## 1 L1Loss（绝对值损失函数）

**1.1 CLASS torch.nn.L1Loss(reduction='mean')**

**reduction：**'none' | 'mean' | 'sum'。'**none**'：返回（batch，每一个样本的loss）， '**mean**'：返回整个batch的loss和除以batch的值，'**sum**'：返回整个batch的loss和。

**1.2 L1LOSS 公式**

x和y可以是任意形状，包含n个元素。

**1.3 L1Loss 输入输出**

**import** torch

loss\_fun **=** torch**.**nn**.**L1Loss()

input **=** torch**.**randn(2, 2, requires\_grad**=True**)

target **=** torch**.**randn(2, 2)

output **=** loss\_fun(input, target)

print(input)

print(target)

print(output)

*#0.7826 =(|-0.0621-(-0.8778)| + |0.5715-(-0.8386)| + |0.0940-(-0.3857)| + |-1.7804-(-1.3557)|)/4*

输出：

tensor([[**-**0.0621, 0.5715],

[ 0.0940, **-**1.7804]], requires\_grad**=True**)

tensor([[**-**0.8778, **-**0.8386],

[**-**0.3857, **-**1.3557]])

tensor(0.7826, grad\_fn**=<**L1LossBackward**>**)

## 2 CrossEntropyLoss（交叉熵损失函数）

它主要刻画的是实际输出（概率）与期望输出（概率）的距离，也就是交叉熵的值越小，两个概率分布就越接近。  
原始：  
优化后：

**2.1 CLASS torch.nn.CrossEntropyLoss(weight=None,ignore\_index=-100, reduction='mean')**

* **weight(tensor)**: 1-D tensor，n个元素，分别代表n类的权重，如果你的训练样本很不均衡的话，是非常有用的。默认值为None。
* **ignore\_index(int）：**用于忽略ground-truth中某些不需要参与计算的类。如忽略类别为1，令ignore\_index=1。
* **reduction：**'none' | 'mean' | 'sum'。'**none**'：返回（batch，每一个样本的loss）， '**mean**'：返回整个batch的loss和除以batch的值，'**sum**'：返回整个batch的loss和。

**2.2 CrossEntropyLoss 公式**

CrossEntropyLoss = softmax + NLLLoss

**weight 非None时：**

**2.3 CrossEntropyLoss输入输出**

import torch

loss\_fun = torch.nn.CrossEntropyLoss()

input = torch.randn(3, 5, requires\_grad=True)

target = torch.empty(3, dtype=torch.long).random\_(5)

output = loss\_fun(input, target)

output.backward()

print(input)

print(target)

print(output)

输出：

tensor([[-0.2730, -0.8026, 1.0645, -0.4534, 1.2991],

[ 0.1100, -0.1885, -0.1602, 0.1714, 0.9912],

[ 0.3633, 0.6047, 0.7828, 0.4092, 1.0347]], requires\_grad=True)

tensor([4, 1, 0])

tensor(1.6114, grad\_fn=<NllLossBackward>)

## 3 DiceLoss（用于计算两个样本点的相似度的距，主要应用，语义分割等）

**Dice：**集合相似度度量函数，用于计算两个样本点的相似度（值范围为[0, 1], Dice等价于F1-Score）

**Dice Loss：**计算两个样本的误差。越小，两个样本越相似。

|X|表示X元素的个数。

**def** **dice\_loss**(input, target):

input **=** input**.**contiguous()**.**view(input**.**size()[0], **-**1)

target **=** target**.**contiguous()**.**view(target**.**size()[0], **-**1)**.**float()

a **=** torch**.**sum(input **\*** target, 1) *# |X⋂Y|*

b **=** torch**.**sum(input **\*** input, 1) **+** 0.001 *# |X|*

c **=** torch**.**sum(target **\*** target, 1) **+** 0.001 *# |Y|*

d **=** (2 **\*** a) **/** (b **+** c)

**return** 1**-**d

## 4 Focal Loss

**思想：是模型更加关注不确定、模凌两可的样，用分段函数不能求导。**

Focal Loss 公式1:

就是对应分类正确的预测值, 交叉熵损失。

Focal Loss 公式2:

一般设置为0.25。

**代码：**

import torch

import torch.nn.functional as F

def focal\_loss(logits, labels, gamma=2, reduction="mean"):

ce\_loss = F.cross\_entropy(logits, labels, reduction="none")

log\_pt = -ce\_loss

pt = torch.exp(log\_pt)

weights = (1 - pt) \*\* gamma

fl = weights \* ce\_loss

if reduction == "sum":

fl = fl.sum()

elif reduction == "mean":

fl = fl.mean()

else:

raise ValueError(f"reduction '{reduction}' is not valid")

return fl

def balanced\_focal\_loss(logits, labels, alpha=0.25, gamma=2, reduction="mean"):

# alpha 在多分类中每个类别的权重。一般设置 0.25。

return alpha \* focal\_loss(logits, labels, gamma, reduction)

if \_\_name\_\_ == "\_\_main\_\_":

logits = torch.tensor([[0.3, 0.6, 0.9, 1], [0.6, 0.4, 0.9, 0.5]])

labels = torch.tensor([1, 3])

print(focal\_loss(logits, labels))

print(focal\_loss(logits, labels, reduction="sum"))

print(balanced\_focal\_loss(logits, labels))