Pytorch中常见的损失函数的原理

本文介绍在pytorch中常用的几种损失函数的原理及代码实现，以方便后续自查，但本文未对其中的数学原理进行详细介绍，后续有需要可再进行补充。

1. MSE Loss

均方损失函数，适用于回归任务。一般损失函数都是计算一个 batch 数据总的损失，而不是计算单个样本的损失。

计算公式：

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = (x_n - y_n)^2$$

其中x是Input，y表示target

参数说明：

```
CLASS torch.nn.MSELoss(size_average=None, reduce=None, reduction='mean')
```

reduce与size_average：

reduce = False，损失函数返回的是向量形式的 loss，这种情况下参数 size_average 失效

reduce = True， 损失函数返回的是标量形式的 loss，这种情况下:
1）当 size_average = True 时，返回 loss.mean()，即所有向量元素求和后再除以向量长度
2）当 size_average = False 时，返回 loss.sum()，即所有向量元素只求和

reduction：默认是mean，还可以是sum

代码示例：

```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

```
# 1.调用MSELoss
mse_loss_fn = torch.nn.MSELoss()   # 默认输出标量并求均值
input = torch.randn(2, 4, requires_grad=True)
target = torch.randn(2, 4)
output =  mse_loss_fn(input, target)
print(output.item())
```

```
2.2671520709991455
```

2. BCE Loss

只适用与二分类任务，且神经网络的输出是一个概率分布，一般输出层的激活函数是 Sigmod 函数（返回一个概率数值，那么可以理解为某一类的概率），因为只有两类，所以输出没必要归一化，直接就是一个概率分布。

计算公式：

一个batch数据的损失为：

$$L_{batch} = -w \cdot [y \ln \hat{y} + (1-y) \ln(1-\hat{y})]$$

一个样本的损失为：

$$L_{\text{one}} = -[y \ln \hat{y} + (1-y) \ln(1-\hat{y})]$$

参数说明：

```
CLASS torch.nn.BCELoss(weight=None, size_average=None, reduce=None,
reduction='mean')
```

weight：weight 必须和 target 的 shape 一致

reduce与size_average：

reduce = False，损失函数返回的是向量形式的 loss，这种情况下参数 size_average 失效

reduce = True， 损失函数返回的是标量形式的 loss，这种情况下:
1）当 size_average = True 时，返回 loss.mean()，即所有向量元素求和后再除以向量长度
2）当 size_average = False 时，返回 loss.sum()，即所有向量元素只求和

reduction：默认是mean，还可以是sum

输入值与目标值说明：

input：可以是任意维度，保证是log的一个概率

target：与input一致，默认情况是线性的概率

outputz：一般是一个张量

代码示例：

```
# logits shape：[BS,NC] Logits可以理解为输出的一个概率分布
# 定义一些通用的
batch_size = 2
num_class = 4

logits = torch.randn(batch_size,num_class) # 未归一化分数，作为损失函数的输入

target_indices = torch.randint(num_class,size = (batch_size,)) # 形成的是
[0,num_class-1]范围的索引的目标分布
target_logits = torch.randn(batch_size,num_class) # 形成的是目标概率分布
```

```
## 2. 调用Binary Cross Entropy loss （BCE Loss）
bce_loss_fn = torch.nn.BCELoss()
logits = torch.randn(batch_size) # 这是logits
prob_1 = torch.sigmoid(logits) # 这是概率
# 如果神经网络输出的是logits，那么就用BCEwithlogitsloss这个方法
target = torch.randint(2,size=(batch_size,))
bce_loss = bce_loss_fn(prob_1,target.float())
print(f"binary cross entropy loss：{bce_loss}")

### 用NLLLoss（本文后面会讲到）代替BCE LOSS做二分类
prob_0 = 1 - prob_1.unsqueeze(-1) # 但是要进行扩充
```

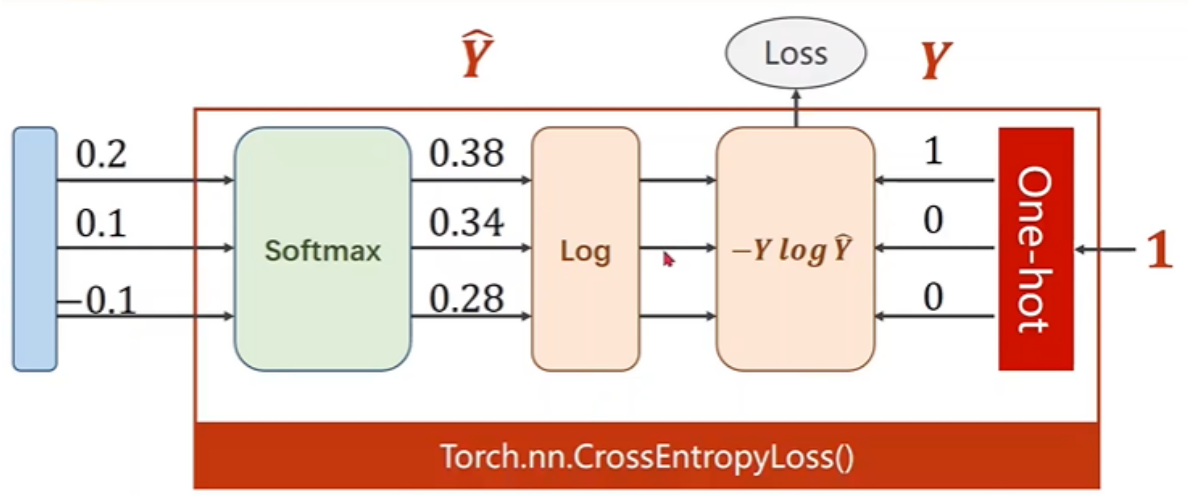
```
prob = torch.cat([prob_0,prob_1.unsqueeze(-1)],dim=-1) # [BS,2]
nll_loss_binary = nll_fn(torch.log(prob),target)
print(f"negative likelihood loss binary is: {nll_loss_binary}")
```

```
binary cross entropy loss: 0.4194680452346802
negative likelihood loss binary is: 0.4194680452346802
```

3. CrossEntropyLoss

用于分类任务中，常用于多分类

当使用 CrossEntropyLoss 损失函数的时候，神经网络的输出就不用再接 softmax 层了，因为这个损失函数内部会做这个归一化，同时它还会根据对应的输出标签 y 生成 one-hot向量。如下图所示：



交叉熵计算公式：

$$H(p,q) = -\sum_{i=1}^{N} p(x_i) \ln q(x_i)$$

其中p表示真实分布，q表示预测分布

参数说明：

```
CLASS torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=-
100, reduce=None, reduction='mean', label_smoothing=0.0
```

weight：分类类别明显不均衡的话，会设置weight来使得更均衡

ignore_index：类似padding的操作，一般设置ingnore_index = 0，传入之后，如果目标为ingnore_index，就不会考虑那个位置

reduction：默认是mean，还可以是sum

label_smoothing：把目标的概率值降低点，相当于平滑化的操作

输入值与目标值说明：

input：输入（在模型中，也就是模型的输出值，放入损失中作为输入）是**未归一化**的一个分数。(*N*,*C*) or (C)，where C = number of classes， or (N, C, d_1, d_2, ..., d_K)(*N*,*C*,*d*1,*d*2,...,*d***K*) with K≥1 in the case of K-dimensional loss.（这些d可理解为时空维度，如d1为视频高度；d2为视频宽度；d3为视频通道数目；d4为视频时间维度）

target：传入的是类别标签，如果input是(C)的话，那么target形状就是()；如果input是(N,C)的话，那么target形状就是(N)；如果input是(N, C, d_1, d_2, ..., d_K)的话，那么target形状就是(N, d_1, d_2, ..., d_K)。总之就是少了C这一维，每一个值范围是[0,C)

output：If reduction is none，那么就与targe一致，反之就是标量

代码示例：

```
## 3. 调用Cross Entropy loss (CE Loss)
### method1 for CE Loss
ce_loss_fn = torch.nn.CrossEntropyLoss()
ce_loss1 = ce_loss_fn(logits,target_indices)
print(f"cross entropy loss is {ce_loss1}")

### method2 for CE Loss
ce_loss2 = ce_loss_fn(logits,torch.softmax(target_logits,dim=-1))
print(f"cross entropy loss is {ce_loss2}")
```

```
cross entropy loss1 is 1.4498498439788818
cross entropy loss2 is 1.7070740461349487
```

4. NLLLoss（负对数似然Loss）

交叉熵就是负对数似然

```
CLASS torch.nn.NLLLoss(weight=None, size_average=None, ignore_index=- 100,
reduce=None, reduction='mean')
```

参数说明：

weight：分类类别明显不均衡的话，会设置weight来使得更均衡

ignore_index：类似padding的操作，一般设置ingnore_index = 0，传入之后，如果目标为ingnore_index，就不会考虑那个位置

reduction：默认是mean，还可以是sum

label_smoothing：把目标的概率值降低点，相当于平滑化的操作

输入值与目标值说明：

input：输入（在模型中，也就是模型的输出值，放入损失中作为输入）必须是每个类别的对数概率，并且是做了对数化的。(*N*,*C*) or (C)，where C = number of classes， or (N, C, d_1, d_2, ..., d_K) with K≥1 in the case of K-dimensional loss.

target：是一个索引，范围是[0,C-1]，其中C为类别的数目。形状是 (N) or 一个标量 or (N, C, d_1, d_2, ..., d_K) with K≥1 in the case of K-dimensional loss.

ouput：If reduction is none, shape (N) or (N, C, d_1, d_2, ..., d_K) with K≥1 in the case of K-dimensional loss.否则就是标量。

代码示例：

```
## 4. 调用Negative Log Likelihood loss (NLL Loss)
nll_fn = torch.nn.NLLLoss()
nll_loss = nll_fn(torch.log(torch.softmax(logits,dim=-1)),target_indices)
print(f"negative log likelihood loss is {nll_loss}")
## 重要结论：cross entropy value = NLL value
```

```
negative log likelihood loss is 1.4498497247695923
```

5. KL散度 Loss

KL散度计算公式：

$$D_{KL}(p\|q) = H(p,q) - H(p)$$

参数说明：

```
CLASS torch.nn.KLDivLoss(size_average=None, reduce=None, reduction='mean',
log_target=False)
```

reduce与size_average：

reduce = False，损失函数返回的是向量形式的 loss，这种情况下参数 size_average 失效

reduce = True， 损失函数返回的是标量形式的 loss，这种情况下:
1）当 size_average = True 时，返回 loss.mean()，即所有向量元素求和后再除以向量长度
2）当 size_average = False 时，返回 loss.sum()，即所有向量元素只求和

reduction：默认是mean，还可以是sum

log_target：表示是否对目标值求log

输入值与目标值说明：

input：可以是任意维度，保证是log的一个概率

target：与input一致，默认情况是线性的概率

output：默认是一个标量，如果redution为"none"，那么与输入具有相同形状

代码示例：

```
## 4. 调用Kullback-Leibler divergence loss（KL Loss）
kid_loss_fn = torch.nn.KLDivLoss()
kid_loss =
kid_loss_fn(torch.softmax(logits,dim=-1),torch.softmax(target_logits,dim=-1))
print(f"Kullback-Leibler divergence  loss is {kid_loss}")
```

```
Kullback-Leibler divergence  loss is -0.3704858124256134
```

验证CE = IE + KLD(交叉熵 = 信息熵 + KL散度)

```
# 验证的时候就按样本了
ce_loss_fn_sample = torch.nn.CrossEntropyLoss(reduction="none")
ce_loss_sample = ce_loss_fn_sample(logits,torch.softmax(target_logits,dim=-1))
print(f"cross entropy loss sample is {ce_loss_sample}")

kid_loss_fn_sample = torch.nn.KLDivLoss(reduction="none")
kid_loss_sample =
kid_loss_fn_sample(torch.log(torch.softmax(logits,dim=-1)),torch.softmax(target_
logits,dim=-1)).sum(dim=-1)
print(f"Kullback-Leibler divergence loss sample is {kid_loss_sample}")

target_information_entropy =
torch.distributions.Categorical(torch.softmax(target_logits,dim=-1)).entropy()
print(f"information entropy sample is {target_information_entropy}") # IE为常数，
如果目标分布是delta分布，IE=0

print(torch.allclose(ce_loss_sample,kid_loss_sample+target_information_entropy))
```

```
cross entropy loss sample is tensor([1.4442, 1.9700])
Kullback-Leibler divergence loss sample is tensor([0.2811, 0.7337])
information entropy sample is tensor([1.1631, 1.2363])
True
```

6. 余弦相似度 Loss

根据余弦相似度来判断输入的两个数是相似还是不相似的（在对比学习、自监督学习、文本相似度、图片相似度、图片检索中比较常用）

```
CLASS torch.nn.CosineEmbeddingLoss(margin=0.0, size_average=None, reduce=None,
reduction='mean')
```

参数说明：

margin：是一个-1 到 1 ， 0 到 0.5 的数，默认为0

reduce与size_average：

reduce = False，损失函数返回的是向量形式的 loss，这种情况下参数 size_average 失效

reduce = True， 损失函数返回的是标量形式的 loss，这种情况下:
1）当 size_average = True 时，返回 loss.mean()，即所有向量元素求和后再除以向量长度
2）当 size_average = False 时，返回 loss.sum()，即所有向量元素只求和

reduction：默认是mean，还可以是sum

输入值与目标值说明：

input1：(N,D)或(D)，其中N是batch_size，D是embedding dimension

input2：与input1一致

target：(N) 或 () -1或1

output：如果redution为"none"，那么就是(N)，否则就是标量

代码示例：

```
## 6.调用Cosine Similarity loss
cosine_loss_fn = torch.nn.CosineEmbeddingLoss()
v1 = torch.randn(batch_size,512)
v2 = torch.randn(batch_size,512)
target = torch.randint(2,size = (batch_size,))*2 - 1 # torch.randint(2是 0,1 *2 是0,2 -1 就是 -1,1
cosine_loss = cosine_loss_fn(v1,v2,target)
print(f"Cosine Similarity loss is: {cosine_loss}")
```

```
Cosine Similarity loss is: 0.942288875579834
```

参考：

- Pytorch官方文档：https://pytorch.org/
- B站up主deep_thoughts讲解：Pytorch常见损失函数原理及实现
- 博客：Pytorch之损失函数
- KL散度和交叉熵基础：KL散度和交叉熵