PYTORCH学习笔记

1. x=torch.Tensor(2,2,3),传入所需的张量的维度即可创建,也接受 数组
2. torch.zeros torch.ones torch.rand torch.randn
3. tensor.numpy() torch.from\_numpy(array)
4. tensor1.add\_(tensor2) 就地操作 注意add之后的\_

tensor1+tensor2

1. tensor.view(2,3) 改变张量维度为（2，3）
2. tensor.permute(0,1) 交换张良的 0维 和 1维
3. torch.matmul() torch.mm()
4. 索引方式与numpy类似

X[:,-1] 索引第一列全部元素

X[[0,3],4]

1. X=x.to(device) 函数to 将张量推送到指定设备
2. 神经网络训练的完整流程

网络搭建、数据集搭建、优化

1.从数据加载器中获取一批

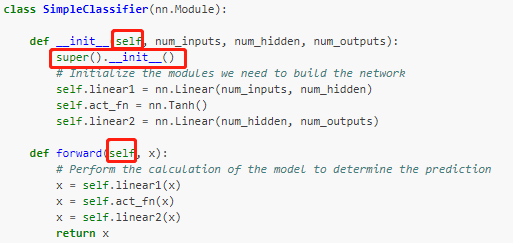
2.从模型中获取批次的预测

3.根据预测和标签之间的差异计算损失

4.反向传播：计算每个参数相对于损失的梯度

5.在梯度方向更新模型的参数

1. 一个简单的神经网络



包含 init以及forward

当需要继承父类构造函数中的内容，且子类需要在父类的基础上补充时，使用super().\_\_init\_\_()方法。

1. 数据集类 torch.untils.data as data

Data提供了两个类

data.Dataset 自定义数据的父类

例：class XORDataset(data.Dataset):

Def \_\_init\_\_(self):

Super().\_\_init\_\_()

Data.DataLoader 数据加载器类

例：data\_loader=data.DataLoader(XORDataset,bitch\_size,shuffle)

data\_inputs, data\_labels = next(iter(data\_loader))

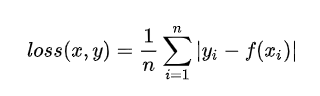
1. 自定义数据集 一般包含三部分 len getitem 和init
2. 损失函数 调用方法

Loss\_module=nn.BCEWithlogitsLoss()

Loss=Loss\_module(predict,true\_lable)

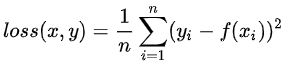
Pytorch提供了很多损失函数

nn.L1loss:它是把目标值与模型输出（估计值）做差的绝对值得到的误差

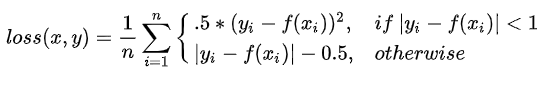


又称为MAE

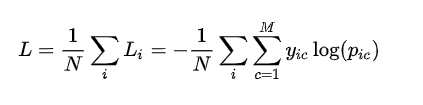
nn.MSELoss:它是把目标值 与模型输出（估计值）做差然后平方得到的误差



nn.SmoothL1Loss:结合了L1和L2



nn.CrossEntropyLoss:交叉熵损失函数 用于多酚类和二分类（代码内还包含softmax）



代码复现：（nn.NLLLoss负对数偶然）

def cross\_entropy\_error(y, t):

if y.ndim == 1:

t = t.reshape(1, t.size)

y = y.reshape(1, y.size)

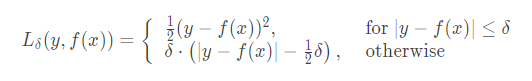
if t.size == y.size:

t = t.argmax(axis=1)

batch\_size = y.shape[0]

return -np.sum(np.log(y[np.arange(batch\_size), t] + 1e-7)) / batch\_size

nn.HuberLoss:同样介于（区分与SmoothL1Loss差别）

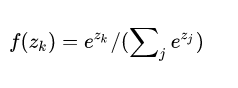


nn.BCEWithLogitsLoss与nn.BCELoss之前的区别：整合了Sigmoid

Hinge Loss (折页损失函数、铰链损失函数)

1. Softmax函数的作用是 将预测值变为概率在0-1之间且和为1

又称为 softmax 归一化指数函数



1. 随机梯度下降

优化器创建 torch.optim.SGD(model.parameters(),lr=0.1)

优化器提供了两个有用的功能：optimizer.step()和optimizer.zero\_grad()

请记住optimizer.zero\_grad()在计算批次的梯度之前调用

θn=θn-1-lr\*梯度偏导

优化器种类：

SGD：随机梯度下降 每一次迭代计算mini-batch的梯度，然后对参数进行更新

Mountain：对梯度偏导 运用 指数加权平均 考虑之前的若干梯度

Agagrad：更新过程中，逐渐减小lr的值 方法：除以前t个梯度平方和

Agadelta：更新过程中，逐渐减小lr得只 方法：指数加权平均

RMSprop：RMSprop 采用均方根（指数加权平均）作为分母，可缓解 Adagrad 学习率下降较快的问题

Adam：本质上也是对lr的下降方法进行调整 一阶矩估计/二阶矩估计开根号（一阶矩阵：指数加权平均 二阶矩阵：均方根）

1. Model.train()和model.eval之间切换是有由于 有些 层 在训练和测试 时前向的步骤不同
2. 引入一个新的知识点 for i in tqdm(range(整数))

可谓程序加载一个进度条 有意思



1. 开始训练

For data\_input,data\_label in dataloader:

上传数据到device

Preds=model(data\_input)#传出来的维度需要降维

Preds=preds.squeeze(dim=1)#我还没理解通透

Loss=Loss\_module(preds,data\_label)

Optimizer.zero\_grad()

Loss.backward()

Optimizer.step()

最好将这个部分 封装到一个函数中

1. 模型保存

state\_dict = model.state\_dict()#提取模型参数

torch.save(state\_dict, "our\_model.tar")#保存模型参数

state\_dict=torch.load( "our\_model.tar")

new\_model.load\_state\_dict(state\_dict)

1. 模型可视化

from torch.utils.tensorboard import SummaryWriter

writer=SummaryWriter(logging\_dir)

writer.add\_figure('predictions',fig,global\_step = epoch + 1)

Torch.where(a>0,a,b)

满足条件返回a,不满足条件返回b

C++： a>0?a:bshuo

Pytorch 加快训练速度的方法

1. num\_works 调节DataLoader中 num\_works的数量

允许并行 加载数据

1. 调高batch\_size到Gpu支持的最大
2. 使用自动混合精度 16字节精度

16bit精度是将内存占用减半的惊人技术。大多数模型使用32bit精度数字进行训练。然而，最近的研究发现，16bit模型也可以工作得很好。混合精度意味着对某些内容使用16bit，但将权重等内容保持在32bit。

1. 使用梯度累加

增加batch大小的另一种方法是在调用optimizer.step()之前，在多个.backward()中累积梯度。

1. 注意CPU和GPU之间频繁的数据传输
2. 开启cudNN benchmarking

损失函数章节

1.nn.NLLLoss()损失函数

输入input input=tensor.randn(3,3)(nn.LogSoftax 函数)

输入target target=tensor.Tensor([[0,0,1],[0,1,0],[0,0,1]])

target=target.argmax(dim=1) #返回每行最大值的索引

输出output -torch.mean(input[torch.arange(3),target])

注意：NLL损失函数的 （1）输入值先经过 softmax 做归一化

（2）在经过取对数之后 输入

2.nn.CrossEntropyLoss()交叉熵损失函数

内部整合了 softmax函数 log函数 和Nllloss函数

3.nn.Softmax(dim=1)

内部执行过程 先取指数

求每行的和

Unsqueeze（dim=1）

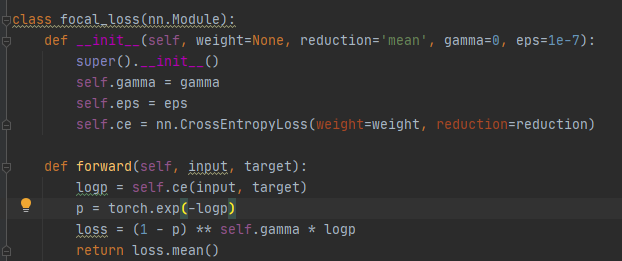
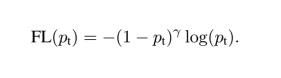
做商

4.nn.L1loss() torch.abs()

5.nn.MSEloss() torch.pow()

6.重点来了 Focal loss

公式（1-p）\*\*γ \* log（p）



1. 激活函数

在Pytorch中的基类 继承nn.Module 以字典形式储存参数

class ActivationFunction(nn.Module):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.name = self.\_\_class\_\_.\_\_name\_\_

self.config = {"name": self.name}

Class Sigmoid(nn.Module):

Def forward(x):

Return 1/(1+torch.exp(-x))

Class tanh(nn.Module):

Def forward(x):

Return

(torch.exp(x)-torch.exp(-x))/(torch.exp(x)+torch.exp(-x))

Torch.where 的作用 与 ？：一样