```
[convolutional]
size=1
stride=1
pad=1
filters=75
activation=linear

[yolo]
mask = 6,7,8
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=20
num=9
jitter=.3
ignore_thresh = .5
truth_thresh = 1
random=1
```

不同尺度上对应的anchor box索引

anchors的大小

目标类别数目

每个grid cell总共预测几个box,和anchors的数量一致。

数据增强手段，此处jitter为随机调整宽高比的范围

参与计算的IOU阈值大小.当预测的检测框与ground true的IOU大于ignore_thresh的时候，参与loss的计算，否则，检测框的不参与损失计算

```
[convolutional]
size=1
stride=1
pad=1
filters=75
activation=linear

[yolo]
mask = 3,4,5
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=20
num=9
jitter=.3
ignore_thresh = .5
truth_thresh = 1
random=1

[route]
layers = -4
```

```
[convolutional]
size=1
stride=1
pad=1
filters=75
activation=linear

[yolo]
mask = 0,1,2
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=20
num=9
jitter=.3
ignore_thresh = .5
truth_thresh = 1
random=1
```

类别预测(Class Prediction)

- 大多数分类器假设输出标签是互斥的。 如果输出是互斥的目标类别，则确实如此。 因此，YOLO应用softmax函数将得分转换为总和为1的概率。 而YOLOv3使用多标签分类。 例如，输出标签可以是“行人”和“儿童”，它们不是非排他性的。 （现在输出的总和可以大于1）
- YOLOv3用多个独立的逻辑（logistic）分类器替换softmax函数，以计算输入属于特定标签的可能性。在计算分类损失时，YOLOv3对每个标签使用二元交叉熵损失。 这也避免使用softmax函数而降低了计算复杂度。

Logistic classification with cross-entropy

Logistic classification with cross-entropy

Logistic function $$\sigma(z) = \frac{\mathbf{1}}{\mathbf{1} + e^{-z}}$$

t : the class correct 1 or not 0

$$P(t=\mathbf{1} \mid z) = \sigma(z) = \frac{\mathbf{1}}{\mathbf{1} + e^{-z}}$$

$$P(t=\mathbf{0} \mid z) = \mathbf{1} - \sigma(z) = \frac{e^{-z}}{\mathbf{1} + e^{-z}}$$

$$P(t=\mathbf{1} \mid z) = \sigma(z) = y$$

cross-entropy error function

$$\xi(t, y) = -t \log(y) - (\mathbf{1} - t)\log(\mathbf{1} - y)$$

$$\frac{\partial \xi}{\partial z} = \frac{\partial y}{\partial z}\frac{\partial \xi}{\partial y} = y(\mathbf{1} - y)\frac{y - t}{y(\mathbf{1} - y)} = y - t$$

```
delta[index + stride*n] = ((n == class_id) ? 1 : 0) - output[index + stride*n];
```

The same:

t==1: i.e. if (detected_class == truth_class) delta = -loss_derivative = -(y-t) = 1-y

t==0: i.e. if (detected_class != truth_class) delta = -loss_derivative = -(y-t) = -y

边界框预测和代价函数计算
(Bounding box prediction & cost function calculation)

- YOLOv3使用逻辑回归（logistic）预测每个边界框的目标性得分(objectness score)。
- YOLOv3改变了计算代价函数的方式。
  - ➢ 如果边界框先验（锚定框）与GT目标比其他目标重叠多，则相应的目标性得分应为1。
  - ➢ 对于重叠大于预定义阈值（默认值0.5）的其它先验框，不会产生任何代价。
  - ➢ 每个GT目标仅与一个先验边界框相关联。 如果没有分配先验边界框，则不会导致分类和定位损失，只会有目标性的置信度损失。
  - ➢ 使用tx和ty（而不是bx和by）来计算损失。

Loss function of YOLOv3未直接定义, look at src/yolo_layer.c

delta for box, line 93

```
float delta_yolo_box(box truth, float *x, float *biases, int n, int index, int i, int j, int lw,
int lh, int w, int h, float *delta, float scale, int stride)
{
    box pred = get_yolo_box(x, biases, n, index, i, j, lw, lh, w, h, stride);
    float iou = box_iou(pred, truth);

    float tx = (truth.x*lw - i);
    float ty = (truth.y*lh - j);
    float tw = log(truth.w*w / biases[2*n]);
    float th = log(truth.h*h / biases[2*n + 1]);

    delta[index + 0*stride] = scale * (tx - x[index + 0*stride]);
    delta[index + 1*stride] = scale * (ty - x[index + 1*stride]);
    delta[index + 2*stride] = scale * (tw - x[index + 2*stride]);
    delta[index + 3*stride] = scale * (th - x[index + 3*stride]);
    return iou;
}
```

delta for class

```
void delta_yolo_class(float *output, float *delta, int index, int class, int classes, int stride, float
*avg_cat)
{
    int n;
    if (delta[index]){
        delta[index + stride*class] = 1 - output[index + stride*class];
        if(avg_cat) *avg_cat += output[index + stride*class];
        return;
    }
    for(n = 0; n < classes; ++n){
        delta[index + stride*n] = ((n == class)?1 : 0) - output[index + stride*n];
        if(n == class && avg_cat) *avg_cat += output[index + stride*n];
    }
}
```

delta for objectness

```
l.delta[obj_index] = 0 - l.output[obj_index];
                    if (best_iou > l.ignore_thresh) {
                     //对于IoU大于预定义阈值（默认值0.5）的其它先验框，不会产生任何代价。
                        l.delta[obj_index] = 0;
                     }
                     if (best_iou > l.truth_thresh) {
                        l.delta[obj_index] = 1 - l.output[obj_index];
}
```