In [ ]:
```python
import torch as t
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import torch.optim as optim
#Author: 坚定的唯物主义鼠鼠
```

In [ ]:
```python
data=np.genfromtxt('./data/Iris_处理后.csv',delimiter=',')
#这里的处理是将 Iris-setosa、Iris-versicolor、Iris-virginica 分别替换为 0、1、2
data=data[1:,1:]#去掉第一行和第一列
np.random.set_state(np.random.get_state())
data=np.random.permutation(data) #打乱数据，对数据进行重排

#将数据转换为tensor，并划分为训练集和测试集
#训练集：测试集=2:1  ，即前100条为训练集，后50条为测试集(数据已经打乱)
# 测试集不参与训练，只用于测试模型的准确率

tu=t.from_numpy(data[:100,0:4]).float()                          #训练集的特征
tc=t.from_numpy(np.array(data[:100,4],dtype=np.int64))           #训练集的标签
tc=F.one_hot(tc,3).float()                  #将训练集标签转换为one-hot编码

vu=t.from_numpy(data[100:,0:4]).float()                          #测试集的特征
vc=t.from_numpy(np.array(data[100:,4],dtype=np.int64))           #测试集的标签
vc=F.one_hot(vc,3).float()                  #将测试集标签转换为one-hot编码
```

In [ ]:
```python
#定义网络    两层全连接层，中间加上一个ReLU函数作为激活函数
# 简单来说就是两层神经网络 （ 4个输入 10个隐藏层神经元  3个输出）
class Net(nn.Module):
    def __init__(self):
        super(Net,self).__init__()
        self.fc1=nn.Linear(4,10)            #输入层
        self.fc2=nn.Linear(10,3)            #输出层
    def forward(self,x):
        x=self.fc1(x)
        x=t.relu(x)
        x=self.fc2(x)
        return x
#训练网络
def train(n_epoches,model,lossfn,opt,tu,tc):
    for epoch in range(n_epoches):
        opt.zero_grad()
        output=model(tu)
        loss=lossfn(output,tc)
        loss.backward()
        opt.step()
        if epoch%500==0:
            print("Epoch: %f ,Loss: %f" % (epoch,loss.item()))
```

In [ ]:
```python
model=Net()
epoches=10000   #迭代10000次
learning_rate=0.001
optimizer=optim.SGD(model.parameters(),lr=learning_rate)
loss_fn=nn.CrossEntropyLoss()
```

In [ ]:
```python
correct=0
total=0
with t.no_grad():
    for i in range(len(vu)):
        output=model(vu[i])
        _,predicted=t.max(output.data,0)
        total+=1
        if predicted==t.argmax(vc[i]):
            correct+=1
print("训练前的准确率：")
print('Accuracy: %f %%' % (float(correct)/total*100))
print('总数: %f ' % (total))
print('正确数: %f ' % (correct))
```

```
训练前的准确率：
Accuracy: 22.000000 %
总数: 50.000000
正确数: 11.000000
```

In [ ]:
```
train(epoches,model,loss_fn,optimizer,tu,tc)
```

```
Epoch: 0.000000 ,Loss: 1.346140
Epoch: 500.000000 ,Loss: 0.923510
Epoch: 1000.000000 ,Loss: 0.784781
Epoch: 1500.000000 ,Loss: 0.673342
Epoch: 2000.000000 ,Loss: 0.589979
Epoch: 2500.000000 ,Loss: 0.533683
Epoch: 3000.000000 ,Loss: 0.496555
Epoch: 3500.000000 ,Loss: 0.471083
Epoch: 4000.000000 ,Loss: 0.452474
Epoch: 4500.000000 ,Loss: 0.437926
Epoch: 5000.000000 ,Loss: 0.425814
Epoch: 5500.000000 ,Loss: 0.415169
Epoch: 6000.000000 ,Loss: 0.405402
Epoch: 6500.000000 ,Loss: 0.396078
Epoch: 7000.000000 ,Loss: 0.386378
Epoch: 7500.000000 ,Loss: 0.374551
Epoch: 8000.000000 ,Loss: 0.358958
Epoch: 8500.000000 ,Loss: 0.339252
Epoch: 9000.000000 ,Loss: 0.319474
Epoch: 9500.000000 ,Loss: 0.301298
```

In [ ]:
```
correct=0
total=0
with t.no_grad():
    for i in range(len(vu)):
        output=model(vu[i])
        _,predicted=t.max(output.data,0)
        total+=1
        if predicted==t.argmax(vc[i]):
            correct+=1
print("训练后的准确率：")
print('Accuracy: %f %%' % (float(correct)/total*100))
print('总数: %f ' % (total))
print('正确数: %f ' % (correct))
#输出测试集的准确率，总体来说还是比较高的,应该没有出现过拟合的情况
#（盲猜因为神经网络太简单，模型的规模、参数比较小，记忆推理远远小于特征推理 ）
```

```
训练后的准确率：
Accuracy: 98.000000 %
总数: 50.000000
正确数: 49.000000
```