多分类问题 softmax classifier

笔记本:	pytorch		
创建时间:	2023/3/29 9:54	**更新时间:**	2023/3/29 15:49
作者:	22qdnlbn		
URL:	https://blog.csdn.net/Jeremy_lf/article/details/102725285?spm=1001.2101.3001....		

1、softmax函数
用来解决多分类问题，将我们的所有输出限制在0-1内，且概率之和等于1，不同类别的概率之间有抑制作用。
softmax函数如下：

一篇很好的文章：https://zhuanlan.zhihu.com/p/105722023
https://blog.csdn.net/qian99/article/details/78046329
由于softmax函数的特性，其与z1,z2,z3都有关。
但比如y1
y1对z1,与y1对z2，z3的导数也不同
因此，对softmax进行求导时，要分为2种情况。即i=j与i≠j

2、softmax的损失函数，我们使用交叉熵损失函数
交叉熵损失函数一般用来表示两种分布的差异。

$$L = -\sum_{c=1}^{C} y_c \, log(p_c)$$

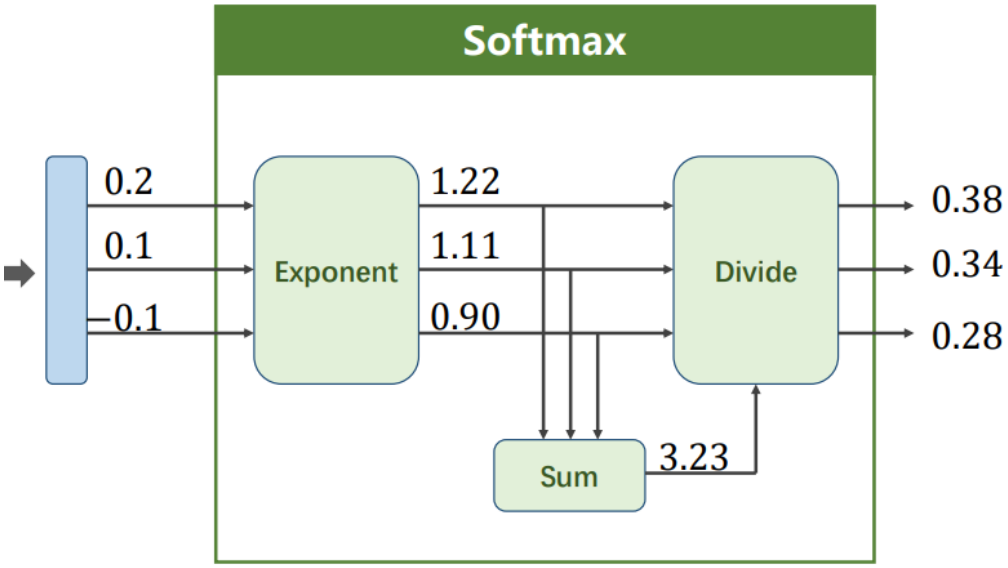
对于分类问题种我们使用的是独热编码，即只有正确的标签是1，其他是0

因此将交叉熵的偏导数进一步简化为：

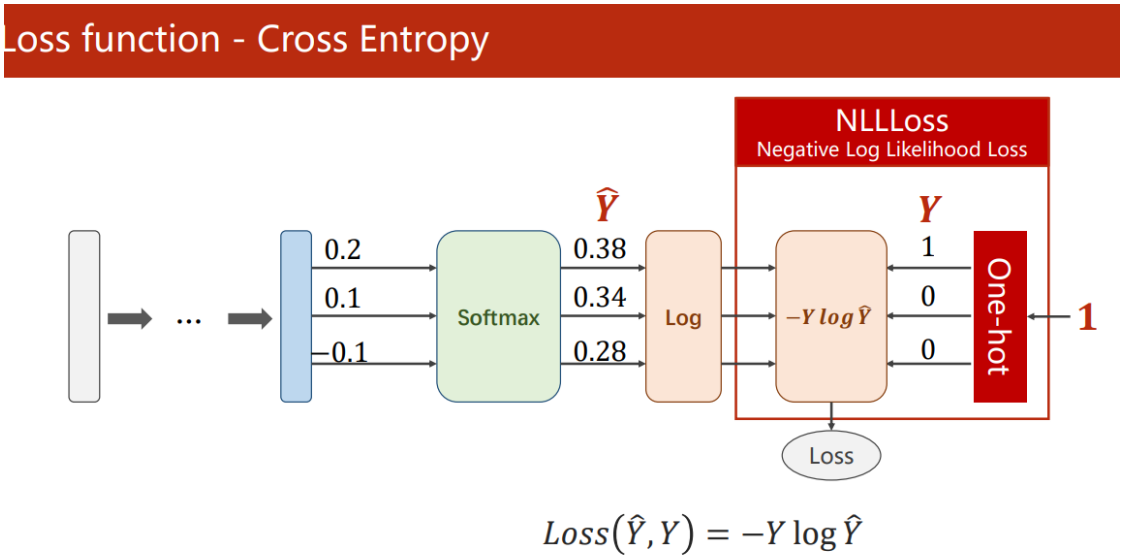
$$\frac{\partial L}{\partial z_i} = p_i - y_i$$

3、多分类问题中的损失函数 CrossEntropyloss=log_softmax+NLLloss

softmax层：



NLLLOSS:



损失函数：等于对softmax求得的结果进行一个element-wise的对数计算+nllloss损失函数

nllloss损失函数： nn.NLLLoss输入是一个对数概率向量和一个目标标签，它与nn.CrossEntropyLoss的关系可以描述为：
softmax(x)+log(x)+nn.NLLLoss====>nn.CrossEntropyLoss

4、维度不同

在opencv pil中图像读取都是w*h*c
而在pytorch中是c*w*h

5、代码

```
# 时间：2023/3/29 11:07
# cky
import numpy as np
import matplotlib.pyplot as plt
```

```
import torch.nn
from torch.utils.data import DataLoader
from torchvision import transforms
from torchvision import datasets
import torch.nn as nn


#parpare datasets
batch_size=64
transform=transforms.Compose([
    transforms.ToTensor(),#将pil图像转化为tensor类型 whc-》cwh
    transforms.Normalize((0.1307,),(0.3801,)) #归一化，平均值，方差
])
train_data=datasets.MNIST(root='../dataset/minst',train=True,download=True,transform=transform)
test_data=datasets.MNIST(root='../dataset/minst',train=False,download=True,transform=transform)
test_len=len(test_data)  #保存一下我们的测试数据集样本数，在测试时我们需要用它来求准确
率
train_load=DataLoader(train_data,batch_size=batch_size,shuffle=True)
test_load=DataLoader(test_data,batch_size=batch_size,shuffle=False)
#design model using class
class Softclass(nn.Module):
    def __init__(self):
        super(Softclass, self).__init__()
        self.linear1=nn.Linear(784,512)
        self.linear2=nn.Linear(512,256)
        self.linear3=nn.Linear(256,128)
        self.linear4=nn.Linear(128,64)
        self.linear5 = nn.Linear(64, 10)
        self.relu=nn.ReLU6()
    def forward(self,x):
        x=x.view(-1,784)
        x=self.relu(self.linear1(x))
        x = self.relu(self.linear2(x))
        x = self.relu(self.linear3(x))
        x=self.relu(self.linear4(x))
        return self.linear5(x)  #最后一层不需要激活函数，因为在交叉熵损失函数中包含了
softmax+log+nllloss
#construct loss and optim
model=Softclass()
loss=nn.CrossEntropyLoss()
optim=torch.optim.SGD(params=model.parameters(),lr=0.1,momentum=0.5)


#training cycle and test
#将训练和测试分别定义为一个函数
def train(epoch):
    loss_sum=0
    for i,data in enumerate(train_load,1):
        inputs,outputs=data
        y_pre=model(inputs)
        cost=loss(y_pre,outputs)
        optim.zero_grad()
        cost.backward()
        optim.step()
        loss_sum+=cost.item()
        if i%300==0:  #不需要每次迭代都输出，在这里我们让每300次迭代输出一次
            print('epoch:',epoch,'i:',i,'loss:',loss_sum/300)
def test():  #测试函数
    correct=0
    with torch.no_grad():  #在测试中不需要计算梯度，这样可以不用保留一些为了计算梯度
的值，可以节省内存空间
        for data in test_load:
            images, labels = data
            y_pre=model(images)
            _,predicts=torch.max(y_pre,dim=1)  #找出维度1 即一行中的哪一列值最大，返
回最大值和相对应的索引
            correct+=(predicts==labels).sum().item()
    print('Accuracy on test data: %d %%' % (100*(correct/test_len)))
if __name__ == '__main__':
    for i in range(5):
```

```
        train(i)
        test()
```