向量化的更多例子

一些经验性的法则

- 在编写新的神经网络或者进行回归时，尽量避免for循环

举例

向量$\vec{u} = A\vec{v}$的计算相当于

$$u_i = \sum_{j=1}^{n} A_{ij} v_i$$

如果使用**for循环**,就会产生两个嵌套

```
for i in range(n)
    for j in range(n)
        u[i] += A[i][j]*v[j]
```

显然这样的运行效率比较低下
但如果我们选择向量化计算

```
u = np.dot(A,v)
```

很明显代码更加简洁，运行效率也更加高效

假设现在内存中有一个向量$\vec{v}$,并且对其进行指数运算,就像

$$u_i = e^{v_i}$$

对于非向量化的运算,就有

```
u = np.zeros((n,1))
for i in range(n)
    u[i] = math.exp(v[i])
```

而向量化的实现

```
u = np.exp(v)
```

还有很多例子,并且这些运算都是对于向量$\vec{v}$中的每一个元素进行运算，比如

```
np.log(v)
np.abs(v)
np.maximum(v,0)
v**2
1/v
```

所以，每当我们想要使用**for循环**时，应当检查能不能使用numpy的内置函数进行计算

针对于logistic回归

首先根据前面的章节推导出的过程,我们可以知道
我们在进行梯度下降时会有两个循环

我们先简化掉**第二个for循环**也就是针对于每个样本中所有参数分别相加的循环

```
dw = np.zeros((n,1))
dw += x[i]*dz[i]
dw /= m
```

这样，我们的代码就从两个for循环简化为了一个for循环