吴恩达《神经网络与深度学习》网络课

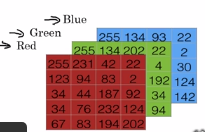
**2019.1.19——二次观看笔记总结**

第二周：

**Logistic回归**

Binary Classification:**二分分类**

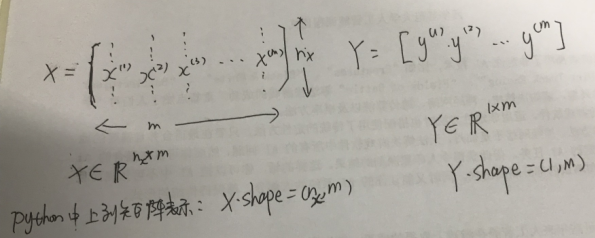
一张图片可看为矩阵：



样本的表示：

（x,y） x是nx维的特征向量， y

m training example,,... 



**二元分类：**

Given X,want 

X 0 <=<=1

Parameters(参数)：w

Output (预测输出值 )= —>z= （sigmoid激活函数） 

If z large  =1

If z large negative number 

**Logistics Regression cost function:**

Given ,want (预测输出值) （实际值）

Loss(error)function（损失函数来衡量预测输出值与实际值有多接近）:L(,y)= 损失函数可用平方差，但是梯度下降法不太好用，所以有个和平方差法相似的损失函数，作用

**L(,y)=**

If y=1: <— want  large, want  large

If y=0: <— want  large, want  small

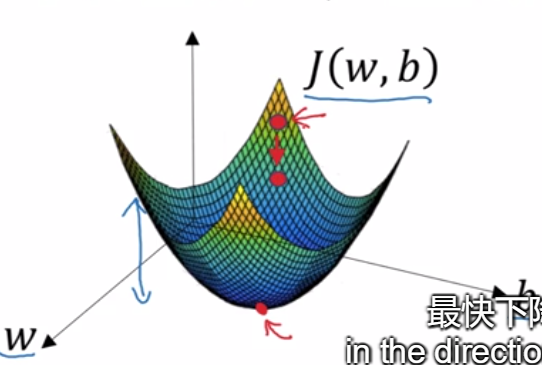
Cost Function:

**梯度下降法：**





Want to find w,b that minimize J(w,b)



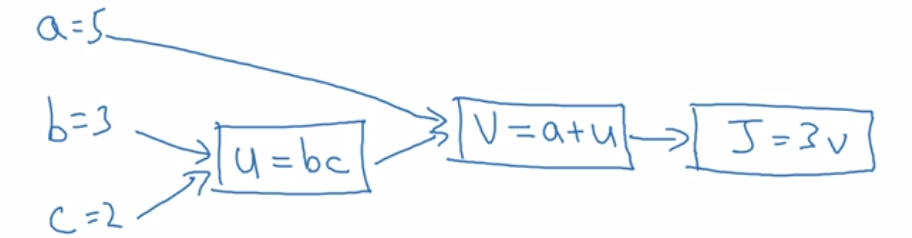
Repeat {

 (偏导)

}是学习率

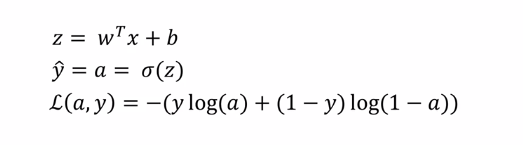
Computation Graph（计算图）

J(a,b,c)=3(a+bc)

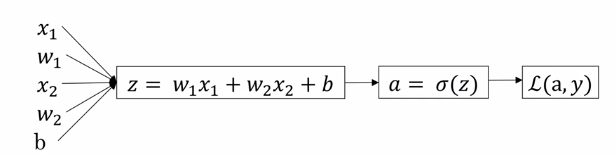


**Logistic回归的梯度下降法：**

以单个样本为例：上面所学公式



公式对应的计算图：



**反向传播：**

在编程中设“da” =

“dz”= ()\*a(1-a) =a-y

“dw1”= ；“dw2”=；“db”=dz



**m个样本的Logistic回归的梯度下降法：**



求



代码格式：



For i=1 tom:

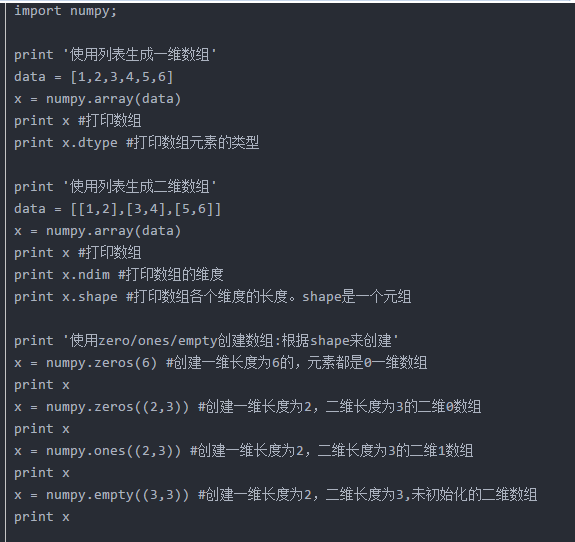


**向量化（vectorization）**np.dot()计算两个数组的乘积。对于二维数组来说，dot()计算的结果就相当于矩阵乘法。 z=代码的向量化表示：z=np.dot(w,x)+b

numpy矩阵相关知识：

为了加快深度学习神经网络运算速度，可以使用比CPU运算能力更强大的GPU。事实上，GPU和CPU都有并行指令（parallelization instructions），称为Single Instruction Multiple Data（SIMD）。SIMD是单指令多数据流，能够复制多个操作数，并把它们打包在大型寄存器的一组指令集。SIMD能够大大提高程序运行速度，例如python的numpy库中的内建函数（built-in function）就是使用了SIMD指令。相比而言，GPU的SIMD要比CPU更强大一些。



np.arange()函数分为一个参数，两个参数，三个参数三种情况   
1）一个参数时，参数值为终点，起点取默认值0，步长取默认值1。   
2）两个参数时，第一个参数为起点，第二个参数为终点，步长取默认值1。   
3）三个参数时，第一个参数为起点，第二个参数为终点，第三个参数为步长。其中步长支持小数。

例子：

用法：zeros(shape, dtype=float, order='C')

返回：返回来一个给定形状和类型的用0填充的数组；

参数：shape:形状

dtype:数据类型，可选参数，默认numpy.float64

U=np.zeros((n,1))

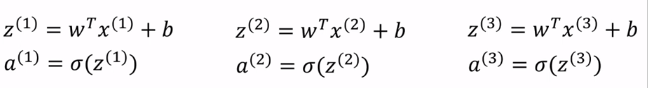
For i in range(n):

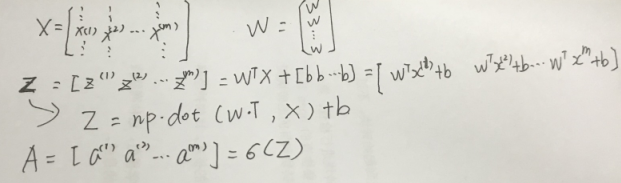
U[i]=math.exp(v[i])

向量化后代码

*import* numpy *as* np  
*import* time  
n=100  
tic = time.time()  
v=np.arange(1,100)  
u=np.zeros((n,1))  
u1=np.exp(v)  
toc = time.time()  
#u2=np.log(v)  
#u3=np.abs(v)  
#u4=np.maximum(v,0)  
print(u1)  
print("Time" + str((toc - tic)\*1000) + 'ms')  
#print(u2)  
#print(u3)  
#print(u4)

**logistics回归函数向量化思想：**





非向量化，logistics回归梯度下降法（一次迭代）



For i=1 tom:





向量化

Z=np.dot(w.T,x)+b

A=sigmoid(Z)

dZ=A-Y

dw=1/m\*np.dot(X,dz.T)

db=1/m\*np.sum(dZ)

w:=w-alpha\*dw

b:=b-alpha\*db

其中，alpha是学习因子，决定w和b的更新速度。上述代码只是对单次训练更新而言的，外层还需要一个for循环，表示迭代次数