前向传播和后向传播简介：

前向传播用于计算成本函数的数值。后向传播用于计算成本函数的导数。
举个简单的例子，假设成本函数 J(a,b,c)=3(a+bc)。
设一些过程变量：

u=bc

v=a+u

J=3v

则计算成本函数 J 的过程可以表示为下图中的黑色和蓝色部分：

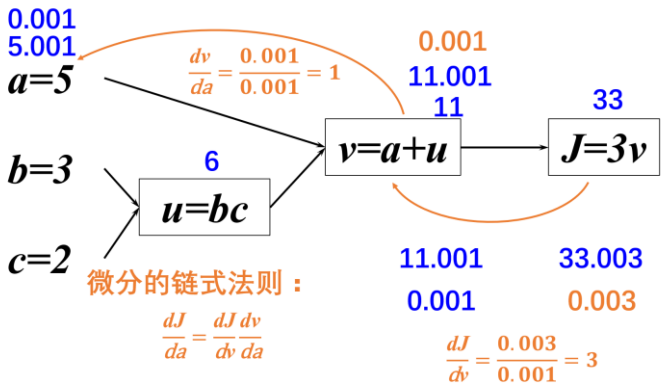


FIGURE. 示意图 1-导数的后向传播

求微分的过程，是通过橙色线条，将因为输入的波动而得到的输出的变化反馈给输入，从而完成导数的计算。此即为反向传播过程。不相连的单元间的偏导计算，将通过链式法则来计算。

在 python 中，通过链式求导法则的计算，成本函数对所有变量的偏导信息储存在 dvar 中。在上图中，dJ/dv 将被表示为 dv，dJ/da 将被表示为 da。

提供一个样本，如何使用逻辑回归成本函数的梯度下降法，求解对该样本合适的 w 和 b？

在二分类中提到过：

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

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

为了使流程更加清晰，增加过程变量来描述上述关系：

$z = w^T x+b$

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

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

可以绘图如下：

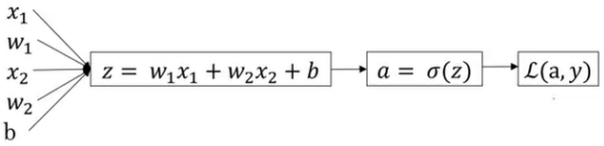


FIGURE. 示意图 2-编程的根据，很重要的一幅图

使用梯度下降法求解凸型成本函数的最值，需要知道成本函数对各个输入系数 w_1, w_2, ⋯, $w_{pixel*pixel*3}$、b 的偏导。

以求$\frac{\partial L}{\partial w_1}$为例：

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w_1}$$

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

$$\frac{\partial a}{\partial z} = \frac{1}{(1+e^{-z})^2 e^{-z}} = a(1-a)$$

$$\frac{\partial L}{\partial z} = a - y$$

$$\frac{\partial z}{\partial w_1} = x_1$$

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

同理可以求出$\frac{\partial L}{\partial w_2}$, $\frac{\partial L}{\partial w_3}$, ⋯, $\frac{\partial L}{\partial w_{pixel*pixel*3}}$, $\frac{\partial L}{\partial b}$。

（技巧：先计算到$\frac{\partial L}{\partial z}$，因为它的公式很简单，是$(a-y)$，a 是已经得到的经过逻辑回归后的$\hat{y}$，一个样本的 y 是个常数。所以你在给任何一个变量求偏导时，不管这个变量是 $w_1, w_2, \cdots, w_{pixel*pixel*3}$ 还是 b，y 都不会变。然后再$\frac{\partial}{\partial var}$，z 对 wi 求偏导是 xi，z 对 b 求偏导是常数 1。）

在 python 中，以上只是一些文本代码，dw1, dw2, ⋯, dwpixel*pixel*3, db 就能得到成本函数对各个变量的偏导。然后写一个循环：

```
w1 := w1 - α dw1
w2 := w2 - α dw2
b := b - α db
……
```

就可以使用梯度下降法寻找成本函数的极值了。当找到成本函数的极值时，这时的 w 和 b 就是需要的 w 和 b。

提供整个样本集，如何求解对样本集合适的 w 和 b？

（使用全体样本集对某个参数的平均梯度，作为在整个样本集上使用梯度下降法时，该参数的下降梯度。）

沿用梯度下降法求解单个样本的 w 和 b 时的符号。参照二分类问题中的数学描述，样本集$\left(x^{(i)}, y^{(i)}\right)$的成本函数表示为：

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

$$\frac{\partial \mathcal{J}(w,b)}{\partial w_1} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial w_1}\mathcal{L}\left(\mathrm{a}^{(i)}, y^{(i)}\right)$$

$\frac{\partial}{\partial w_1}\mathcal{L}\left(\mathrm{a}^{(i)}, y^{(i)}\right)$就是第 i 个样本的损失函数的对 w1 的偏导。在 python 的命令中，就是 dw1。

根据样本集，求解合适于样本集的 w 和 b 参数，其中梯度下降法的编程实现：

（目的：用一个 for 循环遍历训练集，计算每个训练样本中个变量的偏导数，分别加起来，求平均值。）（监视：样本集的平均成本函数）

J=0; dw1=0; dw2=0; db=0
for i = 1 : m
    z(i)=w^Tx(i)+b
    a(i)=sigmoid(z(i))
    J = -(y(i) loga(i) + (1 - y(i)) log(1 – a(i))
    dz(i) = a(i) – y(i)
    dw1 += dz x(1)
    dw2 += dz x(2)
    db += dz
J /= m
dw1 /= m
dw2 /= m
db /=m

(这时，$\frac{\partial \mathcal{J}}{\partial w_1}$ = dw1　$\frac{\partial \mathcal{J}}{\partial w_2}$ = dw2　$\frac{\partial \mathcal{J}}{\partial b}$ = db)

w1 := w1 – α dw1
w2 := w2 – α dw2
b := b – α db

以上只在数据集上使用了一步梯度下降法。并且假设每个样本中只有两个元素（就是两个特征），所以对应地，只有 w1 和 w2。如果单个样本有很多特征（元素），则需要在上述代码内的对应位置处增加循环遍历单个样本内的所有特征（元素）。
如果想在数据集上进行多步的梯度下降，则需要在上述代码外增加循环，可以进行多步的梯度下降法。

当循环嵌套太多时（一般超过 2 层就算多），程序的效率会下降。机器学习的特征非常之多，使用显式循环将影响计算速度。因此，通过使用非显式循环来提升计算效率，变得十分重要。有多种方法可以帮助消除显式循环，向量化方法（vectorization）是其中之一。

向量化方法和 NUMPY：

已知：

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{pixel*pixel*3} \end{bmatrix},\ x = \begin{bmatrix} x \\ x_2 \\ \vdots \\ x_{pixel*pixel*3} \end{bmatrix}$$

计算：

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

使用 for 循环的实现：

z = 0

for i = 1 : $pixel * pixel * 3$

    z += w(i)x(i)

z += b

使用向量化的实现：

z = np.dot(w,x) + b

（可以在 jupyter notebook 上运行代码）（jupyter 是在 CPU 上运行的）（GPU 同样可以做，不过 GPU 更擅长 SIMO。Sigle instruction multiple data：单条指令，多条数据）

使用向量化方法，对样本集的成本函数最值进行梯度下降法求解。

J = 0; dw=np.zeros(nx,1);

for i = 1 : m

    z(i) = (w, x(i))+b

    a(i) = sigmoid(z(i))

    J += -(y(i)loga(i) + (1-y(i)) log(1-a(i))))

    dz(i) = a(i) – y(i)

    dw += dz(i)x(i)

    db += dz(i)

dw /= m

db /= m

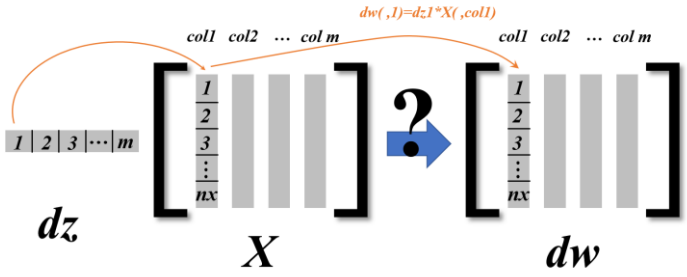
w += -α dw

b += -α db

如何一个 FOR 循环都不使用，同时完成梯度下降法求解成本函数最值的问题？

（样本集共用一套 w1 标量，w2 标量，…，wnx 标量）（nx=pixel*pixel*3）

1. Z = X'w+[b; b; …; b]　　%列向量 z 的每行储存着每个样本的预测输出 $\hat{y}$
2. A = sigmoid(Z)
3. L=-(Y 和 logA 的对应项相乘得到新的同维行向量+（1-Y）和 log(1-A)对应项相乘得到新的同维行向量）　　%行向量 L 中的各项分别对应着各个样本的损失函数 $L_1, L_2, \cdots, L_m$。
4. dz = A – Y　　%A、Y 都是行向量
5. dw = dz 的每个元素以乘法作用到 X 的每一列上　%dw=(标量 dz1*X 的第一列，dz2*X 的第二列，…，dzm*X 的第 m 列)　　（每一列的 nx 个数字，分别对应每个样本上的每个特征的下降梯度）　　（dw 是 nx*m 维矩阵）



取出向量中的每个元素，分别和矩阵各列相乘，缩放后得到新的列向量，组成新矩阵。但是，找不到实现这种功能的矩阵运算。

6. 对 dw 和 dz 分别按列取平均，得到两个列向量，前者产生的维度是 nx*1，后者产生的维度是 1*1。这两个列向量，就是在对样本集进行如 FIGURE. 示意图 2 所示的二分类时，使用梯度下降法，求解样本集的成本函数最值的过程中，在经过对全部样本集的一步计算后，产生的作用在参数样本集共用的 w1，w2，…，wnx 和 b 上的平均下降梯度。
7. w += -αdw
8. b += -α db

上述步骤完成后，进行下一步梯度下降，直至参数 w1，w2，…，wnx 和 b 的变化达到收敛条件。

讨论和问题解决：

在第 5 步中，找不到所描述的那种矩阵运算，所以在第 5 步中获得 dw 有些费力。
如果把第 5 步和第 6 步绑定在一起，有没有那种矩阵运算可以综合实现第 5 步和第 6 步中处理 dw 的流程呢？
考虑第 5 步要实现的功能：dw=(dz_1*X(,1), dz_2*X(,2), ⋯, dz_m*X(,m))
考虑第 6 步要实现的功能：dw=(dz_1*X(,1) + dz_2*X(,2) + ⋯ + dz_m*X(,m))/m，又可以分解为先求和，再取平均
第 5 步的功能和第 6 步中的求和可以综合为：(X(,1), X(,2), ⋯, X(,m)) (dz_1; dz_2; ⋯; dz_m)，即 Xdz^T。所以第 5 步和第 6 步中，关于 dw 的运算可以总结为$\frac{1}{m}\mathrm{X}\,\mathrm{dz}^T$。

所以，上述的步骤可以改进为：

1. Z = np.dot(w', X) + b　　% Z 是行向量，每列储存着每个样本的预测输出 $\hat{y}$
2. A = sigmoid(Z)
3. dz = A – Y　　%A、Y 都是行向量
4. dw=$\frac{1}{\mathrm{m}}$np.dot(X,dz^T)　　%dw 是列向量，nx*1
5. db=$\frac{1}{\mathrm{m}}$np.sum(dz)
6. w += -α dw
7. b += -α db

上述步骤完成后，就以向量的形式完成了一步梯度下降。
如果想要实现多步的梯度下降，任然需要使用 for 循环控制梯度下降的步数。
以循环的方式，不断进行梯度下降，直至参数 w1，w2，⋯，wnx 和 b 的变化达到收敛条件。