用PyTorch实现多层网络

1.引入模块，读取数据

2.构建计算图（构建网络模型）

3.损失函数与优化器

4.开始训练模型

5.对训练的模型预测结果进行评估

1.引入模块，读取数据：

import torch

import numpy as np

from torch import nn

from torch.autograd import Variable

import torch.nn.functional as F

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

%matplotlib inline

data = pd.read\_csv('test\_data.csv')

2.数据预处理：

data1=data.copy()

y=data1.loc[:,['Outcome']] #数据标签

del data1['Outcome']

x = data1 #数据

x\_train, x\_test,y\_train,y\_test= train\_test\_split(x, y, test\_size=0.3,random\_state=2018) #测试数据集为30%，随机种子2018

ss = StandardScaler()

x\_train = ss.fit\_transform(x\_train)

x\_test = ss.fit\_transform(x\_test)

3.数据转化为Tensor：

x\_train\_tensor=torch.from\_numpy(x\_train)

x\_test\_tensor=torch.from\_numpy(x\_test)

y\_train\_numpy=np.array(y\_train)

y\_train\_tensor=torch.from\_numpy(y\_train\_numpy)

y\_test\_numpy=np.array(y\_test)

y\_test\_tensor=torch.from\_numpy(y\_test\_numpy)

x=x\_train\_tensor.float()

y=y\_train\_tensor.float()

#构建计算图（构建网络模型）

class module\_net(nn.Module):

def \_\_init\_\_(self, num\_input, num\_hidden, num\_output):

super(module\_net, self).\_\_init\_\_()

self.layer1 = nn.Linear(num\_input, num\_hidden)

self.layer2 = nn.Tanh()

self.layer3 = nn.Linear(num\_hidden, num\_hidden)

self.layer4 = nn.Tanh()

self.layer5 = nn.Linear(num\_hidden, num\_hidden)

self.layer6 = nn.Tanh()

self.layer7 = nn.Linear(num\_hidden, num\_output)

def forward(self, x):

x = self.layer1(x)

x = self.layer2(x)

x = self.layer3(x)

x = self.layer4(x)

x = self.layer5(x)

x = self.layer6(x)

x = self.layer7(x)

return x

#损失函数

criterion=nn.BCEWithLogitsLoss()

mo\_net = module\_net(8, 10, 1)

optim = torch.optim.SGD(mo\_net.parameters(), 0.01, momentum=0.9)

四.开始训练模型

Loss\_list = [] #用来装loss值，以便之后画图

Accuracy\_list = [] #用来装准确率，以便之后画图

for e in range(10000):

out = mo\_net.forward(Variable(x)) #这里省略了 mo\_net.forward()

loss = criterion(out, Variable(y))

Loss\_list.append(loss.data[0])

#--------------------用于求准确率-------------------------#

out\_class=(out[:]>0).float() #将out矩阵中大于0的转化为1，小于0的转化为0，存入a中

right\_num=torch.sum(y==out\_class).float() #分类对的数值

precision=right\_num/out.shape[0] #准确率

#--------------------求准确率结束-------------------------#

Accuracy\_list.append(precision)

optim.zero\_grad()

loss.backward()

optim.step()

if (e + 1) % 1000 == 0:

print('epoch: {}, loss: {}，precision{},right\_num{}'.format(e+1, loss.data[0],precision,right\_num))

plt.plot(x1, Loss\_list,c='red',label='loss')

plt.plot(x1, Accuracy\_list,c='blue',label='precision')

plt.legend()

五.模型评估

x\_test\_tensor=x\_test\_tensor.float()

y\_test\_tensor=y\_test\_tensor.float()

out\_test=mo\_net.forward(Variable(x\_test\_tensor))

loss\_test = criterion(out\_test, Variable(y\_test\_tensor))

out\_test\_class=(out\_test[:]>0).float() #将out矩阵中大于0的转化为1，小于0的转化为0，存入a中

right\_num\_test=torch.sum(y\_test\_tensor==out\_test\_class).float() #分类对的数值

precision\_test=right\_num\_test/out\_test.shape[0] #准确率

loss\_test=loss\_test.data[0]

print('loss\_test:{},precision\_test:{},right\_num\_test:{}'.format(loss\_test,precision\_test,right\_num\_test))