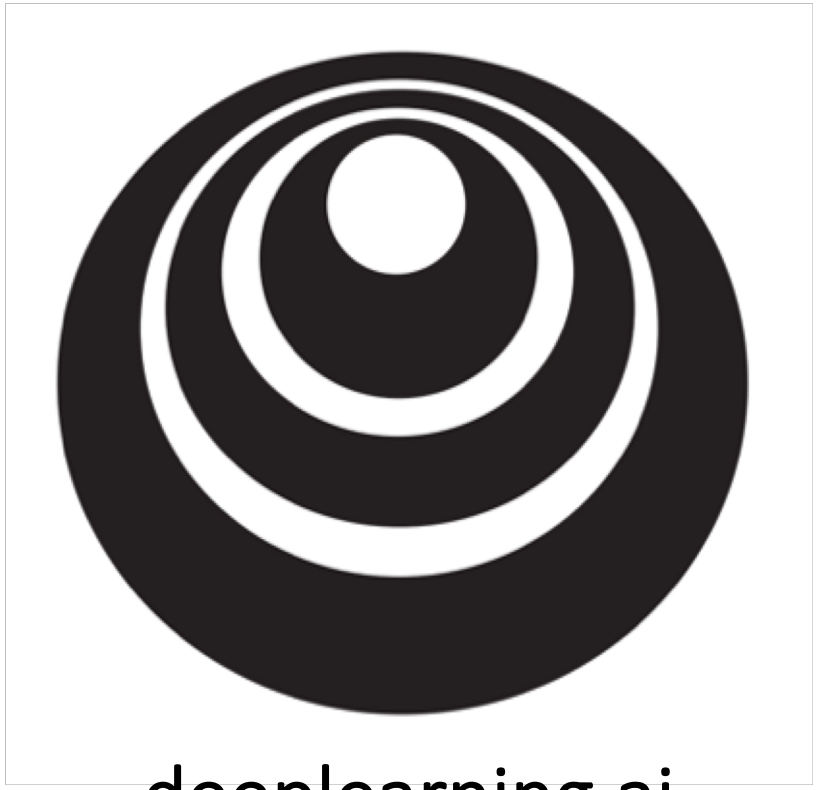


deeplearning.ai

Setting up your optimization problem

Gradient Checking

Gradient check for a neural network

Take $W^{[1]}, b^{[1]}, \ldots, W^{[L]}, b^{[L]}$ and reshape into a big vector θ.

concatenate

$J(W^{[1]}, b^{[1]}, \ldots, W^{[L]}, b^{[L]}) = J(\theta)$

Take $dW^{[1]}, db^{[1]}, \ldots, dW^{[L]}, db^{[L]}$ and reshape into a big vector $d\theta$.

concatenate

Is $d\theta$ the gradient of $J(\theta)$?

Gradient checking (Grad check)

$J(\theta) = J(\theta_1, \theta_2, \theta_3, \ldots)$

For each i:

$$d\theta_{approx}[i] = \frac{J(\theta_1, \theta_2, \ldots, \theta_i + \varepsilon, \ldots) - J(\theta_1, \theta_2, \ldots, \theta_i - \varepsilon, \ldots)}{2\varepsilon}$$

$\approx d\theta[i] = \frac{\partial J}{\partial \theta_i}$

$d\theta_{approx} \overset{?}{\approx} d\theta$

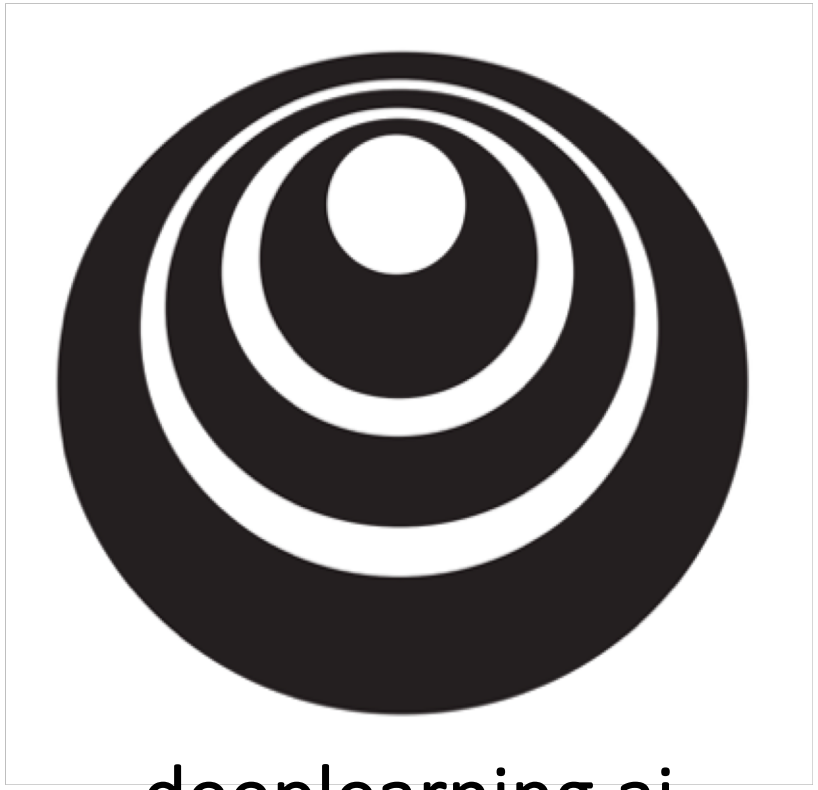
Check $\frac{\|d\theta_{approx} - d\theta\|_2}{\|d\theta_{approx}\|_2 + \|d\theta\|_2}$ $\approx$ 10^{-7} — great! ←

10^{-5}

$\varepsilon = 10^{-7}$

→ 10^{-3} — worry. ←

Andrew Ng



Setting up your optimization problem

Gradient Checking implementation notes

Gradient checking implementation notes

- Don't use in training – only to debug

$d\theta_{approx}[i] \approx d\theta[i]$

- If algorithm fails grad check, look at components to try to identify bug.

$db^{[l]}$ $dW^{[l]}$

- Remember regularization.

$J(\theta) = \frac{1}{m}\sum_i \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_l \|W^{[l]}\|_F^2$

$d\theta$ = gradient of J wrt. θ

- Doesn't work with dropout.

J keep-prob = 1.0

- Run at random initialization; perhaps again after some training.

$W, b \approx 0$

在上一个视频中你已经了解了
有关梯度检验的知识 在这个视频中
我想向你展示一些实用的技巧 以及一些注意事项
关于如何在你的神经网络中运用这些技巧 首先 不要在训练中使用梯度检查
而仅仅在调试时使用 我的意思是
计算d(θapprox i) 计算当i取所有值的情况
这是一个非常慢的计算过程 所以当运用梯度下降时
你会运用反向传播来计算dθ 也就是运用反向传播来
计算导数 只有当你在调试时
你才会需要计算这个 来保证它与dθ足够接近 但当你完成后
你需要关掉梯度检验 别在每一次进行梯度下降迭代的时候
都运行梯度检验 因为这是在是太慢了 第二 如果一个算法没有通过梯度检测
你需要检查它的组成 检查每一个组成成分
尝试找出漏洞 我的意思是 如果d(θapprox)与dθ差距很大的话 我会这么做 检查不同的i值
看看哪些 d(θapprox)的值
与dθ的值差距最大 举个例子 如果你发现
某些θ或者dθ的值 差距非常大 这与某一层或某些层的
db(l)有关 但是dw的组成又很接近 记得我们说过
theta的不同组成与 b和w的不同组成有关 当你发现这点时
那么你也许会发现漏洞 就在你计算db的方法中
即参数b的导数 类似地 反之亦同
如果你发现这两个数相距非常远 d(θapprox)的值与dθ差距非常大 你会发现所有的原因都来自于
某一层中的dw 这也许能帮你找到漏洞的位置 这也许不能总是让你
马上找到漏洞的位置 但是有时候它能帮助你猜测
你需要在哪里找到漏洞 下一个 当你在进行梯度检验时 如果你使用了正则化
别忘了你的正则项 所以你的代价函数为
J(θ)等于(1/m)乘上代价的和 加上正则项 再加上||w(l)||^2关于l求和
这就是J的定义 还有dθ
即J关于θ的倒数 包括这个正则项 所以你需要记住加入这个项 接下来 梯度检验不能与随机失活(dropout)
一起使用 因为在每一次的迭代中 随机失活(dropout)将随机消除
隐藏层单元的不同子集 在使用随机失活(dropout)
进行梯度下降的过程中 并不存在一个容易计算的代价函数J 随机失活(dropout)可以被视为
对于代价函数的优化 但是这个代价函数的定义是
在每一次迭代中 对所有 非常大的可消除节点集进行求和 所以这个代价函数是很难计算的 你只需要对代价函数进行抽样 在那些使用随机失活(dropout)的集合中
每次消除不同的随机集合 所以使用梯度检验来检查 包含了随机失活(dropout)的运算是很困难的 所以我常常在使用梯度检验的同时
不使用随机失活(dropout) 你可以把keep-prob和dropout设为1.0 然后打开dropout
希望我对于dropout的使用是正确的 还有一些别的事情可以做

比如修正那些舍弃节点的模式 并且使用梯度检验来检查它们的模式是否正确 但实际上我通常不这样做 所以我的建议是 关掉随机失活(dropout)
使用梯度检验来检查你的算法 在没有dropout的情况下至少是正确的
然后再打开dropout 最后的这个内容有些微妙 虽然很少发生
但并不是没有可能 你对于梯度下降的使用是正确的
同时w和b在随机初始化的时候 是很接近0的数 但随着梯度下降的进行
w和b有所增大 也许你的反向传播算法
在w和b接近0的时候是正确的 但是当w和b变大的时候
算法精确度有所下降 所以虽然我不经常使用它
但是你可以尝试的一个方法是 在随机初始化的时候
运行梯度检验 然后训练网络一段时间
那么w和b 将会在0附近摇摆一段时间
即很小的随机初始值 在进行几次训练的迭代后
再运行梯度检验 那么这就是梯度检验的全部内容 祝贺你来到了这一周内容的最后部分 在这一周中 你学会了如何
设置训练 开发 和测试集 如何分析高偏差/高方差的情况
以及面对高偏差或者高方差 你该如何应对 你也看到了如何运用
不同形式的正则化 比如L2正则化还有
对你的神经网络进行随机失活(dropout) 即一些加速神经网络训练的技巧 最后是梯度检验的内容 那么 我想你在这一周看了很多内容 你会在这一周的编程训练中
对于这些概念获得很多的训练 那么祝你好运 期待在第二周的视频中见到你