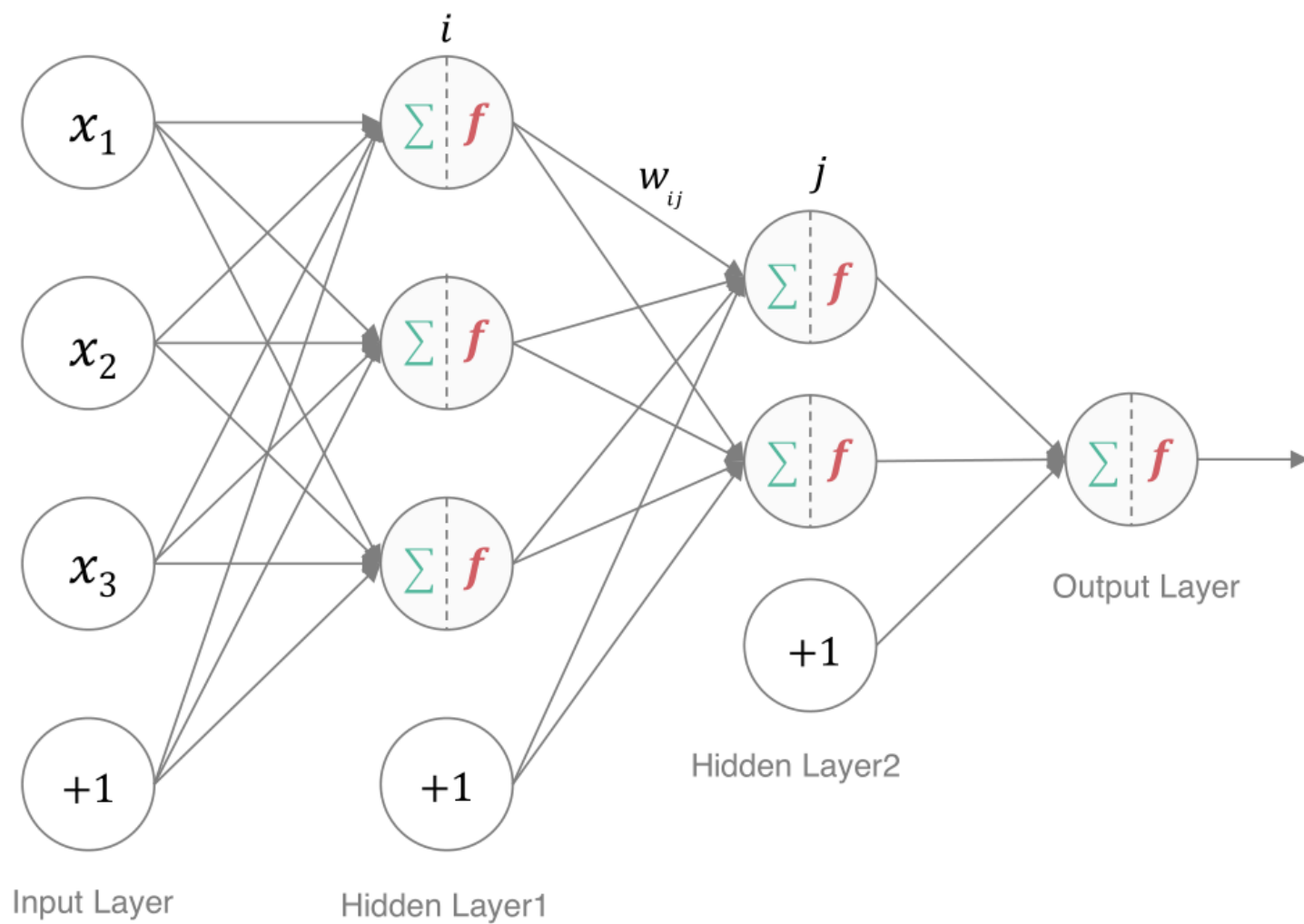


反向传播与误差（敏感度图）计算



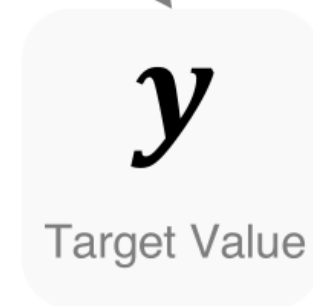
神经网络训练



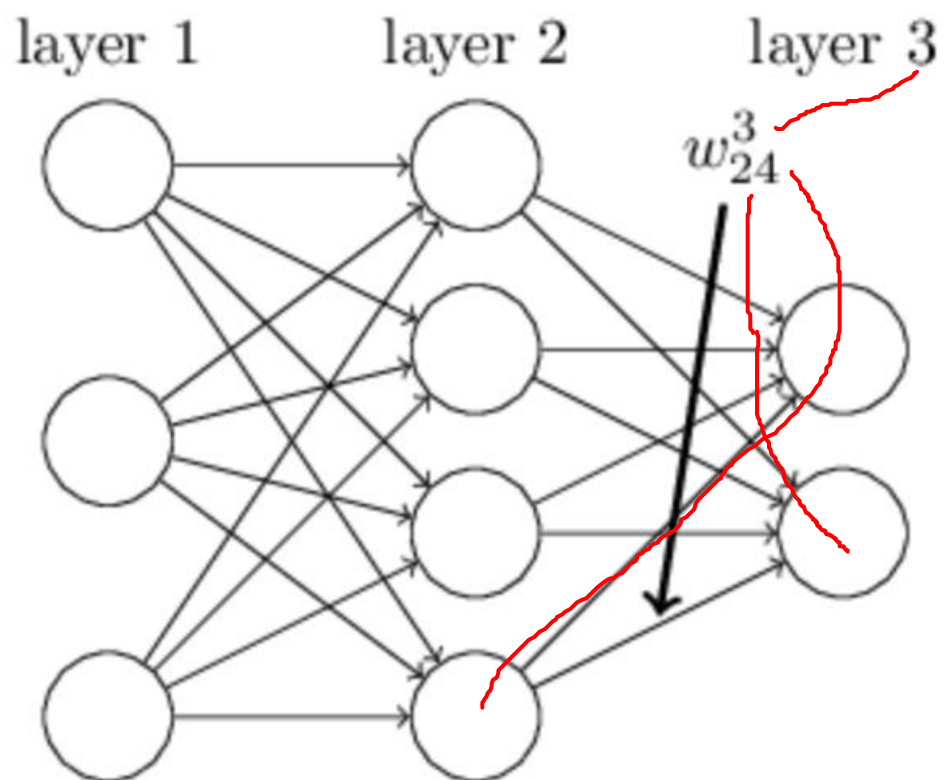
$$w_{ij}^{new} = w_{ij}^{old} - \eta \frac{\partial E}{\partial w_{ij}}$$

η 是学习率

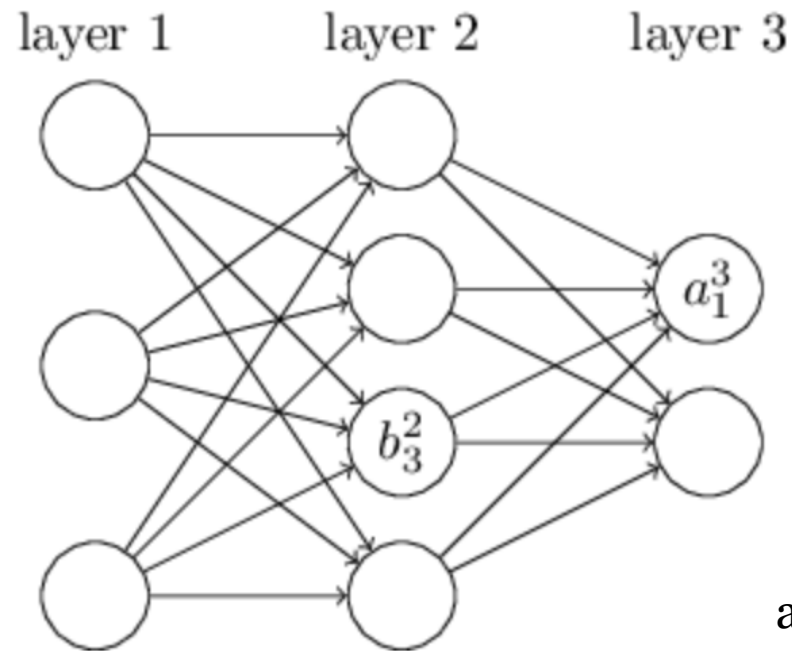
Difference



$$Error = (output - target)^2$$



w_{jk}^l is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer



$$a_j^l = \sigma \left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \right)$$



activation vector

$$a^l = \sigma \left(w^l a^{l-1} + b^l \right)$$

weight matrix

bias vector

how the activations in one layer relate to activations in the previous layer: we just apply the weight matrix to the activations, then add the bias vector, and finally apply the σ function.

反向传播是关于如何改变网络中的权重和偏置从而改变代价函数。这意味着计算偏导数 $\partial C / \partial w_{jk}^l$ 和 $\partial C / \partial b_j^l$ 。为了计算这些，引入一个中间量 δ_j^l ，将其称为第l层第j个神经元中的误差(error)。或称为敏感度图 (sensitivity map)。

反向传播将给我们一个计算误差 δ_j^l 的过程，然后将 δ_j^l 与 $\partial C / \partial w_{jk}^l$ 和 $\partial C / \partial b_j^l$ 联系起来。

weighted input

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

error

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$$

z_j^l the weighted input to the activation function for neuron j in layer l


δ^l vector of errors associated with layer l

An equation for the error in the output layer

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L)$$

$$C = \frac{1}{2} \sum_j (y_j - a_j^L)^2 \quad \longrightarrow \quad \partial C / \partial a_j^L = (a_j^L - y_j)$$

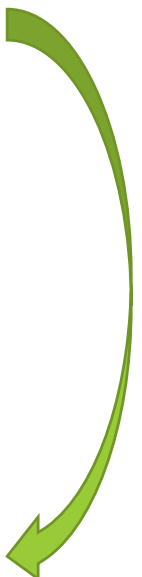
desired output



$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L) \quad \longrightarrow \quad \delta^L = \nabla_a C \odot \sigma'(z^L)$$

Hadamard product

elementwise product of the two vectors $(s \odot t)_j = s_j t_j$

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 * 3 \\ 2 * 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix} \quad \delta^L = (a^L - y) \odot \sigma'(z^L)$$


An equation for the error δ^l in terms of the error in the next layer, δ^{l+1}

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

或者 An equation for the error δ^{l-1} in terms of the error in the next layer, δ^l

$$\delta^{l-1} = ((w^l)^T \delta^l) \odot \sigma'(z^{l-1})$$

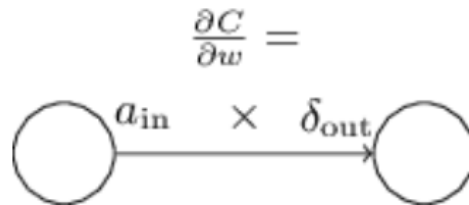
An equation for the rate of change of the cost with respect to any bias in the network:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

An equation for the rate of change of the cost with respect to any weight in the network:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

$$\frac{\partial C}{\partial w} = a_{\text{in}} \delta_{\text{out}}$$



推导过程可参考: <http://neuralnetworksanddeeplearning.com/chap2.html>

对于全连接网络：

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

$$\delta^{l-1} = \left((w^l)^T \delta^l \right) \odot \sigma'(z^{l-1}) \quad (\text{FC-1})$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad (\text{FC-2})$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \quad (\text{FC-3})$$

backpropagation algorithm

反向传播方程提供了一种计算代价函数梯度的方法。

1. **Input x :** Set the corresponding activation a^1 for the input layer.
2. **Feedforward:** For each $l = 2, 3, \dots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.
3. **Output error δ^L :** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.
4. **Backpropagate the error:** For each $l = L - 1, L - 2, \dots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.
5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$ and $\frac{\partial C}{\partial b_j^l} = \delta_j^l$.

In particular, given a mini-batch of m training examples, the following algorithm applies a gradient descent learning step based on that [mini-batch](#):

1. **Input a set of training examples**

2. **For each training example x :** Set the corresponding input activation $a^{x,1}$, and perform the following steps:

◦ **Feedforward:** For each $l = 2, 3, \dots, L$ compute

$$z^{x,l} = w^l a^{x,l-1} + b^l \text{ and } a^{x,l} = \sigma(z^{x,l}).$$

◦ **Output error $\delta^{x,L}$:** Compute the vector

$$\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L}).$$

◦ **Backpropagate the error:** For each

$l = L - 1, L - 2, \dots, 2$ compute

$$\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l}).$$

3. **Gradient descent:** For each $l = L, L - 1, \dots, 2$ update the weights according to the rule $w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$, and the biases according to the rule $b^l \rightarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$.

若 l 层为卷积层:

*为卷积

$$\delta^{l-1} = \delta^l * \text{rot}180(w^l) \odot \sigma'(z^{l-1}) \quad (\text{Conv-1})$$

偏置更新需要的梯度:

偏置项的梯度就是sensitivity map所有误差项之和

$$\frac{\partial C}{\partial b^l} = \sum_{u,v} \delta_{w,h} \quad (\text{Conv-2})$$

其中 u,v 代表卷积核输出的size的长宽

权重更新需要的梯度:

$$\frac{\partial C}{\partial w^l} = a^{l-1} * \delta^l \quad (\text{Conv-3})$$

$$b^l = b^l - \alpha \frac{\partial C}{\partial b^l} \quad (\text{FC-4; Conv-4})$$

$$w^l = w^l - \alpha \frac{\partial C}{\partial w^l} \quad (\text{FC-5; Conv-5})$$

前向传播:

layer $l-1$

$\delta_{1,1}$	$\delta_{1,2}$	$\delta_{1,3}$
$\delta_{2,1}$	$\delta_{2,2}$	$\delta_{2,3}$
$\delta_{3,1}$	$\delta_{3,2}$	$\delta_{3,3}$

input
 3×3

$W_{1,1}$	$W_{1,2}$
$W_{2,1}$	$W_{2,2}$

W_b

filter
 2×2



layer l

$\delta_{1,1}$	$\delta_{1,2}$
$\delta_{2,1}$	$\delta_{2,2}$

feature map
 2×2

反向传播:

layer l

	$\delta_{1,1}$	$\delta_{1,2}$	
	$\delta_{2,1}$	$\delta_{2,2}$	

sensitivity map
 2×2

$W_{2,2}$	$W_{2,1}$
$W_{1,2}$	$W_{1,1}$

flipped filter
 2×2



layer $l-1$

$\delta_{1,1}$	$\delta_{1,2}$	$\delta_{1,3}$
$\delta_{2,1}$	$\delta_{2,2}$	$\delta_{2,3}$
$\delta_{3,1}$	$\delta_{3,2}$	$\delta_{3,3}$

input
 3×3

Pooling层的训练

无论max pooling还是average pooling，都没有需要学习的参数。因此，在卷积神经网络的训练中，Pooling层需要做的仅仅是将**误差**传递到上一层，而没有**梯度**的计算。

- 对于max pooling，下一层的**误差**的值会原封不动的传递到上一层对应区块中的最大值所对应的神经元，而其他神经元的**误差**的值都是0
- 对于average pooling，下一层的**误差**的值会**平均分配**到上一层对应区块中的所有神经元。

YOLOv3当前层的参数的具体梯度计算过程

设当前层为第 $l-1$ 层,那么计算其敏感度分两步:

1. 在 l 层的backward()函数的最后部分,会计算 $l-1$ 层的

$$delta^{l-1} = \delta^l \frac{\partial z^l}{\partial a^{l-1}} \quad (P1)$$

2. 在 $l-1$ 层调用backward函数开头部分,再计算:

$$\delta^{l-1} = delta^{l-1} \odot \sigma' (z^{l-1}) \quad (P2)$$

参数更新

$$w = w - \alpha \nabla C \quad (\text{update-1})$$

引入动量的参数更新:

$$v_t = \gamma v_{t-1} \quad (\text{update-2})$$

$$w = w - v_t - \alpha \nabla C \quad (\text{update-3})$$

$$w = w - \frac{\lambda}{m} w \quad (\text{update-4})$$

BN层反向传播过程

设最终的损失函数为C, 对方差求导:

$$\begin{aligned}
 \frac{\partial C}{\partial (\sigma_B^2)^l} &= \sum_{i=1}^m \left(\frac{\partial C}{\partial \hat{z}_i^l} \frac{\partial \hat{z}_i^l}{\partial (\sigma_B^2)^l} \right) \\
 &= \sum_{i=1}^m \left\{ \frac{-1}{2} \frac{\partial C}{\partial z_i^{l+1}} \frac{\partial z_i^{l+1}}{\partial a_i^l} \frac{\partial a_i^l}{\partial y_i^l} \frac{\partial y_i^l}{\partial \hat{z}_i^l} (z_i^l - \mu_B^l) \left[(\sigma_B^2)^{(l)} + \varepsilon \right]^{-\frac{3}{2}} \right\} \\
 &= \sum_{i=1}^m \left\{ \frac{-1}{2} \left[\delta^{l+1} \frac{\partial z_i^{l+1}}{\partial a_i^l} \odot \left(g(y_i^l)' \cdot \gamma_i^l \right) \right] (z_i^l - \mu_B^l) \left[(\sigma_B^2)^{(l)} + \varepsilon \right]^{-\frac{3}{2}} \right\} \\
 &= \gamma^l \odot \sum_{i=1}^m \left\{ \frac{-1}{2} \left[\delta^{l+1} \frac{\partial z_i^{l+1}}{\partial a_i^l} \odot \sigma(y_i^l)' \right] (z_i^l - \mu_B^l) \left[(\sigma_B^2)^{(l)} + \varepsilon \right]^{-\frac{3}{2}} \right\} \quad (\text{BN 2.1})
 \end{aligned}$$

$$\frac{\partial C}{\partial \mu_B^l} = \sum_{i=1}^m \left(\frac{\partial C}{\partial \hat{z}_i} \frac{\hat{z}_i}{\partial \mu_B^l} + \frac{\partial C}{\partial \sigma_B^2} \frac{\partial \sigma_B^2}{\partial \mu_B^l} \right) = \sum_{i=1}^m \left(\frac{\partial C}{\partial \hat{z}_i} \frac{-\mathbf{1}}{\sqrt{(\sigma_B^2)^l + \varepsilon}} \right) + \frac{\partial C}{\partial (\sigma_B^2)^l} \cdot \frac{-2}{m} \cdot \sum_i^m (z_i^l - \mu_B^l)$$

$$\begin{aligned} \sum_{i=1}^m \left(\frac{\partial C}{\partial \hat{z}_i} \frac{-\mathbf{1}}{\sqrt{(\sigma_B^2)^l + \varepsilon}} \right) &= \sum_{i=1}^m \left(\frac{\partial C}{\partial z_i^{l+1}} \frac{\partial z_i^{l+1}}{\partial a_i^l} \frac{\partial a_i^l}{\partial y_i^l} \frac{\partial y_i^l}{\partial \hat{z}_i^l} \frac{-\mathbf{1}}{\sqrt{(\sigma_B^2)^l + \varepsilon}} \right) \\ &= \gamma^l \odot \sum_{i=1}^m \left(\left[\delta^{l+1} \frac{\partial z_i^{l+1}}{\partial a_i^l} \odot g(y_i^l)' \right] \frac{-\mathbf{1}}{\sqrt{(\sigma_B^2)^l + \varepsilon}} \right) \end{aligned} \quad (\text{BN 2.2})$$

$$\begin{aligned}
\delta_i^l &= \frac{\partial C}{\partial z_i^l} = \frac{\partial C}{\partial \hat{z}_i^l} \frac{\partial \hat{z}_i^l}{\partial z_i^l} + \frac{\partial C}{\partial \sigma_B^2} \frac{\partial \sigma_B^2}{\partial z_i^l} + \frac{\partial C}{\partial \mu_B} \frac{\partial \mu_B}{\partial z_i^l} \\
&= \frac{\partial C}{\partial \hat{z}_i^l} \frac{\mathbf{1}}{\sqrt{\sigma_B^2 + \varepsilon}} + \frac{\partial C}{\partial \sigma_B^2} \cdot \frac{\mathbf{2}}{m} \cdot (z_i^l - \mu_B) + \frac{\partial C}{\partial \mu_B} \cdot \frac{\mathbf{1}}{m} \\
&= \frac{\partial C}{\partial z_i^{l+1}} \frac{\partial z_i^{l+1}}{\partial a_i^l} \frac{\partial a_i^l}{\partial y_i^l} \frac{\partial y_i^l}{\partial \hat{z}_i^l} + \frac{\partial C}{\sqrt{\sigma_B^2 + \varepsilon}} + \frac{\partial C}{\partial \sigma_B^2} \cdot \frac{\mathbf{2}}{m} \cdot (z_i^l - \mu_B) + \frac{\partial C}{\partial \mu_B} \cdot \frac{\mathbf{1}}{m} \\
&= \left[\delta^{l+1} \frac{\partial z_i^{l+1}}{\partial a_i^l} \odot \left(g(y_i^l)' \odot \gamma_i^l \right) \right] \frac{\mathbf{1}}{\sqrt{\sigma_B^2 + \varepsilon}} + \frac{\partial C}{\partial \sigma_B^2} \cdot \frac{\mathbf{2}}{m} \cdot (z_i^l - \mu_B) + \frac{\partial C}{\partial \mu_B} \cdot \frac{\mathbf{1}}{m} \quad (\text{BN 2.3})
\end{aligned}$$

求权重和偏差梯度：

$$\frac{\partial C}{\partial w^l} = \sum_{i=1}^m \frac{\partial C}{\partial z_i^l} \frac{\partial z_i^l}{\partial w^l} = \sum_{i=1}^m \left(a^{l-1} \right)^T \delta^l \quad (\text{BN 2.4})$$

$$\begin{aligned} \frac{\partial C}{\partial \beta^l} &= \sum_{i=1}^m \frac{\partial C}{\partial z_i^{l+1}} \frac{\partial z_i^{l+1}}{\partial a_i^l} \frac{\partial a_i^l}{\partial y_i^l} \frac{\partial y_i^l}{\partial \beta^l} = \sum_{i=1}^m \delta_i^{l+1} \frac{\partial z_i^{l+1}}{\partial a_i^l} \odot \left[g(y_i^l)' \frac{\partial y_i^l}{\partial \beta^l} \right] \\ &= \sum_{i=1}^m \delta_i^{l+1} \frac{\partial z_i^{l+1}}{\partial a_i^l} \odot g(y_i^l)' \end{aligned} \quad (\text{BN 2.5})$$

$$\frac{\partial C}{\partial \gamma^l} = \sum_{i=1}^m \frac{\partial C}{\partial z_i^{l+1}} \frac{\partial z_i^{l+1}}{\partial a_i^l} \frac{\partial a_i^l}{\partial y_i^l} \frac{\partial y_i^l}{\partial \gamma^l} = \sum_{i=1}^m \delta_i^{l+1} \frac{\partial z_i^{l+1}}{\partial a_i^l} \odot \left[g(y_i^l)' \odot \hat{z}_i^l \right] \quad (\text{BN 2.6})$$

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

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

t : the class correct 1 or not 0

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

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

cross-entropy error function

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

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

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

The same:

$t=1$: i.e. if (detected_class == truth_class) $\text{delta} = -\text{loss_derivative} = -(y-t) = 1-y$

$t=0$: i.e. if (detected_class != truth_class) $\text{delta} = -\text{loss_derivative} = -(y-t) = -y$

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];
    }
```