

# Logistic Regression

## 1. Binary Classification

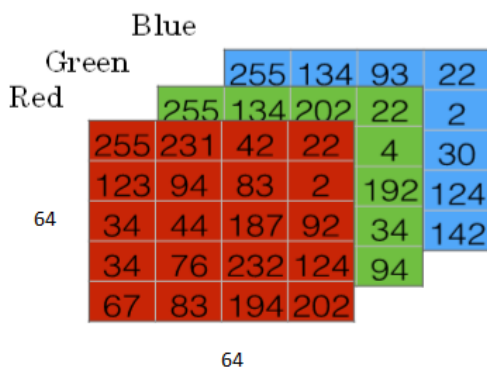
逻辑回归模型一般用来解决二分类（Binary Classification）问题。二分类就是输出y只有{0,1}两个离散值（也有{-1,1}的情况）。

以一个图像识别问题为例，判断图片中是否有猫存在，0代表noncat，1代表cat。

## Binary Classification



$$\begin{cases} 1, & \text{cat} \\ 0, & \text{not cat} \end{cases}$$



$$X_{\text{orig}} = 64 \times 64 \times 3 = 12288$$

$$X = \begin{bmatrix} 255 \\ 231 \\ \dots \\ 255 \\ 134 \\ \dots \\ 132 \end{bmatrix} \quad n = n_x = 12288$$

如上图所示，这是一个典型的二分类问题。

一般来说，彩色图片包含RGB三个通道。例如该cat图片的尺寸为（64，64，3）。在神经网络模型中，我们首先要将图片输入x（维度是（64，64，3））转化为一维的特征向量（feature vector）。方法是每个通道一行一行取，再连接起来。由于64x64x3=12288，则转化后的输入特征向量维度为（12288，1）。此特征向量x是列向量，维度一般记为 $n_x$ 。

如果训练样本共有m张图片，那么整个训练样本X组成了矩阵，维度是（ $n_x$ ，m）。

注意，这里矩阵X的行 $n_x$ 代表了每个样本 $x^{(i)}$ 特征个数，列m代表了样本个数。

这里，X的维度之所以是（ $n_x$ ，m）而不是（m， $n_x$ ）的原因是为了之后矩阵运算的方便。而所有训练样本的输出Y也组成了一维的行向量，写成矩阵的形式后，它的维度就是（1，m）。

## 2. Logistic Regression

逻辑回归中，预测值 $\hat{h} = P(y = 1|x)$ 表示为1的概率，取值范围在[0,1]之间，这是其与二分类模型不同的地方。

使用线性模型，引入参数w和b。权重w的维度是（ $n_x$ ，1），b是一个常数项。

这样，逻辑回归的线性预测输出可以写成

$$\hat{y} = w^T x + b$$

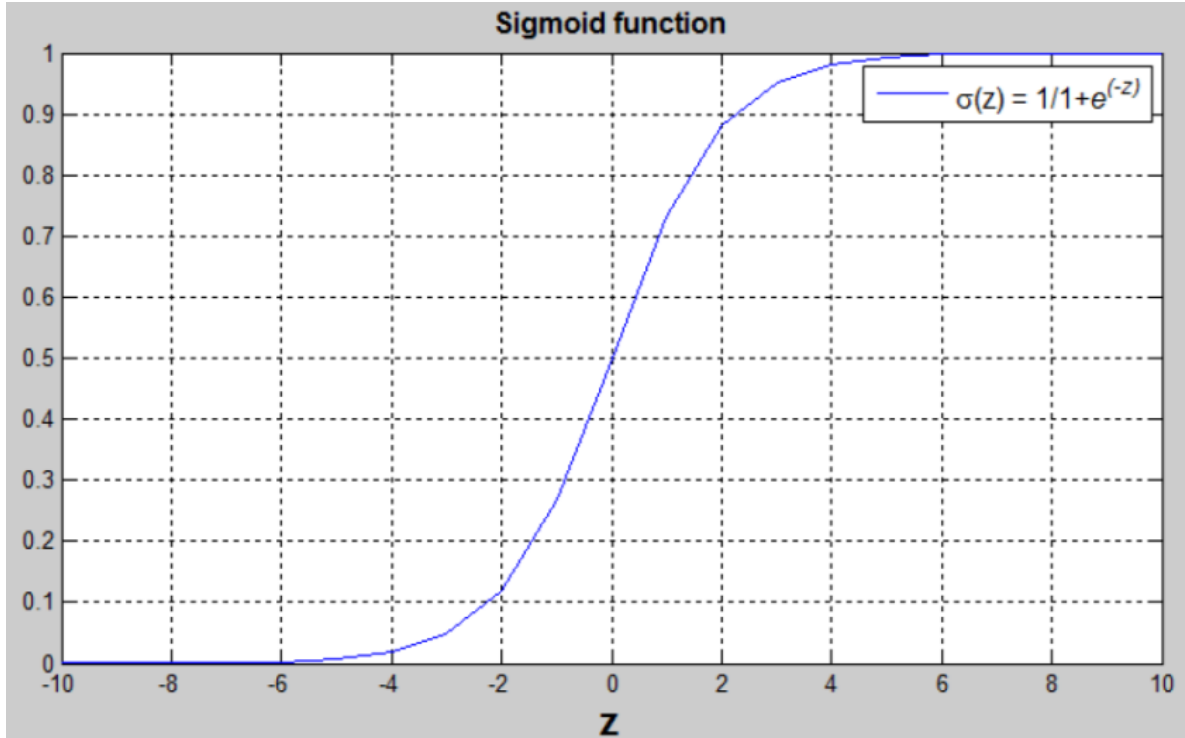
也可以把常数b当作 $w_0$ 处理，并引入 $x_0 = 1$

先行输出区间为整个实数范围，而逻辑回归要求输出范围在[0,1]之间，所以还需要对上式的线性函数输出进行处理。方法是引入Sigmoid函数，让输出限定在[0,1]之间。这样，逻辑回归的预测输出就可以完整写成

$$\hat{y} = \text{Sigmoid}(w^T x + b) = \sigma(w^T x + b)$$

Sigmoid函数是一种非线性的S型函数，输出被限定在[0,1]之间，通常被用在神经网络中当作激活函数（Activation function）使用。Sigmoid函数的表达式和曲线如下所示：

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



从Sigmoid函数曲线可以看出，当z值很大时，函数值趋向于1；当z值很小时，函数值趋向于0。且当z=0时，函数值为0.5。

Sigmoid函数的一阶导数可以用其自身表示：

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

### 3.Logistic Regression Cost Function

逻辑回归中，w和b都是未知参数，需要反复训练优化得到。

因此，需要定义一个cost function，包含了参数w和b。通过优化cost function，当cost function取值最小时，得到对应的w和b。

先从单个样本出发，我们希望该样本的预测值 $\hat{y}$ 与真实值越相似越好，把单个样本的cost function用Loss function来表示，如果用平方错误(squared error)来衡量，如下所示：

$$L(\hat{y}, y) = \frac{1}{2} (\hat{y} - y)^2$$

但是，对于逻辑回归，一般不使用平方错误来作为Loss Function。原因是这种Loss function一般都是non-convex的。non-convex函数在使用梯度下降算法时，容易得到局部最小值(local minimum)，即局部最优化。而目标是计算得到全局最优化(global optimization)。因此，一般选择的Loss function应该是convex的。

Loss function的原则和目的就是要衡量预测输出 $\hat{y}$ 与真实样本输出y的接近程度。平方错误其实也可以，只是它是non-convex的，不利于使用梯度下降算法来进行全局优化。

因此，可以构建另外一种Loss function，且是convex的，如下：

$$L(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$

分析一波这个loss function，它是衡量错误大小的，loss function越小越好。

当 $y=1$ 时,  $L(\hat{y}, y) = -\log \hat{y}$ 。如果 $\hat{y}$ 越接近1,  $L(\hat{y}, y) \approx 0$ , 表示预测效果越好; 如果 $\hat{y}$ 越接近0,  $L(\hat{y}, y) \approx +\infty$ , 表示预测效果越差。

当 $y=0$ 时,  $L(\hat{y}, y) = -\log(1 - \hat{y})$ 。如果 $\hat{y}$ 越接近0,  $L(\hat{y}, y) \approx 0$ , 表示预测效果越好; 如果 $\hat{y}$ 越接近1,  $L(\hat{y}, y) \approx +\infty$ , 表示预测效果越差。

因此, 这个Loss function能够很好地反映预测输出 $\hat{y}$ 与真实样本输出 $y$ 的接近程度。越接近, 其loss function值越小。而且这个函数是convex的。

上面介绍的loss function是针对单个样本的。那对于 $m$ 个样本, 我们定义cost function, cost function是 $m$ 个样本的loss function的平均值, 反映了 $m$ 个样本的预测输出 $\hat{y}$ 与真实样本输出 $y$ 的平均接近程度。cost function可表示为:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m L(y^{(i)}, \hat{y}^{(i)}) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

Cost function已经推导出来了, Cost function是关于待求系数 $w$ 和 $b$ 的函数。

目标就是迭代计算出最佳的 $w$ 和 $b$ 值, 最小化Cost function, 让Cost function尽可能地接近于零

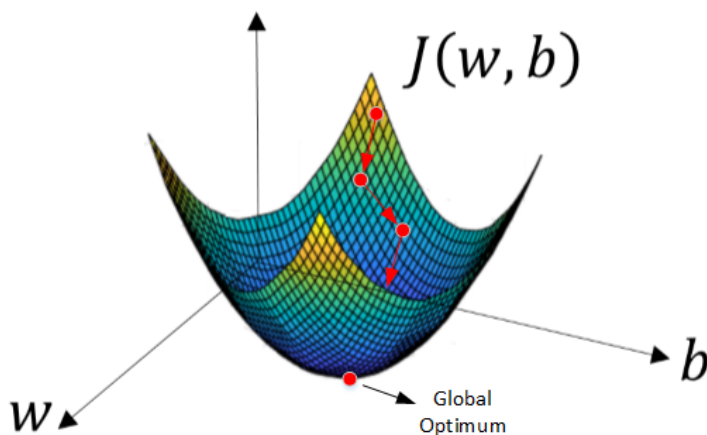
逻辑回归问题可以看成是一个简单的神经网络, 只包含一个神经元。

## 4.Gradient Descent

接下来将使用梯度下降 (Gradient Descent) 算法来计算出合适的 $w$ 和 $b$ 值, 从而最小化 $m$ 个训练样本的Cost function, 即 $J(w, b)$ 。

由于 $J(w, b)$ 是convex function, 梯度下降算法是先随机选择一组参数 $w$ 和 $b$ 值, 然后每次迭代的过程中分别沿着 $w$ 和 $b$ 的梯度 (偏导数) 的反方向前进一小步, 不断修正 $w$ 和 $b$ 。每次迭代更新 $w$ 和 $b$ 后, 都能让 $J(w, b)$ 更接近全局最小值。

梯度下降的过程如下图所示。



梯度下降算法每次迭代更新,  $w$ 和 $b$ 的修正表达式为:

$$w := w - \alpha \frac{\partial J(w, b)}{\partial w}$$

$$b := b - \alpha \frac{\partial J(w, b)}{\partial b}$$

梯度下降算法能够保证每次迭代 $w$ 和 $b$ 都能向着 $J(w, b)$ 全局最小化的方向进行。其数学原理主要是运用泰勒一阶展开来证明的。

## 5.Derivatives

简单的求导数方法不再赘述。

## 6.More Derivative Examples

略

## 7.Computation graph

整个神经网络的训练过程实际上包含了两个过程：正向传播（Forward Propagation）和反向传播（Back Propagation）。

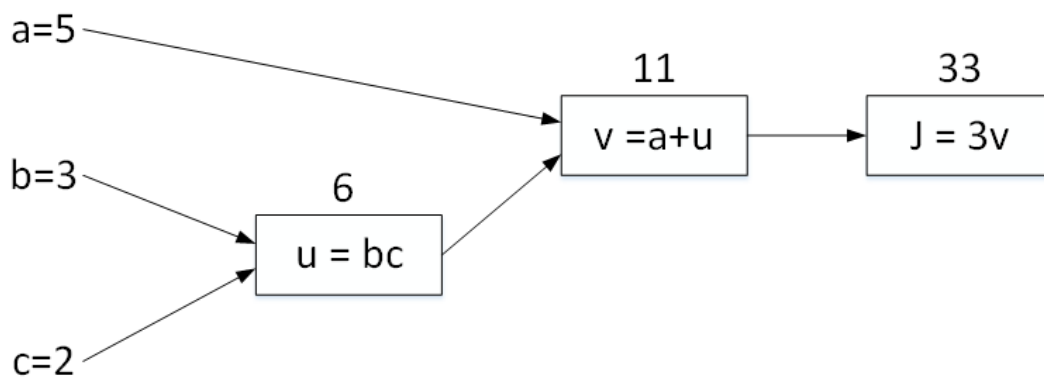
正向传播是从输入到输出，由神经网络计算得到预测输出的过程；

反向传播是从输出到输入，对参数w和b计算梯度的过程。

举个简单的例子，假如Cost function为 $J(a,b,c)=3(a+bc)$ ，包含a, b, c三个变量。我们用u表示bc, v表示a+u, 则 $J=3v$ 。它的计算图可以写成如下图所示：

$$J(a,b,c) = 3(a+bc) = 3(5+3 \times 2) = 33$$

$$\begin{aligned} u &= bc \\ v &= a+u \\ J &= 3v \end{aligned}$$



令 $a=5$ ,  $b=3$ ,  $c=2$ , 则 $u=bc=6$ ,  $v=a+u=11$ ,  $J=3v=33$ 。计算图中，这种从左到右，从输入到输出的过程就对应着神经网络或者逻辑回归中输入与权重经过运算计算得到Cost function的正向过程。

## 8. Derivatives with a Computation Graph

上一部分介绍的是计算图的正向传播（Forward Propagation），下面来介绍其反向传播（Back Propagation），即计算输出对输入的偏导数。

还是上个计算图的例子，输入参数有3个，分别是a, b, c。

首先计算J对参数a的偏导数。从计算图上来看，从右到左，J是v的函数，v是a的函数。则利用求导技巧，可以得到：

$$\frac{\partial J}{\partial a} = \frac{\partial J}{\partial v} \times \frac{\partial v}{\partial a} = 1 \times 3 = 3$$

根据这种思想，然后计算J对参数b的偏导数。从计算图上来看，从右到左，J是v的函数，v是u的函数，u是b的函数。可以推导：

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial v} \times \frac{\partial v}{\partial u} \times \frac{\partial u}{\partial b} = 3 \times 1 \times c = 6$$

最后计算J对参数c的偏导数。仍从计算图上来看，从右到左，J是v的函数，v是u的函数，u是c的函数。可以推导：

$$\frac{\partial J}{\partial c} = \frac{\partial J}{\partial v} \times \frac{\partial v}{\partial u} \times \frac{\partial u}{\partial c} = 3 \times 1 \times b = 9$$

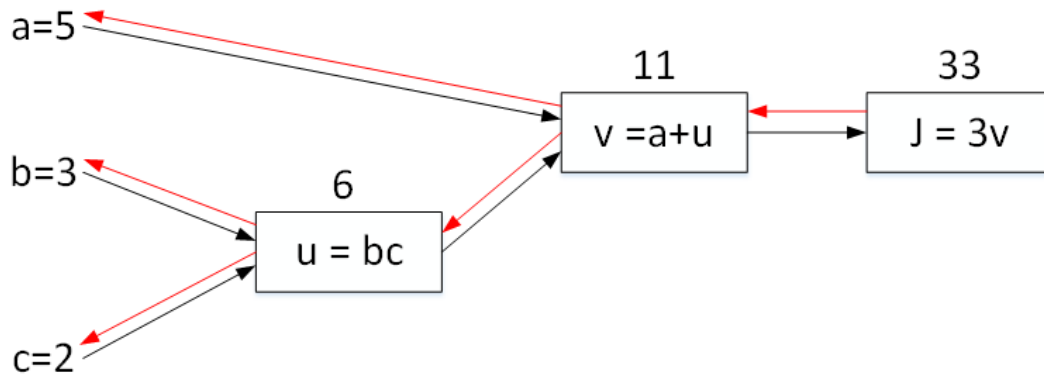
为了统一格式，在程序代码中，使用da, db, dc来表示J对参数a, b, c的偏导数。

$$J(a,b,c) = 3(a+bc) = 3(5+3 \times 2) = 33$$

$$u = bc$$

$$v = a + u$$

$$J = 3v$$



## 9. Logistic Regression Gradient Descent

就单个而言，逻辑回归loss function表达式如下：

$$z = w^T x + b$$

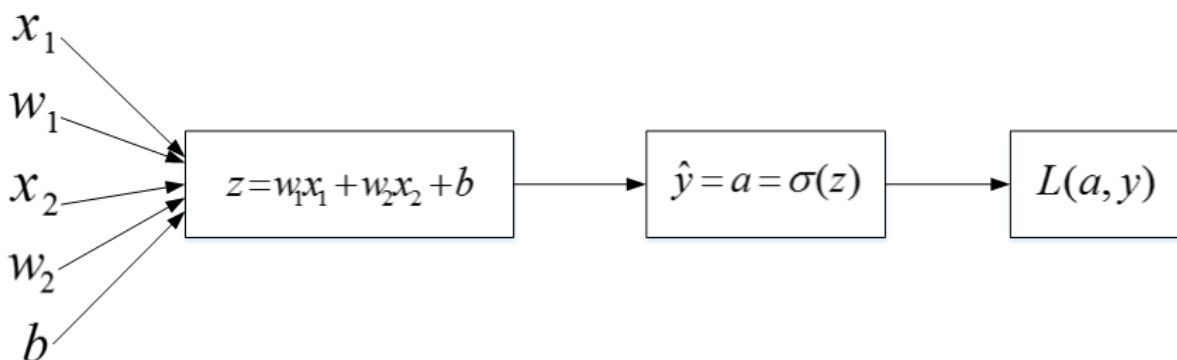
$$\hat{y} = a = \sigma(z)$$

$$L(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$

首先，该逻辑回归的正向传播过程非常简单。

根据上述公式，例如输入样本有两个特征 $(x_1, x_2)$ ，相应的权重 $w$ 维度也是2，即 $(w_1, w_2)$ 。则 $z = w_1 x_1 + w_2 x_2 + b$ ，最后的loss function如下：

## Logistic regression recap



然后，计算该逻辑回归的反向传播过程，即由Loss function计算参数 $w$ 和 $b$ 的偏导数。推导过程如下：

$$da = \frac{\partial L}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$dz = \frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \times \frac{\partial a}{\partial z} = \left(-\frac{y}{a} + \frac{1-y}{1-a}\right) \times a(1-a) = a - y$$

知道了dz之后，就可以直接对 $w_1$ ， $w_2$ 和b进行求导了。

$$dw_1 = \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z} \times \frac{\partial z}{\partial w_1} = x_1 \times dz = x_1(a - y)$$

$$dw_2 = \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial z} \times \frac{\partial z}{\partial w_2} = x_2 \times dz = x_2(a - y)$$

$$db = \frac{\partial L}{\partial b} = \frac{\partial L}{\partial z} \times \frac{\partial z}{\partial b} = 1 \times dz = a - y$$

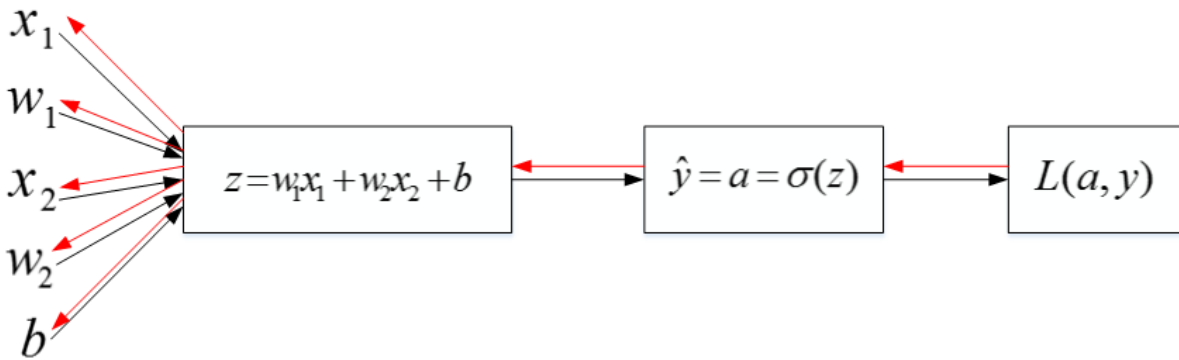
梯度下降算法可表示为：

$$w_1 := w_1 - \alpha dw_1$$

$$w_2 := w_2 - \alpha dw_2$$

$$b := b - \alpha db$$

## Logistic regression derivatives



### 10. Gradient descent on m examples

上一部分讲的是对单个样本求偏导和梯度下降。

如果有m个样本，其Cost function表达式如下：

$$z^{(i)} = w^T x^{(i)} + b$$

$$\hat{y}^{(i)} = a^{(i)} = \sigma(z^{(i)})$$

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

Cost function关于w和b的偏导数可以写成和平均的形式：

$$dw_1 = \frac{1}{m} \sum_{i=1}^m x_1^{(i)} (a^{(i)} - y^{(i)})$$

$$dw_2 = \frac{1}{m} \sum_{i=1}^m x_2^{(i)} (a^{(i)} - y^{(i)})$$

$$db = \frac{1}{m} \sum_{i=1}^m (a^{(i)} - y^{(i)})$$

这样，每次迭代中w和b的梯度有m个训练样本计算平均值得到。其算法流程图如下所示：

```
J=0; dw1=0; dw2=0; db=0;
for i = 1 to m
    z(i) = wx(i)+b;
    a(i) = sigmoid(z(i));
    J += -[y(i)log(a(i))+(1-y(i)) log(1-a(i))];
    dz(i) = a(i)-y(i);
    dw1 += x1(i)dz(i);
    dw2 += x2(i)dz(i);
    db += dz(i);
J /= m;
dw1 /= m;
dw2 /= m;
db /= m;
```

经过每次迭代后，根据梯度下降算法，w和b都进行更新：

$$w_1 := w_1 - \alpha dw_1$$

$$w_2 := w_2 - \alpha dw_2$$

$$b := b - \alpha db$$

这样经过n次迭代后，整个梯度下降算法就完成了。

在上述的梯度下降算法中，我们是利用for循环对每个样本进行dw1，dw2和db的累加计算最后再求平均数的。

在深度学习中，样本数量m通常很大，使用for循环会让神经网络程序运行得很慢。所以，我们应该尽量避免使用for循环操作，而使用矩阵运算，能够大大提高程序运行速度。关于vectorization的内容放在下次笔记中再说。