一、概述

（1）

通过YOLOv8-训练流程-正负样本分配的介绍，我们可以知道，经过预处理与筛选的过程得到最终的训练数据：

a. 网络输出值：pred_scores[bx8400xcls_num]、pred_bboxes[bx8400x4]

b. 训练标签值：

target_scores[bx8400xcls_num](one-hot类型)，在计算损失时与预测结果pred_scores[bx8400xcls_num]，计算交叉熵损失，该损失是对每个类别计算BCE Loss，因为类别预测采用的sigmoid分类器。

target_bboxes[bx8400x4](target_bboxes需要缩放到特征图尺度，即除以相应的**stride**)，在计算损失时，分别与预测的**pred_bboxes**计算**Ciou** Loss, 同时与**pred_regs**(预测的anchors中心点到各边的距离)计算回归DFL Loss。

c. 训练mask值：fg_mask [bx8400]，对8400个anchor进行正负样本标记，计算损失过程中通过fg_mask筛选正负样本。

二、损失函数

(1) 类别分类损失

在yolov8中，类别损失最终采用的是交叉熵损失，该方法是我们非常熟知的，不再赘述。

代码如下：

```
1 self.bce = nn.BCEWithLogitsLoss(reduction='none')
```

```
1 loss[1] = self.bce(pred_scores,target_scores).sum()/target_scores_sum
```

其中预测pred_scores: b x 8400 x cls_num; target_scores: b x 8400 x cls_num, 相当于对于每个box，其cls_num个分类都视为二分类，并进行交叉熵运算。

(2) 边框回归损失

边框回归，采用的是DFL Loss + CIOU Loss

$$\mathbf{DFL}(\mathcal{S}_i, \mathcal{S}_{i+1}) = -\left((y_{i+1} - y)\log(\mathcal{S}_i) + (y - y_i)\log(\mathcal{S}_{i+1})\right)$$

```
target_bboxes /= self.stride_scales
loss[0],loss[2] = self.bbox_loss(pred_regs,pred_bboxes,self.anc_points,target_b
                                 target_scores,target_scores_sum,fg_mask)
```

DFL loss:

```
weight = torch.masked_select(target_scores.sum(-1), fg_mask).unsqueeze(-1)
# DFL loss
if self.use_dfl:
    target_ltrb = self.bbox2reg(anchor_points, target_bboxes, self.reg_max)
    loss_dfl = self._df_loss(pred_regs[fg_mask].view(-1, self.reg_max + 1), tar
```

----------原理分析----------

提问：采用的是交叉熵运算，为什么是交叉熵运算呢?

因为DFL是以概率的方式对预测box进行回归，如设置超参数reg_max = 16 ,这此时网络该分支的输出通道为64=4*reg_max;

在此之前，会预先设置16个固定的参考值A: [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]其每个位置与reg_max的每个位置是相互对应的。

对这reg_max个数，通过softmax函数进行离散化处理，视为一个16分类，得到相对于每个固定值的概率，既然是概率，那么在计算损失的时候就多采用交叉熵损失。

首先将正样本标签target_bboxes[bx8400x4](其是相对于特征图大小的xmin,ymin,xmax,ymax)，将其转换到左上右下四个边，距离中心点距离，并保证偏移不能超过reg_max=15(**这相当于是16个类别的标签，说明距离中心点的距离不能超过15**)。

```
    def bbox2reg(self,anchor_points, target_bboxes,reg_max):
```

```
2
3         """Transform bbox(xyxy) to dist(ltrb)."""
4         x1y1,x2y2 = target_bboxes[...,:2],target_bboxes[...,2:]
5         return torch.cat((anchor_points - x1y1, x2y2 - anchor_points), -1).clam
```

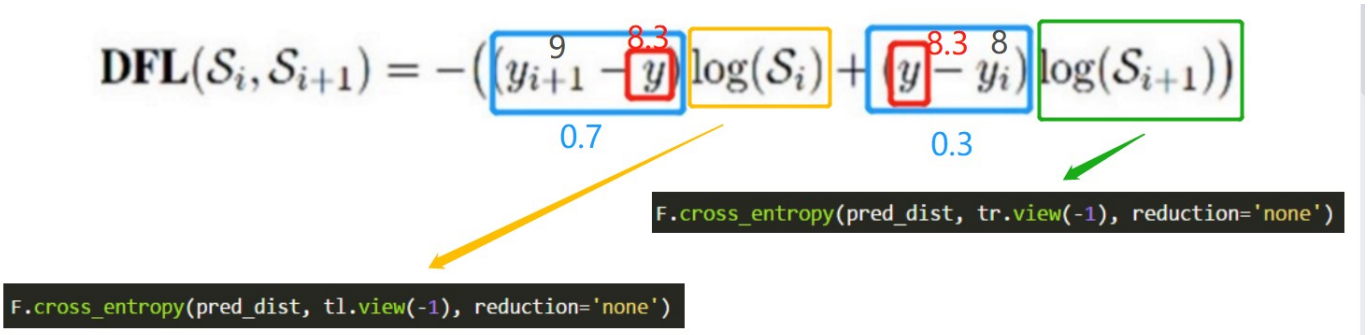
然后，将网络预测值pred_regs[b,8400,4*16]，筛选出正样本部分pred_regs'[K,64],转换为网络输出的分布形式 [Kx4,16],

同时target_bboxes筛选出正样本部分targets_bboxes' [K,4]（每个坐标位置提供一个标签0~15，然后与预测结果）进行DFL计算。

注意！！

上面获得的target_bboxes'其坐标值一般不会落在具体的网格角点上,但是标签又需要为整数，以ximin的预测为例，其真实的值 y = 8.3(**对应公式中的y**), 那么它左侧整数为8(**对应公式中的yi**)，右侧整数为9(**对应公式中的yi+1**), 但是具体用哪个整数来计算又不好说，那就都用!

但是这两个整数，距离真实值的距离又是不同的，所以就给**8**一个大一点的权重**0.7**(9-8.3=0.7，对应于公式中的**yi+1-y**),给**9**一个小一点的权重**0.3**(8.3-8=0.3,对应于公式中的y-yi).



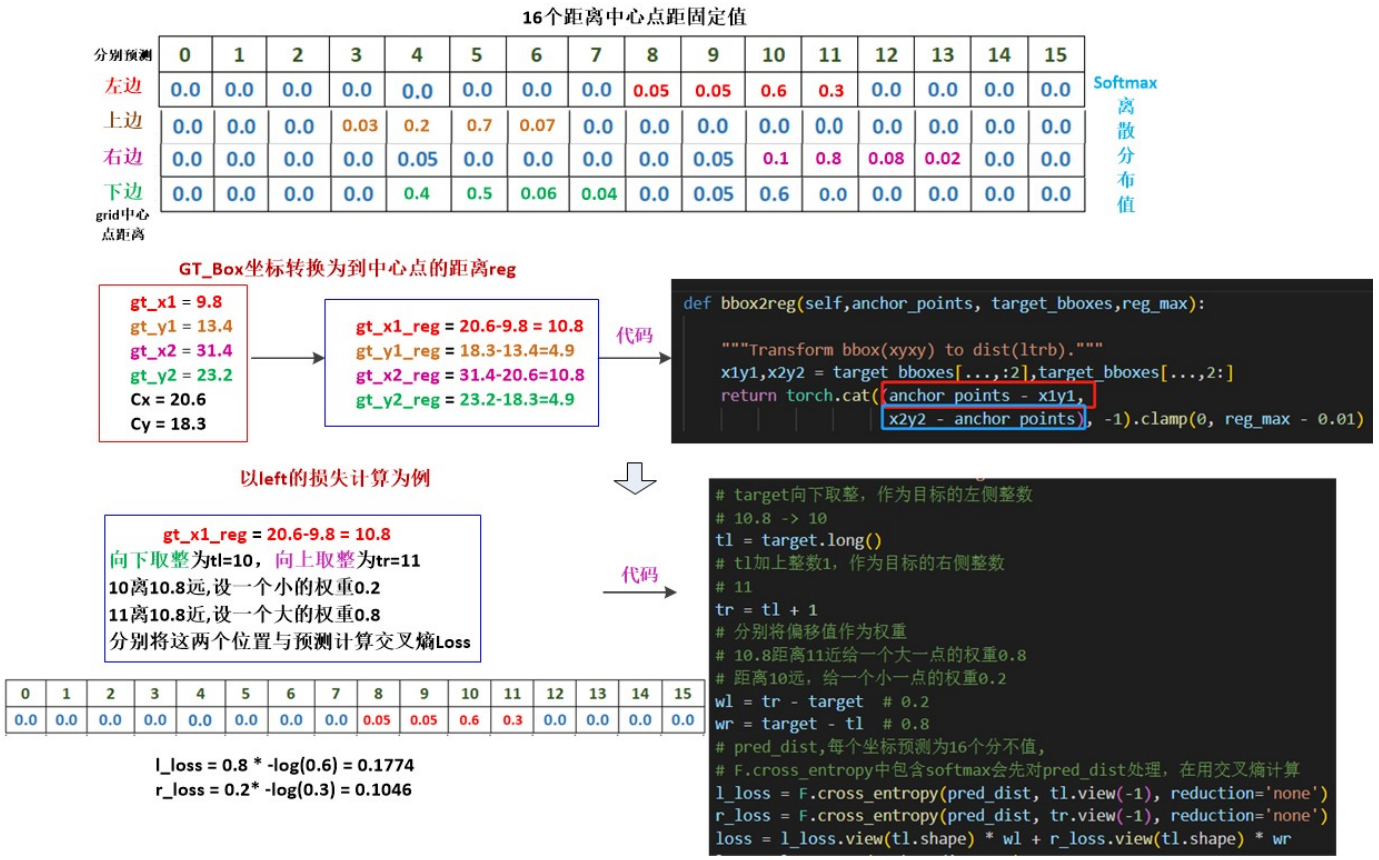

```
1     def _df_loss(pred_dist, target):
2         """Return sum of left and right DFL losses."""
3         # target向下取整，作为目标的左侧整数 K x 4（取值范围0-15）
4         tl = target.long()  # target left  (包括中心点距离左边和上边的距离)
5         # tl加上整数1，作为目标的右侧整数  K x 4（取值范围0-15）
6         tr = tl + 1  # target right (包括中心点距离右边和下边的距离)
7         # 分别将偏移值作为权重
8         # 如真实坐标值为8.3，那么它距离8近给一个大一点的权重0.7，距离9远，给一个小一点
9         wl = tr - target  # weight left  Kx4
10
11        wr = target - tl  # weight right Kx4
12        # 左右目标分别拉直 tl -> K x 4 -> 4K, pred_dist->4K x 16
```

```
        l_loss = F.cross_entropy(pred_dist, tl.view(-1), reduction='none')
        # 左右目标分别拉直 tr -> K x 4 -> 4K
        r_loss = F.cross_entropy(pred_dist, tr.view(-1), reduction='none')
        loss = l_loss.view(tl.shape) * wl + r_loss.view(tl.shape) * wr
        loss = loss.mean(-1,keepdim=True)

        return loss

```

首先pred_dist ,其shape为4K x 16, 其中16是经过softmax处理后得到的离散分布值，而tl和tr经过处理后为 Kx2，这里就相当于在做softmax处理的16个分类，tr和tr是标签(理论上是0-15之间的值)，而pred_dist是16个各自的预测值，经过训练，使预测坐标的结果逐渐向正确的区域逼近。



Ciou Loss:

概括：<u>**边框回归，reg_max个分布值经softmax离散分布，通过巧妙利用交叉熵损失将预测位置迅速聚焦到目标位置附近，然后经过加权求和得到四个预测坐标值，再用Ciou Loss进一步精确坐标位置。**</u>

```
"""IoU loss."""
weight = torch.masked_select(target_scores.sum(-1), fg_mask).unsqueeze(-1)
iou = bbox_iou(pred_bboxes[fg_mask], target_bboxes[fg_mask], xywh=False, CIoU=T
loss_iou = ((1.0 - iou) * weight).sum() / target_scores_sum
```

其中，weights [Kx1]，表示batch中共有K个正样本，故从加权后的target_scores[bx8400xcls_num]通过fg_mask[bx8400],筛选出正样本的K个位置，作为权重因子。

然后计算根据CIOU的计算公式计算Ciou值，并得到ciou loss.