二分类：

问题描述：
输入一张图片，要求输出 0 或者 1，对应描述“图片中无猫”或者“图片中有猫”。

图片是 RGB 格式的，每个颜色有一个“图片像素”*“图片像素”维度的矩阵。把图片的 R、G、B 三个颜色的矩阵输入到一个列向量中，得到列向量 x。
x=($R_{1,1}$; $R_{1,2}$; …; $R_{1,pixel}$; R21; …; $R_{pixel,pixel}$; $G_{1,1}$; …; $G_{pixel,pixel}$; $B_{1,1}$; …; $B_{pixel,pixel}$)
y=0 或 y=1
这张图片样本就表示为(x, y)，x∈$R^{pixel*pixel*3}$。

可以有 m 个样本，(x1,y1), (x2,y2), …, (xm,ym)。
使用矩阵 X 用来储存输入向量：
X = (x1, x2, …, xm)。这里，X 的维度是(pixel*pixel*3,m)。
在 python 中，可以使用 X.shape 命令，获取矩阵 X 的维度。X.shape = (pixel*pixel*3,m)。

那么输出怎么储存呢？同样使用列向量的形式储存，将输入储存在 Y 矩阵中。
Y = (y1, y2, …, ym)。Y 的维度是(1,m)。
在 python 中，可以使用 Y.shape 命令，获取矩阵 Y 的维度。Y.shape = (1,m)。

X 和 Y 就是样本集。

符号说明：
m：样本数量
nx：pixel*pixel*3

逻辑回归：

输入一张新图片 xt，当使用二分类去判断图中是否有猫的时候，我们希望输出 y 能告诉我们，图中有猫的概率是多少。即 P(y=1|xt)，当输入为 xt 时，y=1 的概率 P 是多少。使用$\hat{y}$表示二分法估计的输出，即 $\hat{y}$=P(y=1|xt)。
P 的数值需要落在(0, 1)区间上，以便更好地表示概率。

函数$\sigma(z) = \frac{1}{1+e^{-z}}$可以把任意 z 映射到(0, 1)范围。当 z 值趋于∞时，σ(z)趋于 1；当 z=0 时，σ(z)=0.5；当 z 值趋于−∞时，σ(z)趋于 0。σ(z)拥有表征概率的性质。

对样本集进行训练的目的其实是找到 $w_{pixel*pixel*3,1}$ 和 b∈R，使得：
($w^T x$+b)能在每一个输入样本的输入下，产生与输入样本对应的输出 y。

加入逻辑回归后，其实训练就是在完成以下任务：
找到合适的 $w_{pixel*pixel*3,1}$ 和 b，使σ($w^T x$+b)=y。
而不是 $w^T x$+b=y。

为什么加入逻辑回归：

逻辑回归 $\sigma(z) = \frac{1}{1+e^{-z}}$ 的加入，**能够增大 $w_{pixel*pixel*3,1}$ 和 b 的变化范围，即参数调节的自由度，也就增大了找到合适参数的可能性**。

二分类的逻辑回归成本函数：

损失函数(Loss Function)用来表示对单个输入数据的预测误差，成本函数(Cost Function)用来表示对所有输入数据总的预测误差。

$\hat{y} = \sigma(w^T x+b)$，where $\sigma(z) = \frac{1}{1+e^{-z}}$ 。

提供训练样本$\{(x^{(1)},y^{(1)}), \cdots, (x^{(m)},y^{(m)})\}$，训练的目的是希望找到合适的 w 和 b，使得每一个$\hat{y}$≈y。即$\hat{y}^{(i)} \approx y^{(i)}$。

二分类的逻辑回归型损失函数定义为：

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

当 y=1 时，$\mathcal{L}(\hat{y}, y) = -log\hat{y}$ ，我们希望损失函数越小越好，所以希望$log\hat{y}$尽可能大，而$\hat{y}\epsilon(0,1)$，所以此时我们希望$\hat{y}$尽可能接近 1。

当 y=0 时，$\mathcal{L}(\hat{y}, y) = -\log(1-\hat{y})$ ，我们希望损失函数越小越好，所以希望$\log(1-\hat{y})$尽可能大，而$\hat{y}\epsilon(0,1)$，所以此时我们希望$\hat{y}$尽可能接近 0。

当然，损失函数的定义不是唯一的，完全可以使用误差的平方来定义损失，比如下列形式：

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

但是定义为这种形式的误差函数并非凸函数，它有很多局部极值。逻辑回归型的损失函数是凸函数，只有一个极值，即全体极值。凸函数形式的损失函数能够保证损失达到最小，而不是局部极小。

二分类的逻辑回归型成本函数定义为：

$$\mathcal{J}(w, b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

将定义的二分类逻辑回归型损失函数代入到成本函数中，可以得到：

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

以上是逻辑回归(logistic)算法的过程。

“对于有 m 个样本的训练集，想要遍历每个样本，你可能会写 for 循环。但在实现神经网络的过程中个，如果你想遍历整个训练集，并不需要使用显式的 for 循环。”

梯度下降法寻找凸型成本函数最值：

凸函数只有一个极值，即整体最值。
我们希望找到 w 和 b，能让这样的 w 和 b 配置在预测所有的样本时产生的预测误差最小。
即寻找 w 和 b，使成本函数最小。
由于构造的逻辑回归型成本函数是凸函数，所以成本函数的函数值具有单调性。因此，在成本函数的自变量取值范围的任意位置处开始使用梯度下降法，均能最终抵达成本函数的最值。

考虑成本函数只有一个自变量的情况。J=J(w)。
（“:=”表示更新，其实就是赋值。）
（α称为学习率，当只有一个自变量时，其实就是自变量移动的步长大小。）
Repeat{

$w := w - \alpha \frac{dJ(w)}{dw}$ %python 中，用 dw 表示导数 dJ/dw。左式写作：w := w - α dw

}

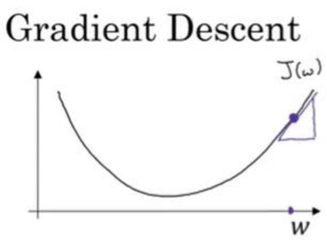

FIGURE. 示意图 1-单个自变量的凸函数的梯度下降法

考虑成本函数有两个自变量的情况。J=J(w,b)。
（当有多个自变量时，因变量对某一个自变量的微分符号 d 写成偏微分符号∂的形式。只是把 d 写成了花体 d，含义上并没有变化。）
Repeat{

$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$ %python 中，用 dw 表示导数 dJ/dw。左式写作：w := w - α dw

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

}

以上是梯度下降法的实现过程。(未含赋初值和收敛性判断条件。)

单个样本的损失函数形式说明：

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

因为：

当 y=1 时，假设二分类的输出是$\hat{y}$

那么，当 y=0 时，输出应该是（1-$\hat{y}$）

所以可以构造 $\hat{y}^{y}\,(1-\hat{y})^{1-y}$，当 y=0 或 1 时，能够满足上述条件。Log 是单调递增函数，可以对其取 log，并不影响原来函数的单调性。

更具体的内容，参看凸函数的理论。