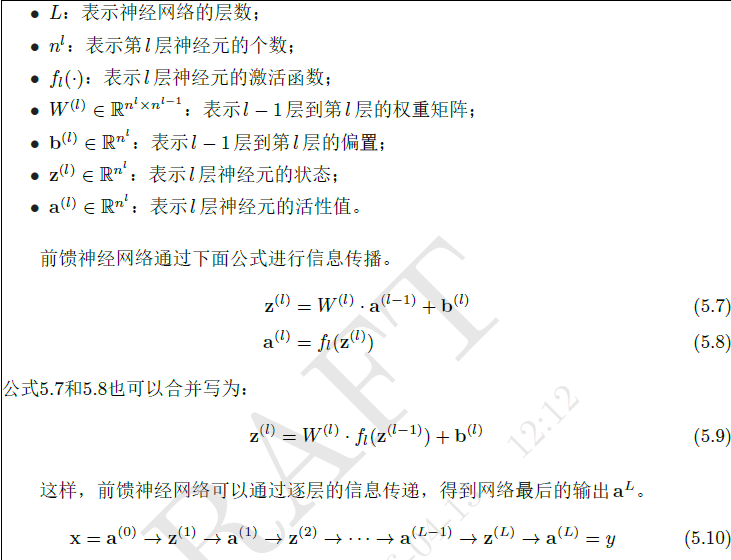
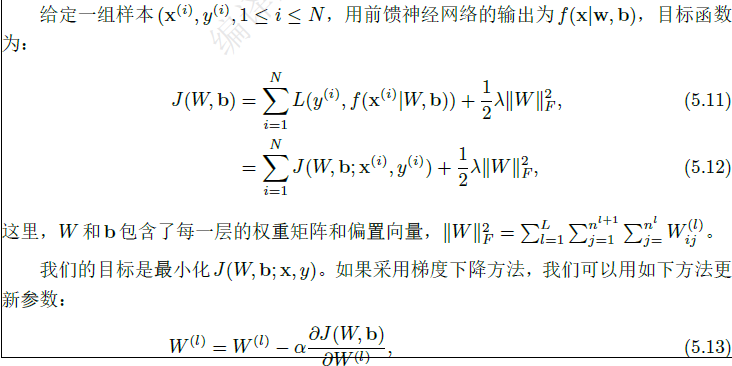
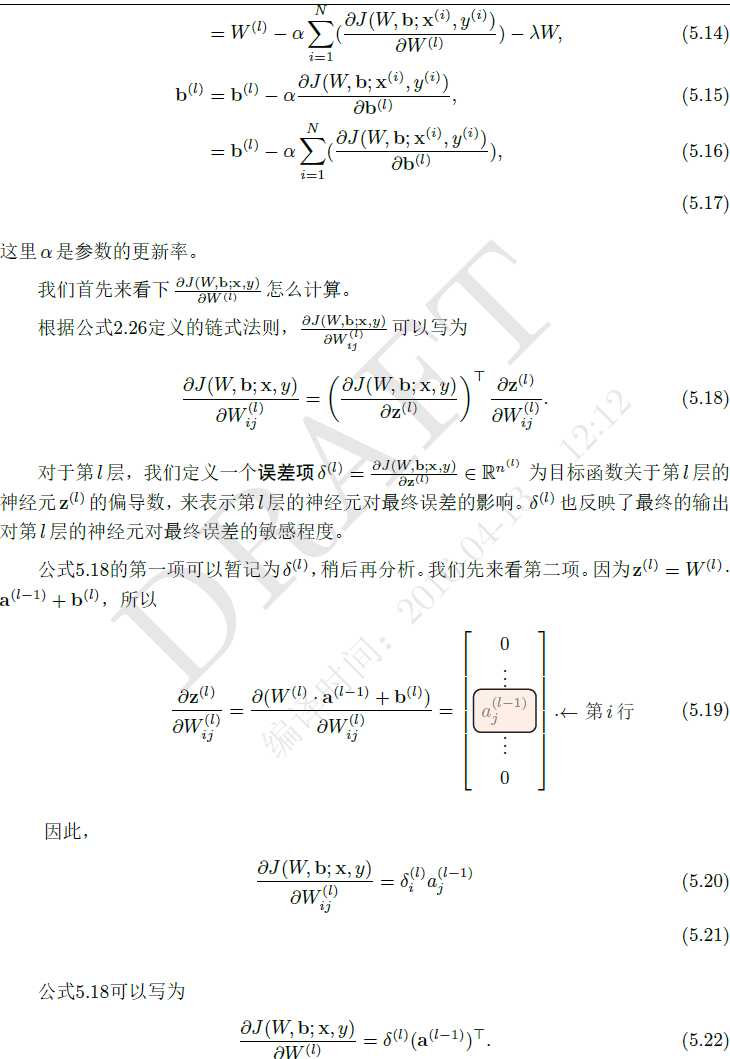
# BP推导及代码实现

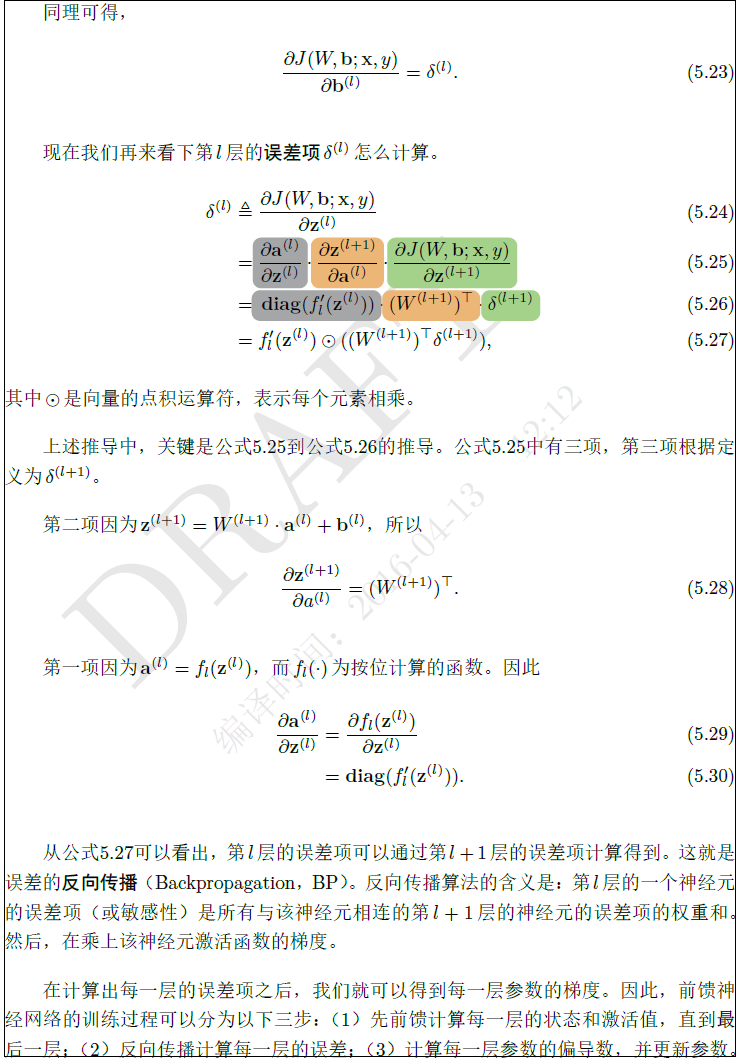
By DanielWang

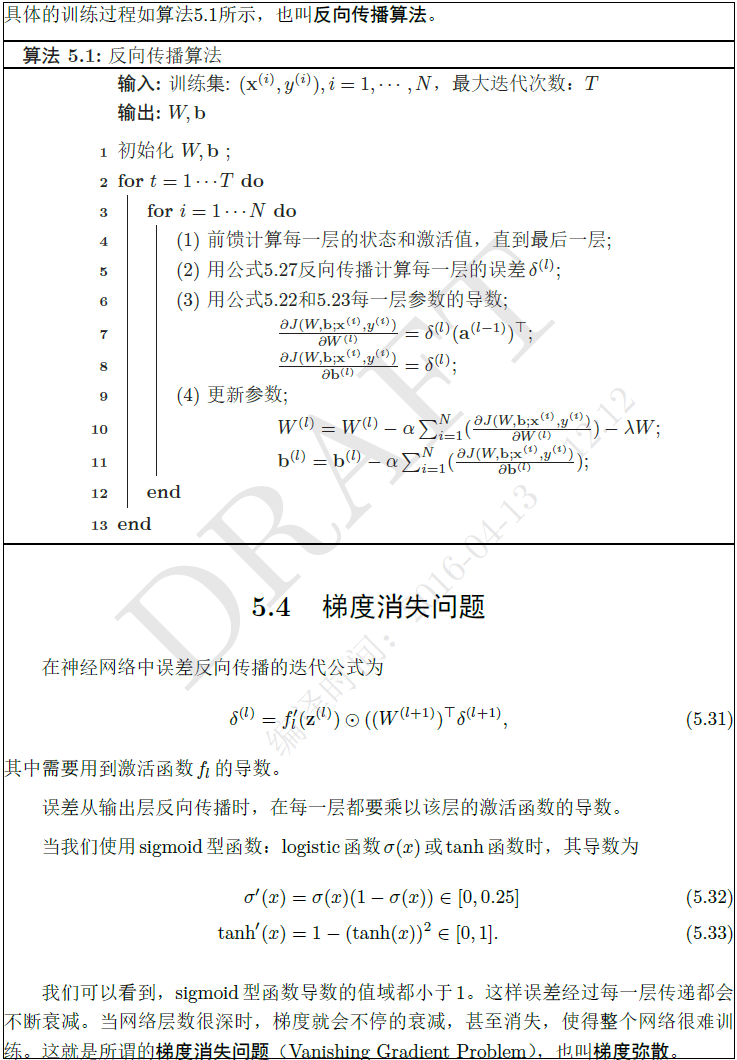
1. BP算法推导











1. 神经网络代码示例
2. 两层神经网络（无隐层）

print '2 Layer Neural Network: input,output'

import numpy as np

# sigmoid function

def nonlin(x, deriv=False):

if (deriv == True):

return x \* (1 - x)

return 1 / (1 + np.exp(-x))

# input dataset

X = np.array([[0, 0, 1],

[0, 1, 1],

[1, 0, 1],

[1, 1, 1]])

# output dataset

y = np.array([[0, 0, 1, 1]]).T

# seed random numbers to make calculation

# deterministic (just a good practice)

np.random.seed(1)

# initialize weights randomly with mean 0

syn0 = 2 \* np.random.random((3, 1)) - 1

for iter in xrange(10000):

# forward propagation

l0 = X

l1 = nonlin(np.dot(l0, syn0))

# how much did we miss?

l1\_error = y - l1

# multiply how much we missed by the

# slope of the sigmoid at the values in l1. import

l1\_delta = l1\_error \* nonlin(l1, True)

# update weights

syn0 += np.dot(l0.T, l1\_delta)

print "Output After Training:"

print l1

只有两层神经网络时，训练数据为（,）,假定神经网络的输出为 = {}, 根据上面的BP算法推导知道==, 网络在（,）的均方差为： = =  (输出层有m个神经元), 对 进行梯度下降求解即可。

X 输入数据，每一行为训练数据的输入实例；

y 训练数据的实际输出，每一行为训练样例值

l0 = X 网络的第一层，即为输入数据

l1 网络的第二层，即输出层

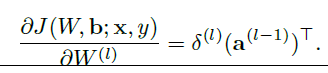
syn0 l0,l1的连接权重矩阵

l1 = nonlin(np.dot(l0, syn0))

l1\_error = y - l1

l1\_delta = l1\_error \* nonlin(l1, True)

syn0 += np.dot(l0.T, l1\_delta)





对于两层网络，因为有=l0，求偏导因此有：

l1\_delta == )f’(l1)

syn0 += np.dot(l0.T, l1\_delta)

通过上面的方法我们知道网络最后一层 l-1层的误差l\_error，l\_delta的求法，如果网络层数较多可以通过BP（backpropagation）逆向求解第l-1层的误差..。

1. 三层神经网络（含有一层隐层）

print '3 Layer Neural Network: input, hidden,output'

import numpy as np

def nonlin(x, deriv=False):

if (deriv == True):

return x \* (1 - x)

return 1 / (1 + np.exp(-x))

X = np.array([[0, 0, 1],

[0, 1, 1],

[1, 0, 1],

[1, 1, 1]])

y = np.array([[0],

[1],

[1],

[0]])

np.random.seed(1)

# randomly initialize our weights with mean 0

syn0 = 2 \* np.random.random((3, 4)) - 1

syn1 = 2 \* np.random.random((4, 1)) - 1

for j in xrange(60000):

# Feed forward through layers 0, 1, and 2

l0 = X

l1 = nonlin(np.dot(l0, syn0))

l2 = nonlin(np.dot(l1, syn1))

# how much did we miss the target value?

l2\_error = y - l2

if (j % 10000) == 0:

print "Error:" + str(np.mean(np.abs(l2\_error)))

# in what direction is the target value?

# were we really sure? if so, don't change too much.

l2\_delta = l2\_error \* nonlin(l2, deriv=True)

# how much did each l1 value contribute to the l2 error (according to the weights)?

l1\_error = l2\_delta.dot(syn1.T)

# in what direction is the target l1?

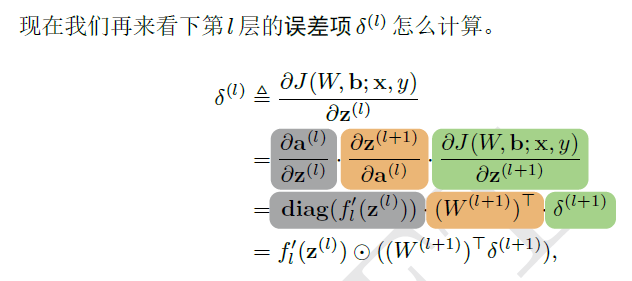
# were we really sure? if so, don't change too much.

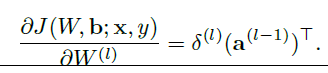
l1\_delta = l1\_error \* nonlin(l1, deriv=True)

syn1 += l1.T.dot(l2\_delta)

syn0 += l0.T.dot(l1\_delta)

这里网络含有隐层，网络最后一层l2的误差等的求法和上面介绍的方法一样，求得l2层相关的参数后，利用BP算法逆向求解即可，参考如下公式：





参考资料：

<http://iamtrask.github.io/2015/07/12/basic-python-network/>

<https://iamtrask.github.io/2015/07/27/python-network-part2/>

 邱锡鹏《神经网络与深度学习讲义》

周志华 《机器学习》