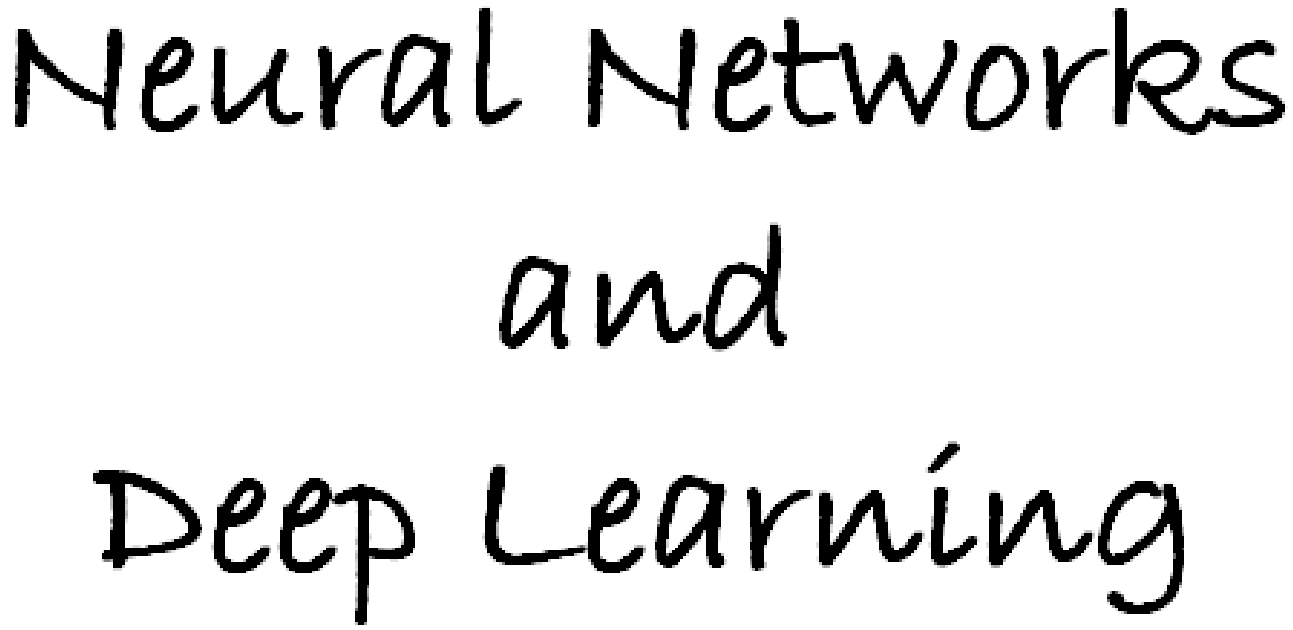


Coursera吴恩达《神经网络与深度学习》课程笔记（3）-- 神经网络基础之Python与向量化

红色石头 · 3 个月前

> **我的CSDN博客地址：红色石头的专栏**
> **我的知乎主页：红色石头**
> **我的知乎专栏：红色石头的机器学习之路**
> **欢迎大家关注我！共同学习，共同进步！**

上节课我们主要介绍了逻辑回归，以输出概率的形式来处理二分类问题。我们介绍了逻辑回归的Cost function表达式，并使用梯度下降算法来计算最小化Cost function时对应的参数w和b。通过计算图的方式来讲述了神经网络的正向传播和反向传播两个过程。本节课我们将来探讨Python和向量化的相关知识。

深度学习算法中，数据量很大，在程序中应该尽量减少使用loop循环语句，而可以使用向量运算来提高程序运行速度。

向量化（Vectorization）就是利用矩阵运算的思想，大大提高运算速度。例如下面所示在Python中使用向量化要比使用循环计算速度快得多。

```
import numpy as np
import time

a = np.random.rand(1000000)
b = np.random.rand(1000000)

tic = time.time()
c = np.dot(a,b)
toc = time.time()

print(c)
print("Vectorized version:" + str(1000*(toc-tic)) + "ms")

c = 0
tic = time.time()
for i in range(1000000):
        c += a[i]*b[i]
toc = time.time()

print(c)
print("for loop:" + str(1000*(toc-tic)) + "ms")
```

输出结果类似于：

```
250286.989866
Vectorized version:1.5027523040771484ms
250286.989866
For loop:474.29513931274414ms
```

从程序运行结果上来看，该例子使用for循环运行时间是使用向量运算运行时间的约300倍。因此，深度学习算法中，使用向量化矩阵运算的效率要高得多。

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。

2. More Vectorization Examples

上一部分我们讲了应该尽量避免使用for循环而使用向量化矩阵运算。在python的numpy库中，我们通常使用np.dot()函数来进行矩阵运算。

我们将向量化的思想使用在逻辑回归算法上，尽可能减少for循环，而只使用矩阵运算。值得注意的是，算法最顶层的迭代训练的for循环是不能替换的。而每次迭代过程对J，dw，b的计算是可以直接使用矩阵运算。

3. Vectorizing Logistic Regression

在《神经网络与深度学习》课程笔记（2）中我们介绍过，整个训练样本构成的输入矩阵X的维度是（n_x，m），权重矩阵w的维度是（n_x，1），b是一个常数值，而整个训练样本构成的输出矩阵Y的维度为（1，m）。利用向量化的思想，所有m个样本的线性输出Z可以用矩阵表示：

$$Z = w^T X + b$$

在python的numpy库中可以表示为：

```
A = sigmoid(Z)
```

其中，w.T表示w的转置。

这样，我们就能够使用向量化矩阵运算代替for循环，对所有m个样本同时运算，大大提高了运算速度。

4. Vectorizing Logistic Regression's Gradient Output

再来看逻辑回归中的梯度下降算法如何转化为向量化的矩阵形式。对于所有m个样本，dZ的维度是（1，m），可表示为：

$$dZ = A - Y$$

db可表示为：

$$db = \frac{1}{m}\sum_{i=1}^{m} dz^{(i)}$$

对应的程序为：

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

dw可表示为：

对应的程序为：

```
dw = 1/m*np.dot(X,dZ.T)
```

这样，我们把整个逻辑回归中的for循环尽可能用矩阵运算代替，对于单次迭代，梯度下降算法流程如下所示：

```
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。

5. Broadcasting in Python

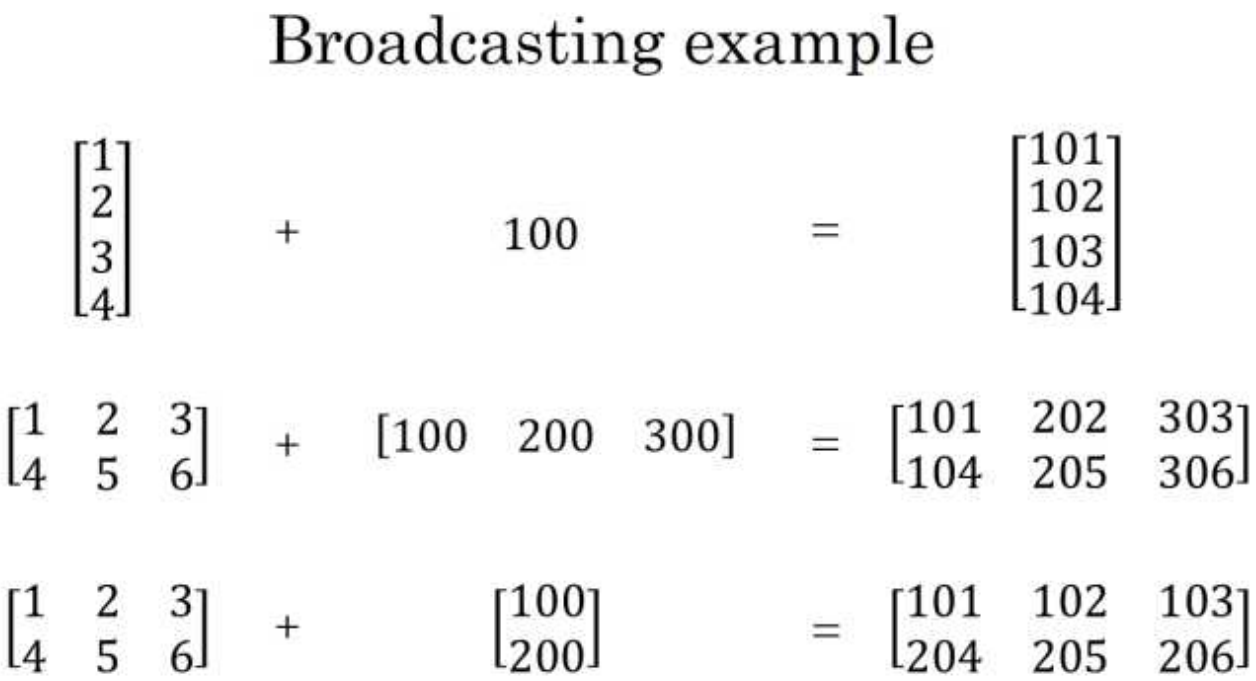
下面介绍使用python的另一种技巧：广播（Broadcasting）。python中的广播机制可由下面四条表示：

- **让所有输入数组都向其中shape最长的数组看齐，shape中不足的部分都通过在前面加1补齐**
- **输出数组的shape是输入数组shape的各个轴上的最大值**

算，否则出错

- **当输入数组的某个轴的长度为1时，沿着此轴运算时都用此轴上的第一组值**

简而言之，就是python中可以对不同维度的矩阵进行四则混合运算，但至少保证有一个维度是相同的。下面给出几个广播的例子，具体细节可参阅python的相关手册，这里就不赘述了。



值得一提的是，在python程序中为了保证矩阵运算正确，可以使用reshape()函数来对矩阵设定所需的维度。这是一个很好且有用的习惯。

6. A note on python/numpy vectors

接下来我们将总结一些python的小技巧，避免不必要的code bug。

python中，如果我们用下列语句来定义一个向量：

```
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，避免使用rank 1 array。

```
a = np.random.randn(5,1)
b = np.random.randn(1,5)
```

除此之外，我们还可以使用assert语句对向量或数组的维度进行判断，例如：

```
assert(a.shape == (5,1))
```

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。

另外，还可以使用reshape函数对数组设定所需的维度：

```
a.reshape((5,1))
```

7. Quick tour of Jupyter/iPython Notebooks

Jupyter notebook（又称IPython notebook）是一个交互式的笔记本，支持运行超过40种编程语言。本课程所有的编程练习题都将在Jupyter notebook上进行，使用的语言是python。

关于Jupyter notebook的简介和使用方法可以看我的另外两篇博客：

Jupyter notebook入门教程（上）

Jupyter notebook入门教程（下）

在上一节课的笔记中，我们介绍过逻辑回归的Cost function。接下来我们将简要解释这个Cost function是怎么来的。

首先，预测输出 $\hat{y}$ 的表达式可以写成：

$$haty = \sigma(w^T x + b)$$

其中， $\sigma(z) = \frac{1}{1+exp(-z)}$ 。 $\hat{y}$ 可以看成是预测输出为正类（+1）的概率：

$$\hat{y} = P(y=1|x)$$

那么，当y=1时：

$$p(y|x) = \hat{y}$$

当y=0时：

$$p(y|x) = 1 - \hat{y}$$

我们把上面两个式子整合到一个式子中，得到：

由于log函数的单调性，可以对上式P(y|x)进行log处理：

$$log\ P(y|x) = log\ \hat{y}^{y}(1-\hat{y})^{(1-y)} = y\ log\ \hat{y} + (1-y)log(1-\hat{y})$$

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。

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

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：

$$max \prod_{i=1}^{m} P(y^{(i)}|x^{(i)})$$

同样引入log函数，加上负号，将上式转化为Cost function：

$$J(w,b) = -\frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\ log\ \hat{y}^{(i)} + (1-y^{(i)})log(1-\hat{y}^{(i)})$$

上式中，$\frac{1}{m}$ 表示对所有m个样本的Cost function求平均，是缩放因子。

9. Summary