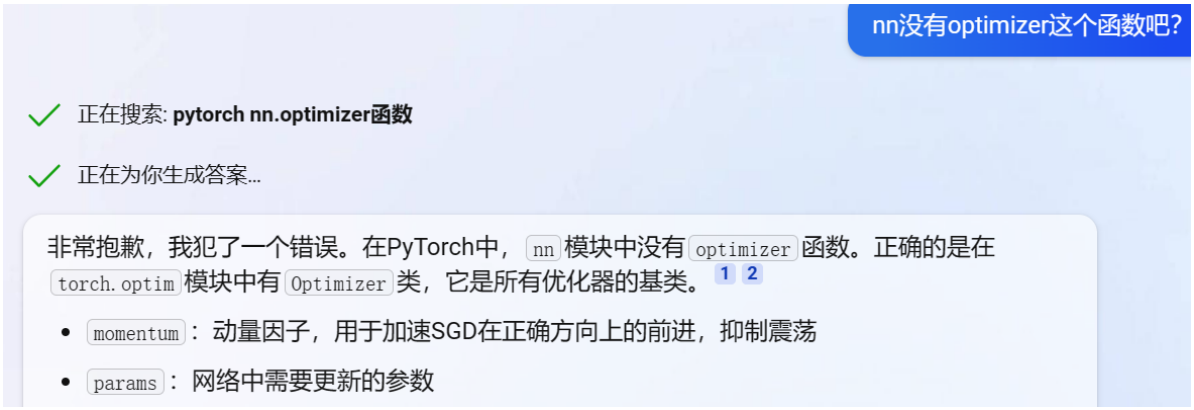
了解详细信息: 1. zhuanlan.zhihu.com 2. blog.csdn.net 3. blog.csdn.net +2 更多 3 共 20

14. torch.optim的参数

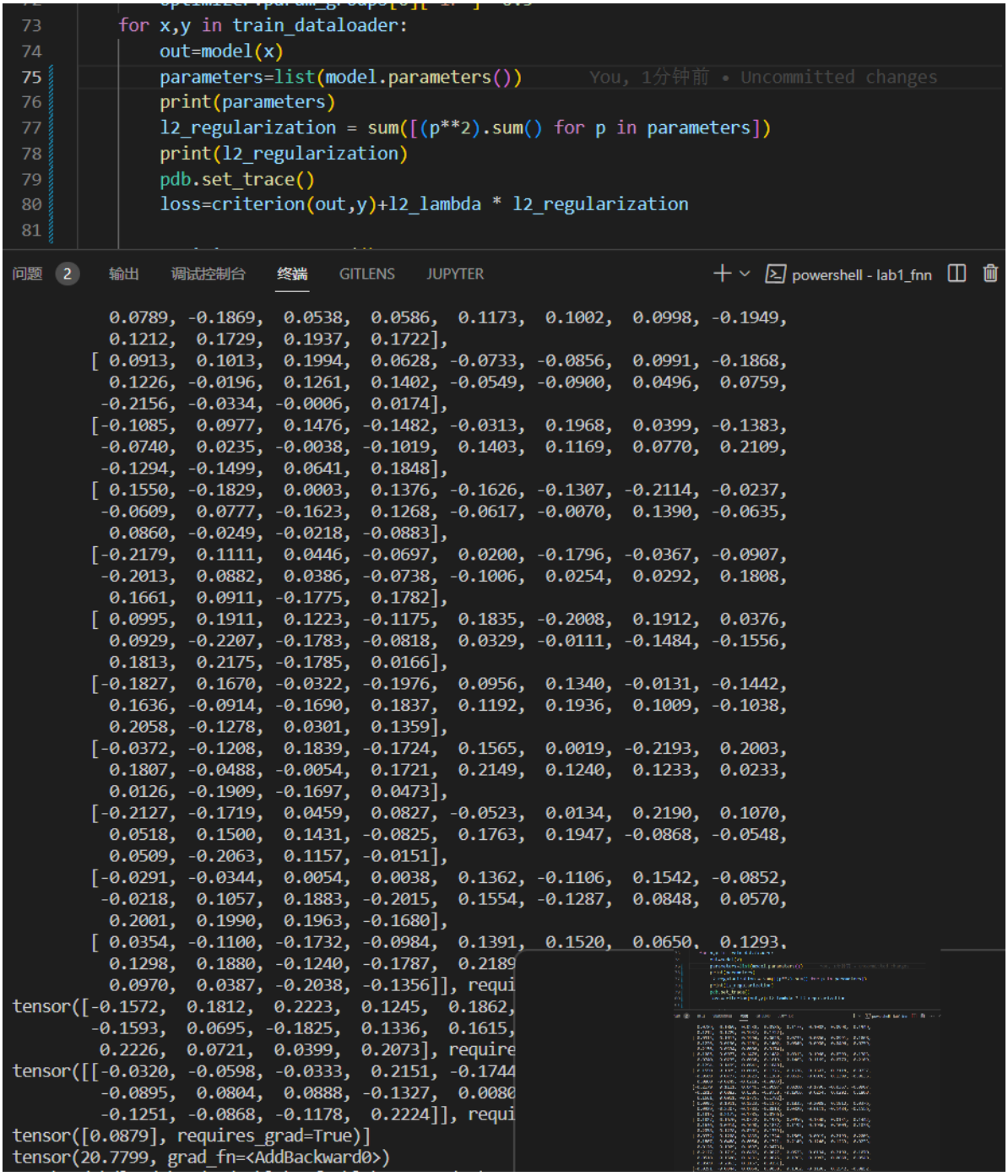
```
    5  help(torch.optim.SGD)
[4] ✓ 0.0s

... Output exceeds the size limit. Open the full output data in a text editor
Help on class SGD in module torch.optim.sgd:

class SGD(torch.optim.optimizer.Optimizer)
 |  SGD(params, lr=<required parameter>, momentum=0, dampening=0, weight_decay=0, nesterov=False, *, ma
 |
 |  Implements stochastic gradient descent (optionally with momentum).
 |
 |  .. math::
 |     \begin{aligned}
```



15. model's 权重参数print



16. torch是带grad()的变量，不能直接plt.plot()，用.item()转为普通数列，torch的任何运算都会产生torch!

17. 梯度下降后，与修改过的tensor变量相关的变量值会发生改变吗?

```
        l2_regularization = sum(torch.sum(torch.pow(param, 2)) for param in
model.parameters()) #这个也是torch!
#         print(l2_regularization.size())
        loss=criterion(out,y)+l2_lambda * l2_regularization

        optimizer.zero_grad()
        loss.backward()    #loss梯度下降求导
        nn.utils.clip_grad_norm_(model.parameters(), 1)   #gradient clip
        optimizer.step()    #利用这个导数迭代
        train_loss+=(loss-l2_lambda * l2_regularization).item() #这里要把
torch(l2_regulation)转为item()
```

这里l2_regulation在下降前后会变化吗，不会

不同条件结果比对

Num	situation+样本数+周期数	学习率+衰减率	loss final cycle	train_acc final cycle	test_loss	test_acc
1	(10,10)，无Normalization，有衰减，10000，500	(0.01,0.5)		1.20		
2	(20,20)，无Normalization，有衰减，10000，500	(0.01,0.5)		1.10		
3	(10,20,20)，无Normalization，有衰减，10000，500	(0.01,0.5)		1.20		
4	(10,20,20)，无Normalization，10000，500	(0.5,0.1)		15.x		
5	(40,40)，无Normalization，有衰减，10000，500	(0.01,0.4)		1.03		
6	(50,50)，无Normalization，10000，500	(0.01,0.4)		1.03		
7	(20,20,30)，规范化+梯度裁剪+L2正则化(1000 datas)，150	(0.01,0.5)		1.034		1.048
8	(20,20,30)，规范化+梯度裁剪+L2正则化(5000 datas)，150	(0.01,0.5)	0.002960	1.122541	0.002974	1.013080
9	(20,20,30)，规范化+梯度裁剪+L2正则化(10000 datas)，150	(0.01,0.5)	0.050139	1.010618		1.010062

Num	situation+样本数+周期数	学习率+衰减率	loss final cycle	train_acc final cycle	test_loss	test_acc
10	(50,50)，有规范化，无学习率衰减，150	(0.01,NULL)	0.550541	1.157888		1.0033341
11	(50,50)，有Norm，有衰减(5000)，150	(0.01,0.5)	0.367890	1.042537		1.03592

调参总结

- 3，4组与5，6组的对比，表明增大权重矩阵数目可能比加深网络深度更有效，

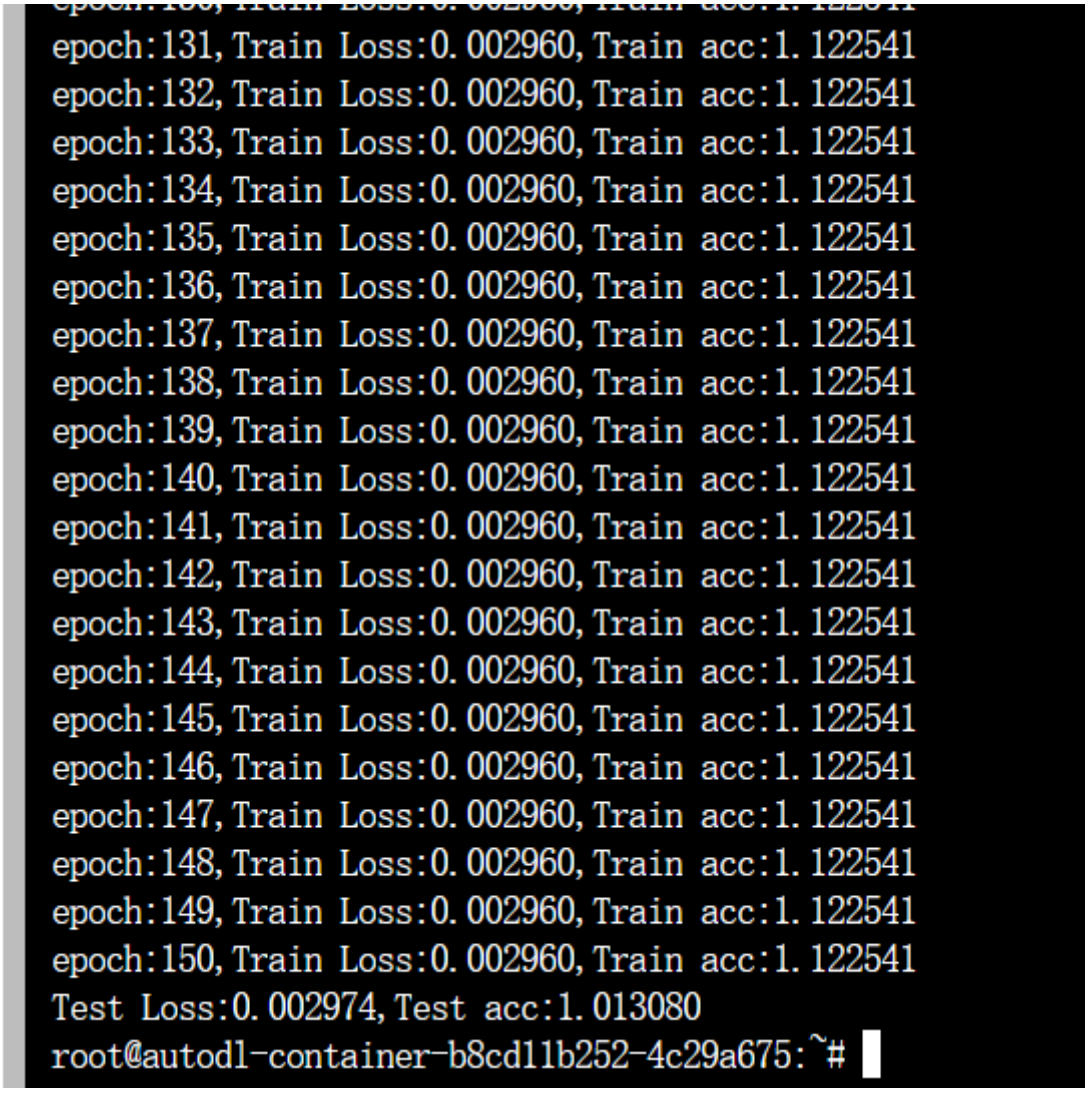
  5，6第10周期的train_acc≈3，4第500周期的train_acc，显示了weight matrix的优越性
- 3，4组的强烈对比说明，训练初期宁可把学习率设小点
- 3，4与7.8组的对比说明只增加深度，不批量正则化效果不变，
- 9表明隐藏层规范化和损失函数正则化可以提高训练效果，和增加深度一起更好，
- 7，8，9对比表明增大数据量可以提高训练效果，

  虽然8的训练准确度较低，这可能是生成数组的随机性导致的，但loss相比7显著较小，
- 5与10，11的对比说明不设置学习率衰减，容易陷入在最小值左右反复横跳，而无法逼近最小值
- 3，7，8，9的对比表明网络结构的加深增大显著提高训练效果
- 5，11的对比似乎表明规范化在本实验的效果不明显

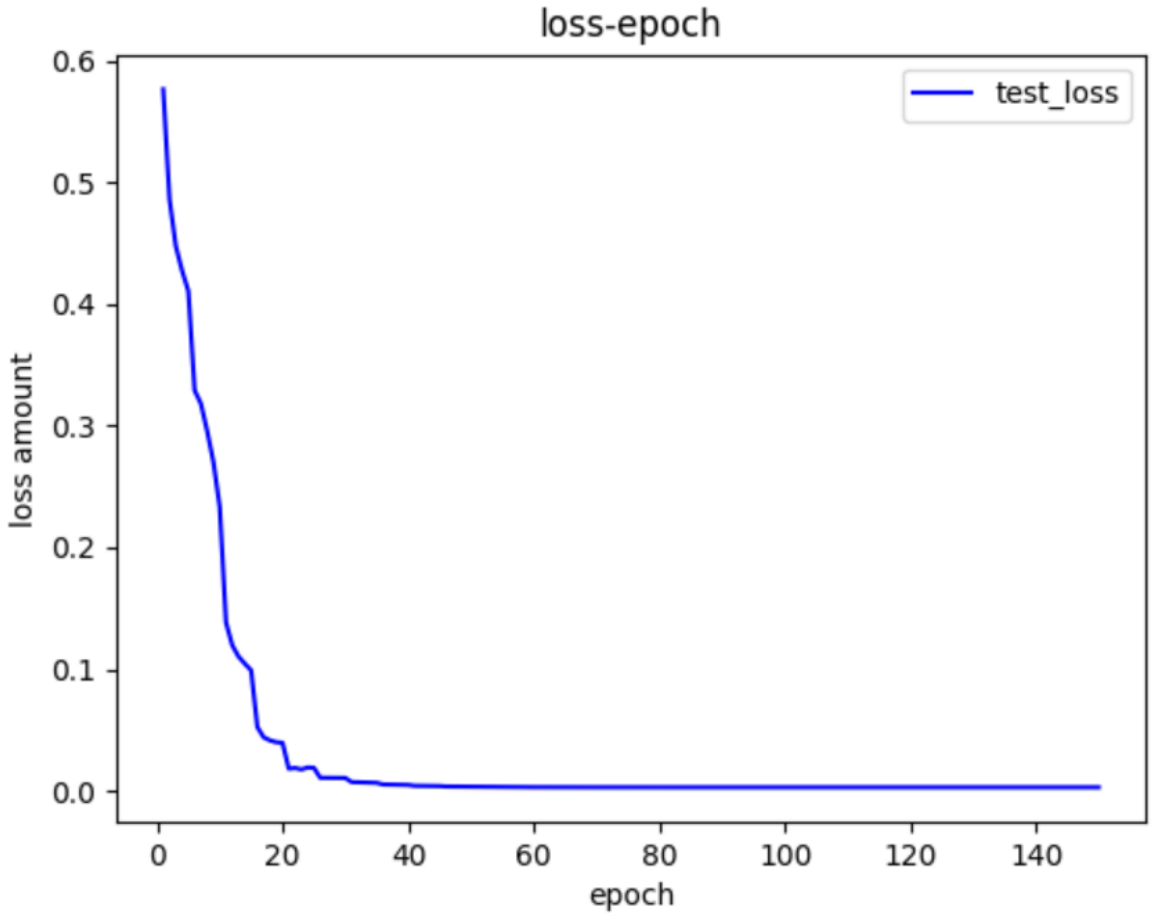
若干实验截图

~~一些忘记截图子~~

- num8(best result)

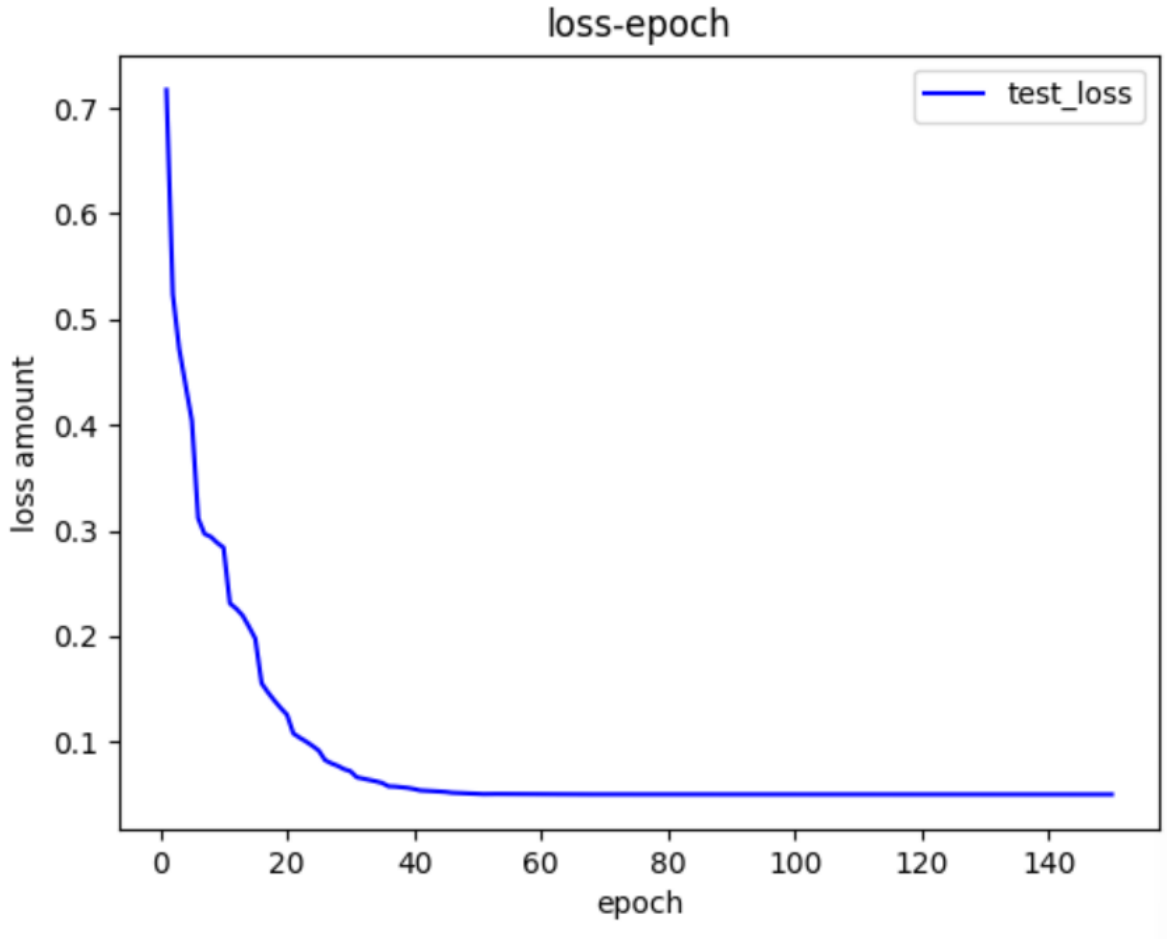

epoch:131, Train Loss:0.002960, Train acc:1.122541
epoch:132, Train Loss:0.002960, Train acc:1.122541
epoch:133, Train Loss:0.002960, Train acc:1.122541
epoch:134, Train Loss:0.002960, Train acc:1.122541
epoch:135, Train Loss:0.002960, Train acc:1.122541
epoch:136, Train Loss:0.002960, Train acc:1.122541
epoch:137, Train Loss:0.002960, Train acc:1.122541
epoch:138, Train Loss:0.002960, Train acc:1.122541
epoch:139, Train Loss:0.002960, Train acc:1.122541
epoch:140, Train Loss:0.002960, Train acc:1.122541
epoch:141, Train Loss:0.002960, Train acc:1.122541
epoch:142, Train Loss:0.002960, Train acc:1.122541
epoch:143, Train Loss:0.002960, Train acc:1.122541
epoch:144, Train Loss:0.002960, Train acc:1.122541
epoch:145, Train Loss:0.002960, Train acc:1.122541
epoch:146, Train Loss:0.002960, Train acc:1.122541
epoch:147, Train Loss:0.002960, Train acc:1.122541
epoch:148, Train Loss:0.002960, Train acc:1.122541
epoch:149, Train Loss:0.002960, Train acc:1.122541
epoch:150, Train Loss:0.002960, Train acc:1.122541
Test Loss:0.002974, Test acc:1.013080
root@autodl-container-b8cd11b252-4c29a675:~#

loss


loss-epoch
test_loss
loss amount
epoch
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
20
40
60
80
100
120
140

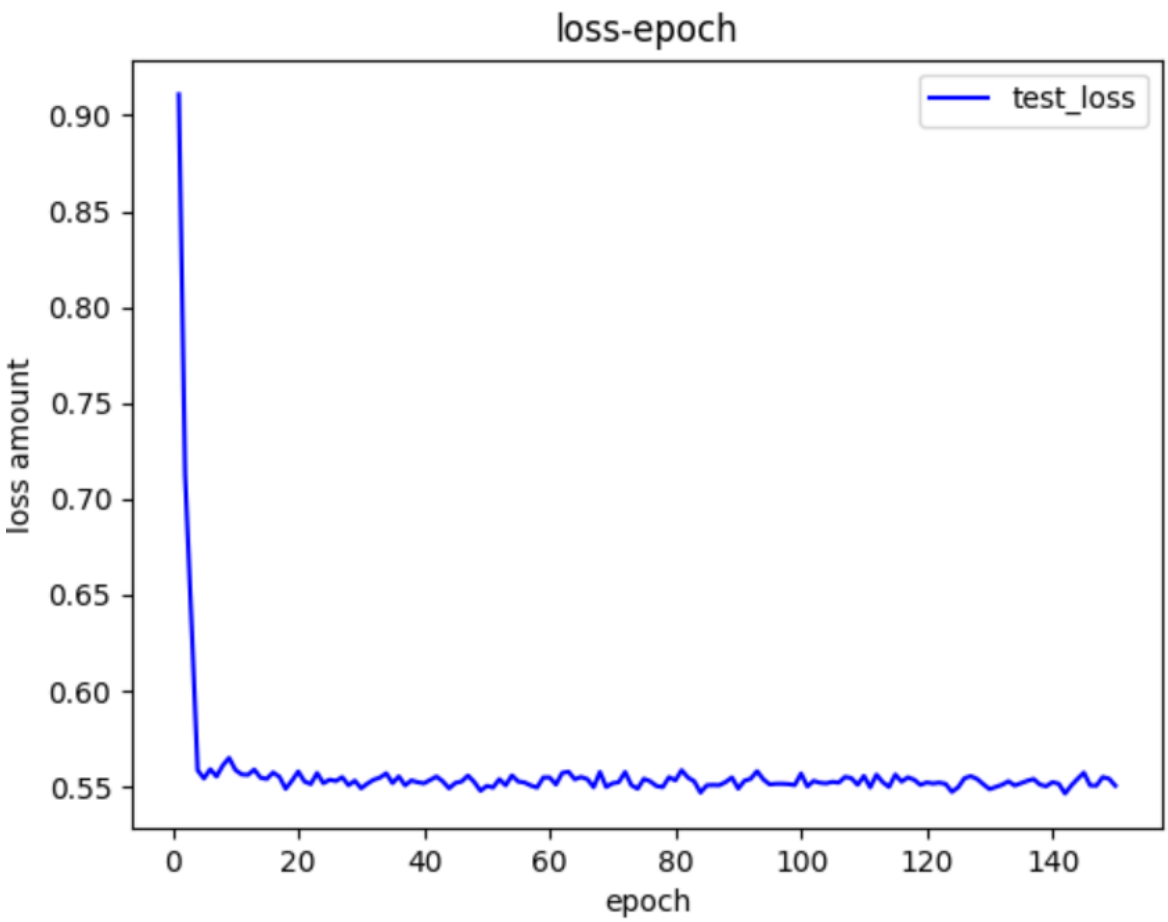
- num 9

loss


loss-epoch
test_loss
loss amount
epoch
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
20
40
60
80
100
120
140

- num10——无学习率衰减，参数在极小值两侧震荡

loss



- num11

```
epoch:127, Train Loss:0.367890, Train acc:1.042537
epoch:128, Train Loss:0.367890, Train acc:1.042537
epoch:129, Train Loss:0.367890, Train acc:1.042537
epoch:130, Train Loss:0.367890, Train acc:1.042537
epoch:131, Train Loss:0.367890, Train acc:1.042537
epoch:132, Train Loss:0.367890, Train acc:1.042537
epoch:133, Train Loss:0.367890, Train acc:1.042537
epoch:134, Train Loss:0.367890, Train acc:1.042537
epoch:135, Train Loss:0.367890, Train acc:1.042537
epoch:136, Train Loss:0.367890, Train acc:1.042537
epoch:137, Train Loss:0.367890, Train acc:1.042537
epoch:138, Train Loss:0.367890, Train acc:1.042537
epoch:139, Train Loss:0.367890, Train acc:1.042537
epoch:140, Train Loss:0.367890, Train acc:1.042537
epoch:141, Train Loss:0.367890, Train acc:1.042537
epoch:142, Train Loss:0.367890, Train acc:1.042537
epoch:143, Train Loss:0.367890, Train acc:1.042537
epoch:144, Train Loss:0.367890, Train acc:1.042537
epoch:145, Train Loss:0.367890, Train acc:1.042537
epoch:146, Train Loss:0.367890, Train acc:1.042537
epoch:147, Train Loss:0.367890, Train acc:1.042537
epoch:148, Train Loss:0.367890, Train acc:1.042537
epoch:149, Train Loss:0.367890, Train acc:1.042537
epoch:150, Train Loss:0.367890, Train acc:1.042537
Test Loss:0.350947, Test acc:1.035952
root@autodl-container-b8cd11b252-4c29a675:~#
```

终端 1 | 50_50.png | lab1_50_50.py