吴恩达《神经网络与深度学习》网络课

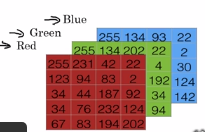
**2019.1.19——二次观看笔记总结**

第二周：

**Logistic回归**

Binary Classification:**二分分类**

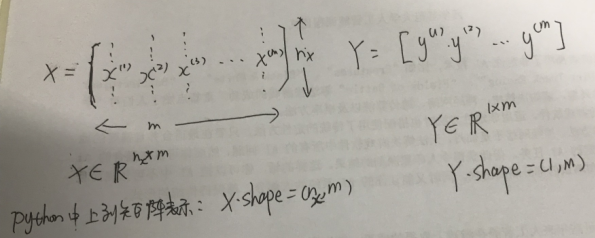
一张图片可看为矩阵：



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

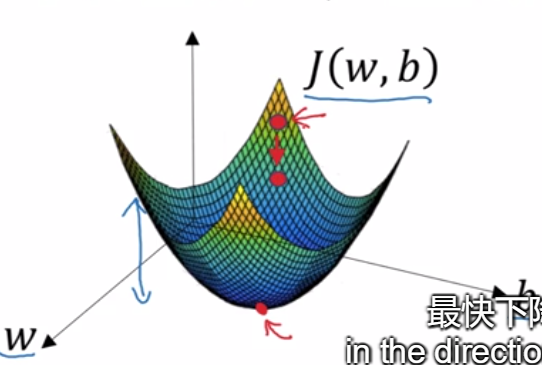
Cost Function:

**梯度下降法：**





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



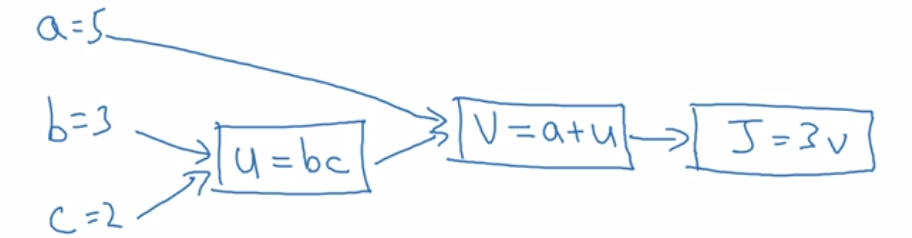
Repeat {

 (偏导)

}是学习率

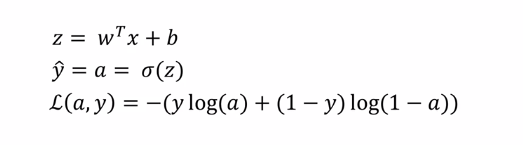
Computation Graph（计算图）

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

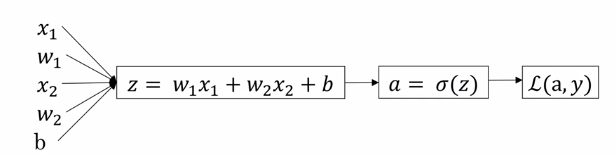


**Logistic回归的梯度下降法：**

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:

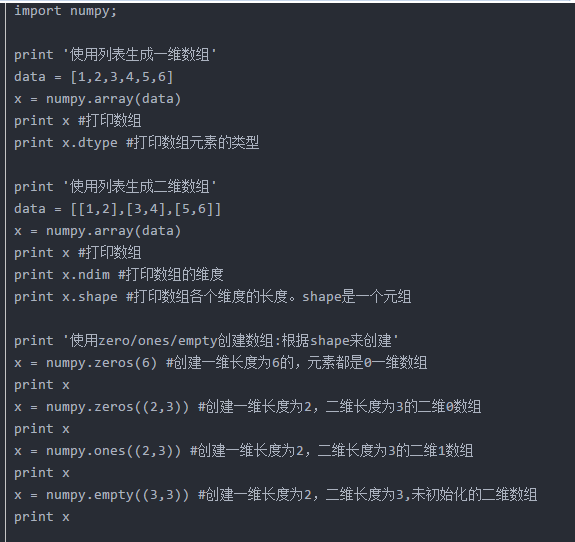


**向量化（vectorization）**np.dot()计算两个数组的乘积。对于二维数组来说，dot()计算的结果就相当于矩阵乘法。 z=代码的向量化表示：z=np.dot(w,x)+b

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))

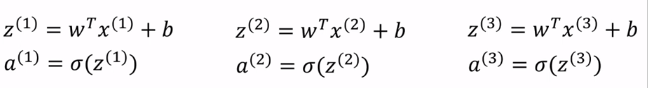
For i in range(n):

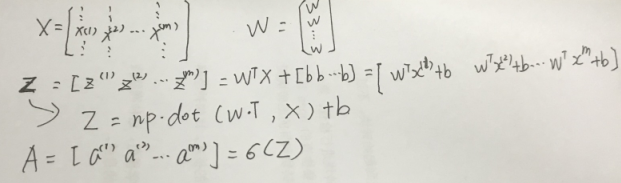
U[i]=math.exp(v[i])

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

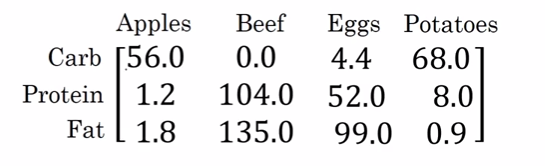
b:=b-alpha\*db

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

**2019.1.20总结**

**Python广播**

例题，Calories from carb,protein,fat in 100g of different foods:



代码：

**import numpy as np**

**A=np.array( [[56.0,0.0,4.4,68.0],**

**[1.2,104.0,52.0,8.0],**

**[1.8,135.0,99.0,0.9]])**#array()函数加入数值后生成了矩阵

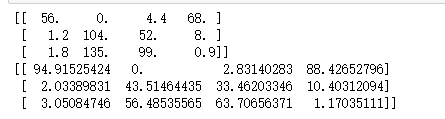
**print(A)**

**cal=A.sum(axis=0)**#求和，axis=0表示竖直相加，axis=1表示水平相加

**percentage = 100\*A/(cal.reshape(1,4))**”””reshape()函数将cal变成1x4的矩阵（其实这里可以不用，因为cal已经是了，但是为了养成习惯，便于改变，加上为好）reshape()函数是数组对象中的用法，用于改变数组的形状，确保矩阵或向量是需要的维度。”””

**print(percentage)**

Generated result(生成的结果)：



 在python中计算矩阵与常数的加法，python会自动把常数复制补全成对应的矩阵

 同理复制补全

复制m次

mxn

fu

**m**xn

1xn

 同理复制补全

mx**n**

复制m次

Mx1

mxn

**（+，-，\*，/）都同理python可复制补全成对应矩阵**

**Python/numpy vectors(numpy 向量)：**

生成矩阵代码：

**import numpy as np**

**a=np.random.randn(5)**

**print(a)**

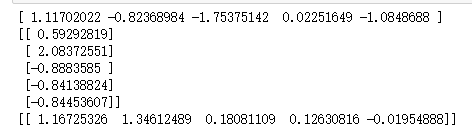
**a=np.random.randn(5,1)**

**print(a)**

**a=np.random.randn(1,5)**

**print(a)**

a=np.random.randn(5)这条语句生成的a的维度是（5，）。它既不是行向量也不是列向量，我们把a叫做rank 1 array。这种定义会带来一些问题。例如我们对a进行转置，还是会得到a本身。所以，如果我们要定义（5，1）的列向量或者（1，5）的行向量，最好使用下来标准语句，a=np.random.randn(5,1)；a=np.random.randn(1,5) 避免使用rank 1 array。



**assert(a.shape ==(5,1))**# 使用assert语句对向量或数组的维度进行判断, assert会对内嵌语句进行判断，即判断a的维度是不是（5，1）的。如果不是，则程序在此处停止。使用assert语句也是一种很好的习惯，能够帮助我们及时检查、发现语句是否正确。

**a.reshape((5,1))**# 使用reshape函数对数组设定所需的维度

logisitic损失函数解释:

= when 

Interpret 

If y=1  If y=0 

由上面可得：

由于log函数的单调性，可以对上式P(y|x)进行log处理：

我们希望上述概率P(y|x)越大越好，对上式加上负号，则转化成了单个样本的Loss function，越小越好，也就得到了我们之前介绍的逻辑回归的Loss function形式。

如果对于所有m个训练样本，假设样本之间是独立同分布的（iid），我们希望总的概率越大越好：

p（labels in training set）= 同样引入log函数，加上负号，将上式转化为Cost function：

log p(labels in training set)=log



把上式中的“-”去掉，因为我们求最小化成本，不需要直接用最大似然估计，为了方便，对成本函数加上1/m进行适当缩放

Cost function:

Maimun likelihood estamation(最大似然估计)即求出一组参数，使这个式子取最大值



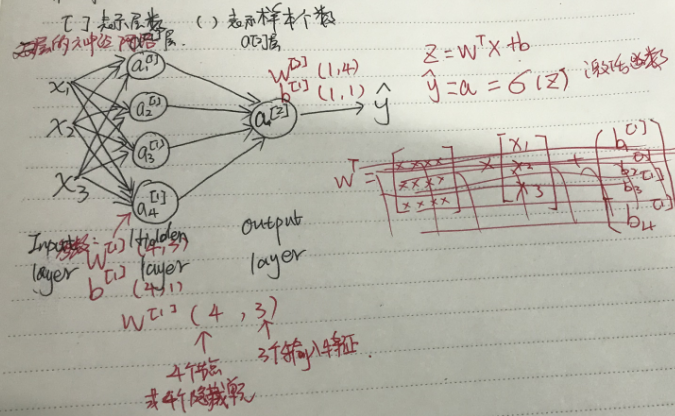
= 

**第三周：Neural Network**

[i]表示层数 (i)表示样本个数

Neural Network Representation(表示法)

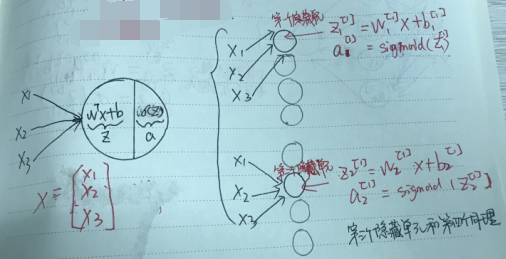
**一个样本**

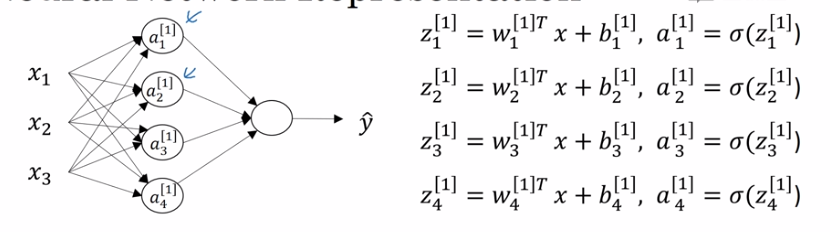


**2019.1.20总结**

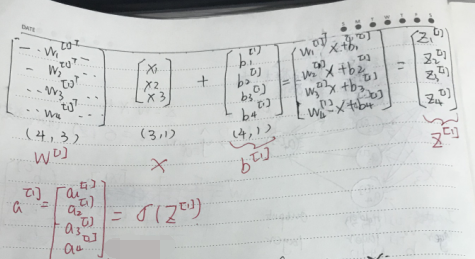
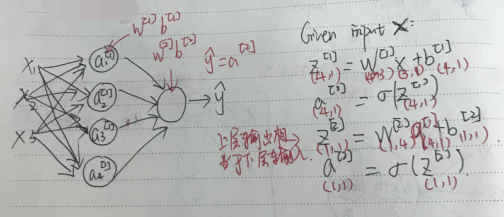
**第三周**

一个节点的计算





对应矩阵形式的运算

**m个训练样本**

for i=1 to m:#（双层NN训练m个样本）







X



X



X

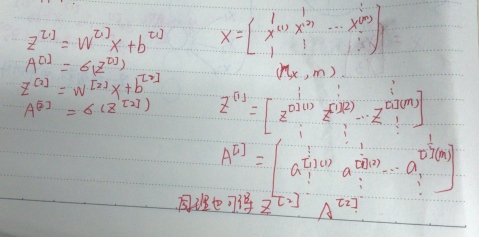


X

**...**

[2]表示第二层

(i)表示第i个样本



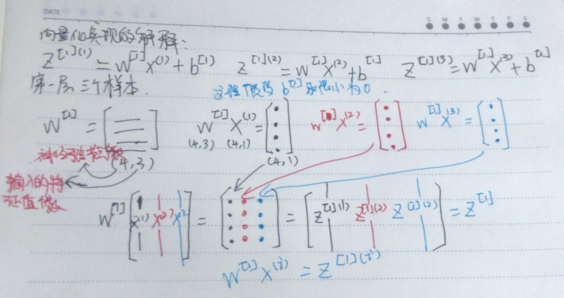
上述矩阵中从左往右横向看表示：表示不同样本

从上往下竖向看表示：神经网络中不同节点

横向矩阵A扫过不同的训练样本

竖向是矩阵A中的不同指标

**向量化实现解释：**



**Activation functions(激活函数)**

Sigmoid function 几乎已经不用了，tanh( )函数比sigmoid 函数好用，函数平均值更接近0。tanh()函数适合于任何场合，但是在输出层中不适合，因为如果y为1或者0，在0到1·之间而不是-1到1之间。（在二元分类时，输出层可用sigmoid函数）

Sigmoid 和tanh 函数都有一个缺点，如果z非常大或非常小，那么导数的梯度（函数的斜率）可能很小，很接近0，这样可能会拖慢梯度算法速度。

