IoU、GIoU、DIoU、CIoU损失函数的那点事儿

一、IOU(Intersection over Union)

1. 特性(优点)

IoU就是我们所说的**交并比**，是目标检测中最常用的指标，在anchor-based的方法中，他的作用不仅用来确定正样本和负样本，还可以用来评价输出框（predict box）和ground-truth的距离。

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$

可以说**它可以反映预测检测框与真实检测框的检测效果。**
还有一个很好的特性就是**尺度不变性**，也就是对尺度不敏感（scale invariant）， 在regression任务中，判断predict box和gt的距离最直接的指标就是IoU。**(满足非负性；同一性；对称性；三角不等性)**

```
import numpy as np
def Iou(box1, box2, wh=False):
    if wh == False:
        xmin1, ymin1, xmax1, ymax1 = box1
        xmin2, ymin2, xmax2, ymax2 = box2
    else:
        xmin1, ymin1 = int(box1[0]-box1[2]/2.0), int(box1[1]-box1[3]/2.0)
        xmax1, ymax1 = int(box1[0]+box1[2]/2.0), int(box1[1]+box1[3]/2.0)
        xmin2, ymin2 = int(box2[0]-box2[2]/2.0), int(box2[1]-box2[3]/2.0)
        xmax2, ymax2 = int(box2[0]+box2[2]/2.0), int(box2[1]+box2[3]/2.0)
    # 获取矩形框交集对应的左上角和右下角的坐标（intersection）
    xx1 = np.max([xmin1, xmin2])
    yy1 = np.max([ymin1, ymin2])
    xx2 = np.min([xmax1, xmax2])
    yy2 = np.min([ymax1, ymax2])
    # 计算两个矩形框面积
    area1 = (xmax1-xmin1) * (ymax1-ymin1)
    area2 = (xmax2-xmin2) * (ymax2-ymin2)
    inter_area = (np.max([0, xx2-xx1])) * (np.max([0, yy2-yy1]))  #计算交集面积
    iou = inter_area / (area1+area2-inter_area+1e-6)    #计算交并比

    return iou
```

2. 作为损失函数会出现的问题(缺点)

如果两个框没有相交，根据定义，IoU=0，不能反映两者的距离大小（重合度）。同时因为loss=0，没有梯度回传，无法进行学习训练。
IoU无法精确的反映两者的重合度大小。如下图所示，三种情况IoU都相等，但看得出来他们的重合度是不一样的，左边的图回归的效果最好，右边的最差。

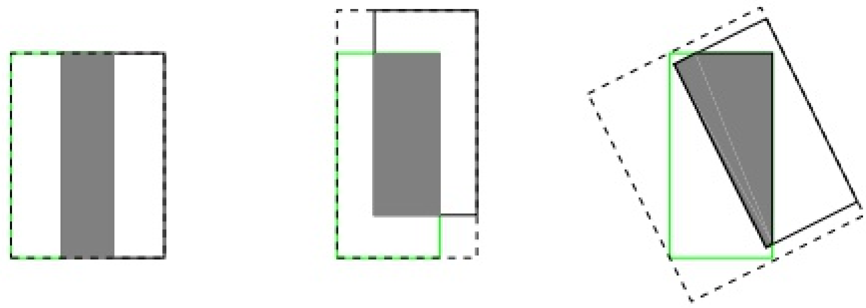


Figure 2. Three different ways of overlap between two rectangles with the exactly same IoU values, *i.e.* $IoU = 0.33$, but different $GIoU$ values, *i.e.* from the left to right $GIoU = 0.33, 0.24$ and -0.1 respectively. $GIoU$ value will be higher for the cases with better aligned orientation.

二、GIOU(Generalized Intersection over Union)

1、来源

在CVPR2019中，论文

《Generalized Intersection over Union: A Metric and A Loss for Bounding Bo...
arxiv.org

的提出了GIoU的思想。由于IoU是**比值**的概念，对目标物体的scale是不敏感的。然而检测任务中的BBox的回归损失(MSE loss, l1-smooth loss等）优化和IoU优化不是完全等价的，而且 Ln 范数对物体的scale也比较敏感，IoU无法直接优化没有重叠的部分。

这篇论文提出可以直接把IoU设为回归的loss。

$$GIoU = IoU - \frac{|A_c - U|}{|A_c|}$$

上面公式的意思是：先计算两个框的最小闭包区域面积 A_c (通俗理解：**同时包含了预测框和真实框**的最小框的面积)，再计算出IoU，再计算闭包区域中不属于两个框的区域占闭包区域的比重，最后用IoU减去这个比重得到GIoU。

附：

generalized-iou/g-darknet
github.com

2、 特性[1]

与IoU相似，GIoU也是一种距离度量，作为损失函数的话， $L_{GIoU}=1-GIoU$,满足损失函数的基本要求
GIoU对scale不敏感
GIoU是IoU的下界，在两个框无限重合的情况下，IoU=GIoU=1
IoU取值[0,1]，但GIoU有对称区间，取值范围[-1,1]。在两者重合的时候取最大值1，在两者无交集且无限远的时候取最小值-1，因此GIoU是一个非常好的距离度量指标。
与IoU只关注重叠区域不同，**GIoU不仅关注重叠区域，还关注其他的非重合区域**，能更好的反映两者的重合度。

Algorithm 2: IoU and $GIoU$ as bounding box losses

input : Predicted B^p and ground truth B^g bounding box coordinates:
$B^p = (x_1^p, y_1^p, x_2^p, y_2^p)$, $\quad B^g = (x_1^g, y_1^g, x_2^g, y_2^g)$.

output: $\mathcal{L}_{IoU}$, $\mathcal{L}_{GIoU}$.

1 For the predicted box B^p, ensuring $x_2^p > x_1^p$ and $y_2^p > y_1^p$:
$\hat{x}_1^p = \min(x_1^p, x_2^p), \quad \hat{x}_2^p = \max(x_1^p, x_2^p),$
$\hat{y}_1^p = \min(y_1^p, y_2^p), \quad \hat{y}_2^p = \max(y_1^p, y_2^p).$

2 Calculating area of B^g: $A^g = (x_2^g - x_1^g) \times (y_2^g - y_1^g)$.

3 Calculating area of B^p: $A^p = (\hat{x}_2^p - \hat{x}_1^p) \times (\hat{y}_2^p - \hat{y}_1^p)$.

4 Calculating intersection $\mathcal{I}$ between B^p and B^g:
$x_1^{\mathcal{I}} = \max(\hat{x}_1^p, x_1^g), \quad x_2^{\mathcal{I}} = \min(\hat{x}_2^p, x_2^g),$
$y_1^{\mathcal{I}} = \max(\hat{y}_1^p, y_1^g), \quad y_2^{\mathcal{I}} = \min(\hat{y}_2^p, y_2^g),$
$$\mathcal{I} = \begin{cases} (x_2^{\mathcal{I}} - x_1^{\mathcal{I}}) \times (y_2^{\mathcal{I}} - y_1^{\mathcal{I}}) & \text{if} \quad x_2^{\mathcal{I}} > x_1^{\mathcal{I}}, y_2^{\mathcal{I}} > y_1^{\mathcal{I}} \\ 0 & \text{otherwise.} \end{cases}$$

5 Finding the coordinate of smallest enclosing box B^c:
$x_1^c = \min(\hat{x}_1^p, x_1^g), \quad x_2^c = \max(\hat{x}_2^p, x_2^g),$
$y_1^c = \min(\hat{y}_1^p, y_1^g), \quad y_2^c = \max(\hat{y}_2^p, y_2^g).$

6 Calculating area of B^c: $A^c = (x_2^c - x_1^c) \times (y_2^c - y_1^c)$.

7 $IoU = \frac{\mathcal{I}}{\mathcal{U}}$, where $\mathcal{U} = A^p + A^g - \mathcal{I}$.

8 $GIoU = IoU - \frac{A^c - \mathcal{U}}{A^c}$.

9 $\mathcal{L}_{IoU} = 1 - IoU, \quad \mathcal{L}_{GIoU} = 1 - GIoU.$

```
def Giou(rec1,rec2):
    #分别是第一个矩形左右上下的坐标
    x1,x2,y1,y2 = rec1
    x3,x4,y3,y4 = rec2
    iou = Iou(rec1,rec2)
    area_C = (max(x1,x2,x3,x4)-min(x1,x2,x3,x4))*(max(y1,y2,y3,y4)-min(y1,y2,y3,y4))
    area_1 = (x2-x1)*(y1-y2)
    area_2 = (x4-x3)*(y3-y4)
    sum_area = area_1 + area_2

    w1 = x2 - x1    #第一个矩形的宽
    w2 = x4 - x3    #第二个矩形的宽
    h1 = y1 - y2
    h2 = y3 - y4
    W = min(x1,x2,x3,x4)+w1+w2-max(x1,x2,x3,x4)      #交叉部分的宽
    H = min(y1,y2,y3,y4)+h1+h2-max(y1,y2,y3,y4)      #交叉部分的高
```

```
    Area = W*H     #交叉的面积
    add_area = sum_area - Area     #两矩形并集的面积

    end_area = (area_C - add_area)/area_C     #闭包区域中不属于两个框的区域占闭包区域的比重
    giou = iou - end_area
    return giou
```

三、DIoU(Distance-IoU)[2]

1、来源

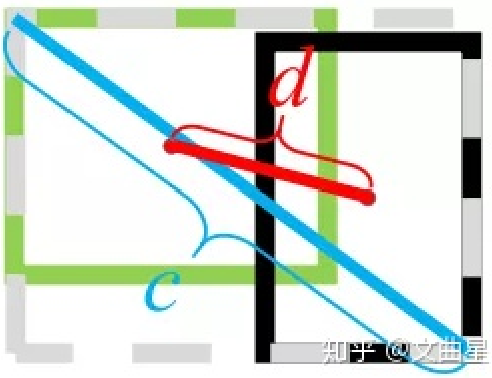
DIoU要比GIou更加符合目标框回归的机制，**将目标与anchor之间的距离，重叠率以及尺度都考虑进去**，使得目标框回归变得更加稳定，不会像IoU和GIoU一样出现训练过程中发散等问题。论文中

Distance-IoU
arxiv.org

> 基于IoU和GIoU存在的问题，作者提出了两个问题：
> 1. 直接最小化anchor框与目标框之间的归一化距离是否可行，以达到更快的收敛速度？
> 2. 如何使回归在与目标框有重叠甚至包含时更准确、更快？

$$DIoU = IoU - \frac{\rho^2(b, b^{gt})}{c^2}$$

其中， b ， b^{gt} 分别代表了预测框和真实框的中心点，且 ρ 代表的是计算两个中心点间的欧式距离。 c 代表的是能够同时包含预测框和真实框的**最小闭包区域**的对角线距离。



DIoU中对anchor框和目标框之间的归一化距离进行了建模

附：

YOLOV3 DIoU GitHub项目地址
github.com

2、优点

与GIoU loss类似，DIoU loss（ $L_{DIoU} = 1 - DIoU$ ）在与目标框不重叠时，仍然可以为边界框提供移动方向。
DIoU loss可以直接最小化两个目标框的距离，因此比GIoU loss收敛快得多。
对于包含两个框在水平方向和垂直方向上这种情况，DIoU损失可以使回归非常快，而GIoU损失几乎退化为IoU损失。
DIoU还可以替换普通的IoU评价策略，应用于NMS中，使得NMS得到的结果更加合理和有效。

实现代码：[3]

```
def Diou(bboxes1, bboxes2):
    rows = bboxes1.shape[0]
    cols = bboxes2.shape[0]
    dious = torch.zeros((rows, cols))
    if rows * cols == 0:#
        return dious
    exchange = False
    if bboxes1.shape[0] > bboxes2.shape[0]:
        bboxes1, bboxes2 = bboxes2, bboxes1
        dious = torch.zeros((cols, rows))
        exchange = True
    # #xmin,ymin,xmax,ymax->[:,0],[:,1],[:,2],[:,3]
    w1 = bboxes1[:, 2] - bboxes1[:, 0]
    h1 = bboxes1[:, 3] - bboxes1[:, 1]
    w2 = bboxes2[:, 2] - bboxes2[:, 0]
    h2 = bboxes2[:, 3] - bboxes2[:, 1]

    area1 = w1 * h1
    area2 = w2 * h2

    center_x1 = (bboxes1[:, 2] + bboxes1[:, 0]) / 2
    center_y1 = (bboxes1[:, 3] + bboxes1[:, 1]) / 2
    center_x2 = (bboxes2[:, 2] + bboxes2[:, 0]) / 2
    center_y2 = (bboxes2[:, 3] + bboxes2[:, 1]) / 2

    inter_max_xy = torch.min(bboxes1[:, 2:],bboxes2[:, 2:])
    inter_min_xy = torch.max(bboxes1[:, :2],bboxes2[:, :2])
    out_max_xy = torch.max(bboxes1[:, 2:],bboxes2[:, 2:])
```

```
    out_min_xy = torch.min(bboxes1[:, :2],bboxes2[:, :2])

    inter = torch.clamp((inter_max_xy - inter_min_xy), min=0)
    inter_area = inter[:, 0] * inter[:, 1]
    inter_diag = (center_x2 - center_x1)**2 + (center_y2 - center_y1)**2
    outer = torch.clamp((out_max_xy - out_min_xy), min=0)
    outer_diag = (outer[:, 0] ** 2) + (outer[:, 1] ** 2)
    union = area1+area2-inter_area
    dious = inter_area / union - (inter_diag) / outer_diag
    dious = torch.clamp(dious,min=-1.0,max = 1.0)
    if exchange:
        dious = dious.T
    return dious
```

四、CIoU(Complete-IoU)

论文考虑到bbox回归三要素中的长宽比还没被考虑到计算中，因此，进一步在DIoU的基础上提出了CIoU。其惩罚项如下面公式：

$$\mathcal{R}_{CIoU}=\frac{\rho^2\left(\mathbf{b},\mathbf{b}^{gt}\right)}{c^2}+\alpha v$$ 其中 α 是权重函数，

而 ν 用来度量长宽比的相似性，定义为 $v=\frac{4}{\pi^2}\left(\arctan\frac{w^{gt}}{h^{gt}}-\arctan\frac{w}{h}\right)^2$

完整的 CIoU 损失函数定义：

$$\mathcal{L}_{CIoU}=1-IoU+\frac{\rho^2\left(\mathbf{b},\mathbf{b}^{gt}\right)}{c^2}+\alpha v$$

最后，CIoU loss的梯度类似于DIoU loss，但还要考虑 ν 的梯度。在长宽在 $[0,1]$ 的情况下，w^2+h^2 的值通常很小，会导致梯度爆炸，因此在 $\frac{1}{w^2+h^2}$ 实现时将替换成1。[4]

实现代码：[5]

```
def bbox_overlaps_ciou(bboxes1, bboxes2):
    rows = bboxes1.shape[0]
    cols = bboxes2.shape[0]
    cious = torch.zeros((rows, cols))
```

```python
if rows * cols == 0:
    return cious
exchange = False
if bboxes1.shape[0] > bboxes2.shape[0]:
    bboxes1, bboxes2 = bboxes2, bboxes1
    cious = torch.zeros((cols, rows))
    exchange = True

w1 = bboxes1[:, 2] - bboxes1[:, 0]
h1 = bboxes1[:, 3] - bboxes1[:, 1]
w2 = bboxes2[:, 2] - bboxes2[:, 0]
h2 = bboxes2[:, 3] - bboxes2[:, 1]

area1 = w1 * h1
area2 = w2 * h2

center_x1 = (bboxes1[:, 2] + bboxes1[:, 0]) / 2
center_y1 = (bboxes1[:, 3] + bboxes1[:, 1]) / 2
center_x2 = (bboxes2[:, 2] + bboxes2[:, 0]) / 2
center_y2 = (bboxes2[:, 3] + bboxes2[:, 1]) / 2

inter_max_xy = torch.min(bboxes1[:, 2:],bboxes2[:, 2:])
inter_min_xy = torch.max(bboxes1[:, :2],bboxes2[:, :2])
out_max_xy = torch.max(bboxes1[:, 2:],bboxes2[:, 2:])
out_min_xy = torch.min(bboxes1[:, :2],bboxes2[:, :2])

inter = torch.clamp((inter_max_xy - inter_min_xy), min=0)
inter_area = inter[:, 0] * inter[:, 1]
inter_diag = (center_x2 - center_x1)**2 + (center_y2 - center_y1)**2
outer = torch.clamp((out_max_xy - out_min_xy), min=0)
outer_diag = (outer[:, 0] ** 2) + (outer[:, 1] ** 2)
union = area1+area2-inter_area
u = (inter_diag) / outer_diag
iou = inter_area / union
with torch.no_grad():
    arctan = torch.atan(w2 / h2) - torch.atan(w1 / h1)
    v = (4 / (math.pi ** 2)) * torch.pow((torch.atan(w2 / h2) - torch.atan(w1 / h1
    S = 1 - iou
    alpha = v / (S + v)
    w_temp = 2 * w1
ar = (8 / (math.pi ** 2)) * arctan * ((w1 - w_temp) * h1)
cious = iou - (u + alpha * ar)
cious = torch.clamp(cious,min=-1.0,max = 1.0)
if exchange:
```

```
        cious = cious.T
    return cious
```

五、损失函数在YOLOv3上的性能(论文效果)

Table 1: Quantitative comparison of **YOLOv3** (Redmon and Farhadi 2018) trained using $\mathcal{L}_{IoU}$ (baseline), $\mathcal{L}_{GIoU}$, $\mathcal{L}_{DIoU}$ and $\mathcal{L}_{CIoU}$. (D) denotes using DIoU-NMS. The results are reported on the test set of PASCAL VOC 2007.

Loss / Evaluation	AP		AP75	
	IoU	**GIoU**	**IoU**	**GIoU**
$\mathcal{L}_{IoU}$	46.57	45.82	49.82	48.76
$\mathcal{L}_{GIoU}$	47.73	46.88	52.20	51.05
Relative improv. %	2.49%	2.31%	4.78%	4.70%
$\mathcal{L}_{DIoU}$	48.10	47.38	52.82	51.88
Relative improv. %	3.29%	3.40%	6.02%	6.40%
$\mathcal{L}_{CIoU}$	**49.21**	**48.42**	**54.28**	**52.87**
Relative improv. %	**5.67%**	**5.67%**	**8.95%**	**8.43%**
$\mathcal{L}_{CIoU}(D)$	**49.32**	**48.54**	**54.74**	**53.30**
Relative improv. %	**5.91%**	**5.94%**	**9.88%**	**9.31%**

目标检测算法之AAAI 2020 DIoU Loss 已开源(YOLOV3涨近3个点)

cloud.tencent.com