# 神经网络中的一些技巧

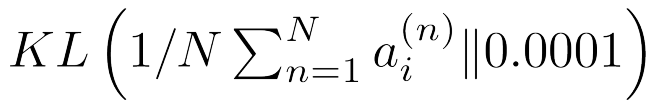
## 1梯度检验

## 

其中，从微分的定义来看，上述公式显然是正确的，但是怎么将其应用到求解误差梯度呢？对于一个给定的数据集，当我们正向扰动参数θ的第i个元素时(可以简单理解成θ加上一个极小的正数)，咱们基于前向传导可以计算出误差项J(θ(i+))。同理，当我们负向扰动参数θ的第i个元素时，咱们基于前向传导可以计算出新的误差项J(θ(i－))。因此，其实通过做两次前向运算，我们就可以根据上面的公式估计出任何给定参数的梯度。当然了，其实只做一次前向传导所需要的运算量也不小了，所以在估计梯度时，这种方法比较耗时，但是，在用于验证反向传播的实现时，这种方法很赞，也用得很多。

## 2预防过拟合

1. 减少模型深度
2. 在权重上使用L1或者L2正则化
3. 尽早停止：使用使得验证误差最好的参数
4. Sparsity constraints on hidden activations, e.g., add to cost：

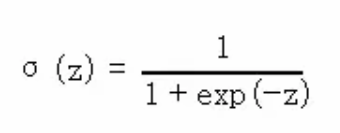


在上述公式中，是矩阵的F范数（frobenius norm），λ是用于在加权和目标函数中进行正则化的相对权重。加上这个正则化项，意在通过作用到损失的平方来惩罚那些在数值上特别大的权重，也就是让权重的分配更均匀一些。这样一来，目标函数（也就是分类器）的随意度(也就是可用于拟合的复杂度)就被降低了，约束了拟合函数的假设空间，因此减少了发生过拟合的可能性。施加这样一种约束条件可以用先验贝叶斯思想来理解，即最优的权重分配是所有权重都接近0。你想知道有多接近？对啦，这正是λ所控制的——大的λ会倾向于使所有权重都趋于0。值得注意的是，偏移量b不会被正则化，也不会被计算入上述的损失项。

## 3 神经单元

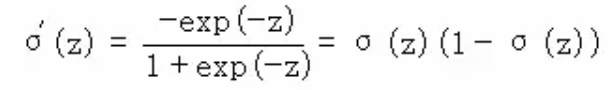
常用的有sigmoid神经元（sigmoidal neurons）来实现非线性分类的神经网络算法，然而在许多应用中，使用其他激励(激活)函数（activation functions）可以设计出更好的神经网络。这里列举了一些常用选择的函数表达式和梯度定义，它们是可以和sigmoid函数（sigmoidal functions）互相替代的。

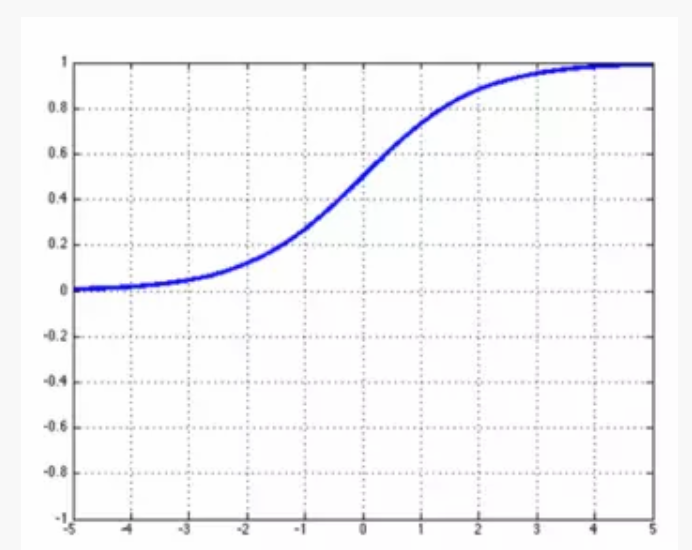
Sigmoid：



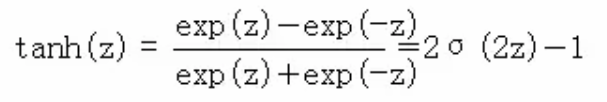
其中， σ(z)∈(0,1)

σ(z)的梯度为：

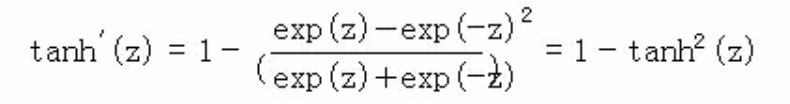


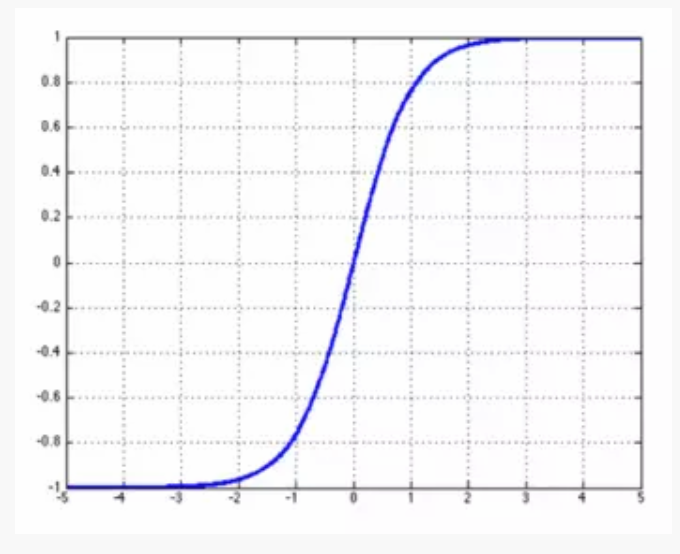


Tanh：tanh函数是除了sigmoid函数之外的另一种选择，在实际中，它的收敛速度更快。tanh函数与sigmoid函数最主要的不同是tanh函数的输出结果在-1和1之间，而sigmoid函数的输出结果在0和1之间。

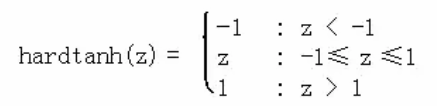


其中， tanh(z)∈(－1,1)   
tanh(z)的梯度为：

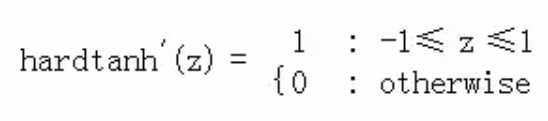


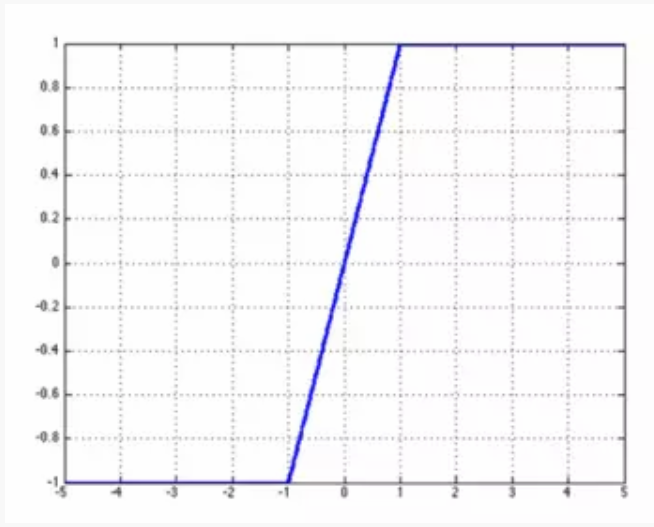


**Hard Tanh**：hard tanh(硬双曲余弦正切)函数在有些时候要优于tanh函数，因为它在计算上更为简便。然而当z大于1时，hard tanh函数会在数值上形成饱和（译者注：即恒等于1）。hard tanh的激活函数为：

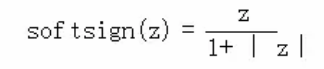


其微分也可以用分段函数来表达：



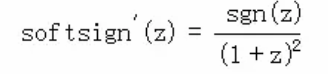


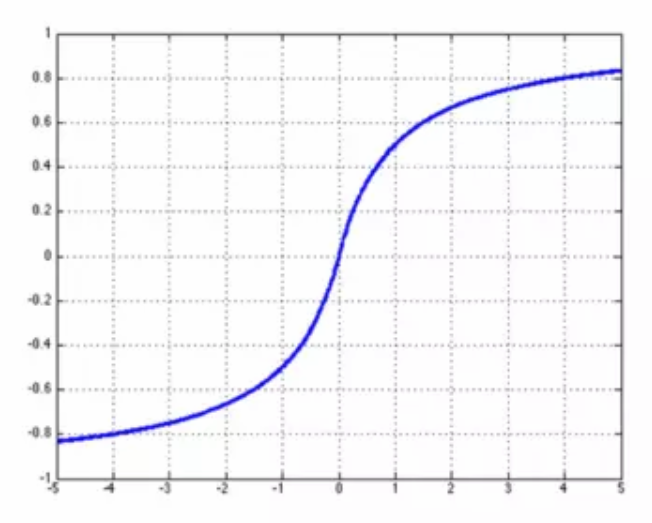
**Soft Sign**：Soft Sign函数是另一个可以被用来替代Tanh函数的非线性函数，因为它也不会像硬限幅函数（hard clipped functions）那样过早饱和。其函数表达式为：



其微分表达式为：

其中sgn()是符号函数，即根据z的符号返回＋1或－1。

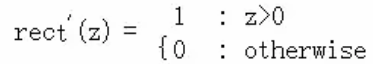


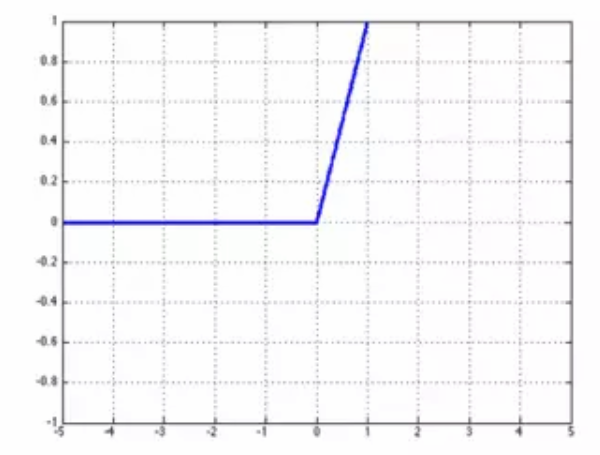


**ReLU**：ReLU（修正线性单元，Rectiﬁed Linear Unit）函数是激活函数的一个流行选择，因为即使对特别大的z，它也不会饱和，并且已经发现它在计算机视觉应用中非常好用。其函数表达式为：

rect(z)=max(z,0)

其微分表达式为：

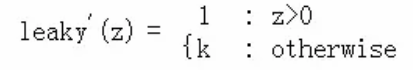


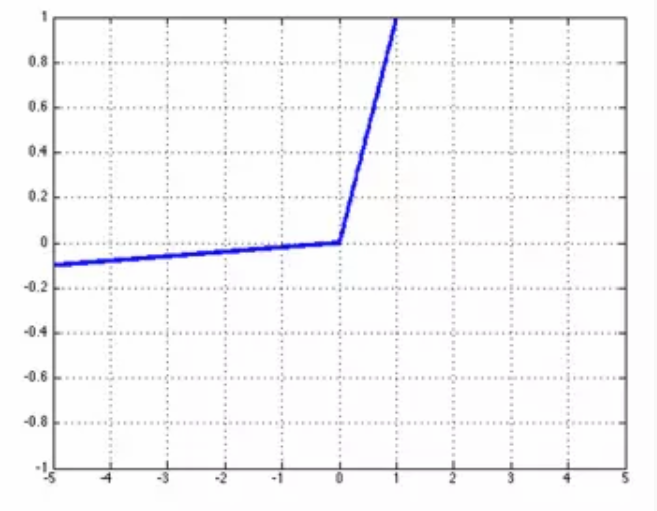


**Leaky ReLU**：对于非正数的z，传统设计上的ReLU单元不会回传误差——而leaky ReLU修正了这一点，使得z是负数时，很小的误差也会反向传播回传回去。其函数表达式为：

leaky(z)=max(z,k×z)

其中，0<k<1   
因此其微分表达式可以被表示为：





## 4 Xavier参数初始化

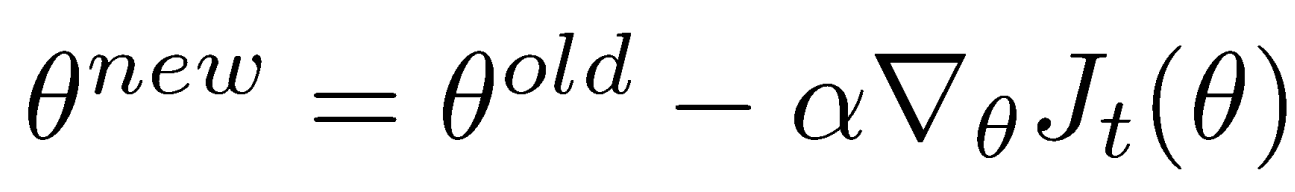
在《理解训练深层前馈神经网络的困难（Understanding the Difficulty of Training Deep Feedforward Neural Networks）》(2010)一文中，Xavier等人研究了不同权重和偏差的初始化方案对训练动力（training dynamics）的影响。实证研究结果表明，对于sigmoid和tanh激活单元，当矩阵的权重以均匀分布在以下值域范围内被随机初始化时，有着更低的错误率和更快的收敛速度：

其中，是W关联的输入单元的数量（fan-in），是W关联的输出单元的数量（fan-out）。

在这种参数初始化方案里，偏差项(b)被初始化为0。这种方法的目的是维持跨层的激活方差和反向传播梯度方差。如果不初始化，梯度方差（包含大量修正信息）一般会随层间反向传播而很快衰减。

## 5学习速率

模型最优化的过程中，参数更新的速度可以通过学习速率来控制。比如下面的梯度下降公式中，α是学习速率：



学习速率过大甚至可能会导致损失函数的不收敛，因为有时候因为太激进，参数的迭代步伐太大，一不小心跨过了凸优化的极小值

那怎么办呢？一个简单的方案就是，初始化一个比较小的学习速率，谨慎地在参数空间内迭代和调整以避免模型不收敛。同时，我们还可以固定模型中所有参数的学习速率，而不是为模型中所有参数设定不同的学习速率。

深度学习系统训练阶段通常最耗时耗资源，一些研究也试图应用一些新的方法来设置学习速率。

例如，Ronan Collobert通过取神经元输入单元数的平方根的倒数来把权重（的学习速率进行标准化。

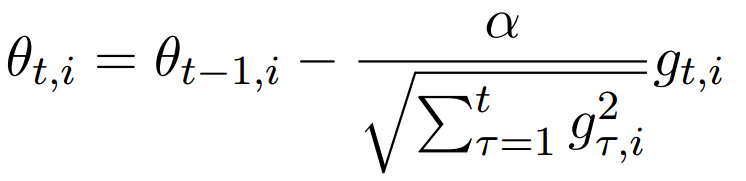
另一种方法是允许学习速率随着时间而减小，如：



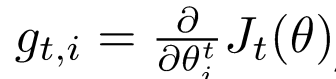
在上述方案中，是一个可调参数，代表起始学习速率。τ也是一个可调参数，代表学习速率应该开始降低的时间。实践中，这种方法相当有效

## 6 使用AdaGrad进行次梯度优化

AdaGrad是标准随机梯度下降法（SGD）的一种实现，但是有一个关键的区别：每个参数的学习速率是不同的。参数的学习速率取决于该参数梯度更新的历史情况，更新的历史越稀疏，就应该使用更大的学习速率加快更新。换句话说，那些在过去未被更新的参数更有可能在现在获得更高的学习速率。其形式如下：



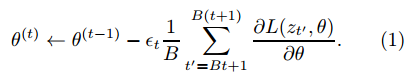
其中，



对应上述公式我们可以看到，在这种算法中，如果梯度历史的方均根（RMS）非常低，学习速率会比较高。算法的实现如下：

## 梯度下降法

使用mini-batch来进行梯度更新



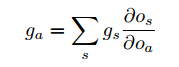
B是每一个mini-batch的大小，当B=1时是通常所说的在线梯度下降法，B=数据集大小时，是标准的梯度下降法，当B增加时，我们可以选择并行方式去计算或者采用矩阵-矩阵的乘法运算而不是矩阵-向量的乘法运算，通常这样能节省俩倍的时间。另一方面当B增加时，相同时间内能得到更新次数减少，收敛速度变慢。将这俩中相反的结果结合在一起，可以得到一个典型的U型曲线，选择U的平衡点作为B的值。当B变小时，更新速度变快，在相同时间内可以获得更多信息，进行更多次搜索，这也许可以解释为什么小的B有时反而能得到更好的观察结果。

在一些大的训练集上，标准的梯度下降法会使得训练过慢，而随机梯度训练法与训练集大小无关，在一些大的数据集也能快速收敛，因此，为了能在合理时间内收敛，使用随机梯度下降法是十分重要的。

另外，还需要注意的一点就是，在使用随机梯度下降法（包括mini-batch方法）时，每次选取的样例应该是随机的，这样会使得训练速度变快。

## 梯度计算和自动微分

通常来说，一个神经网络的节点可以分成三种，输入节点，输出节点，内部节点。通常，每个节点会包含一个来自前一层节点的加权和输入（除输入节点）和一个经过激活函数运算的输出，在这个基础上，我们为每一个节点多添加一个属性，g，表示损失函数对该节点的输出o的偏导数值（梯度在这一方向的值）。g的计算与损失函数的计算方向返，从输出节点开始，逐层向前计算，例如，对于节点a，它的输出为为，则对的偏导值可以通过下式计算，其中s是节点a的后一层节点。



在具体编程时，要使用模块化编程方法，可以将前向传播时的激活函函数、反向误差传播时的梯度计算函数封装起来

## 防止特征互相适应

1.训练时间：在每一个评估的实例中（在线随机梯度下降法）,随机设置每个神经元的输入的的50%为0

2.测试时间：将模型重量减半（现在是原来的两倍）

3.这个可以用来防止特征互相适应：一个特征不能只在特定的其他特性的情况下有用

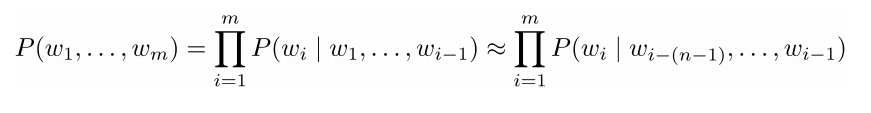
4.在朴素贝叶斯（所有特征权都是独立的）和逻辑斯谛回归模型（在所有其他因素的背景下设置权重）之间的一种中间地带

5.可以认为是一种装袋模型

6.也可以认为是正则化矩阵

## 语言模型

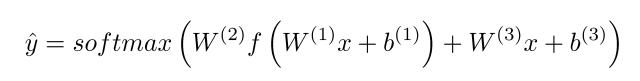
可以使用N个前面出现的单词来描述该单词出现的概率，表示成序列就是：

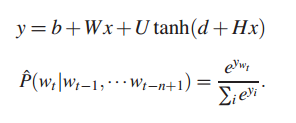


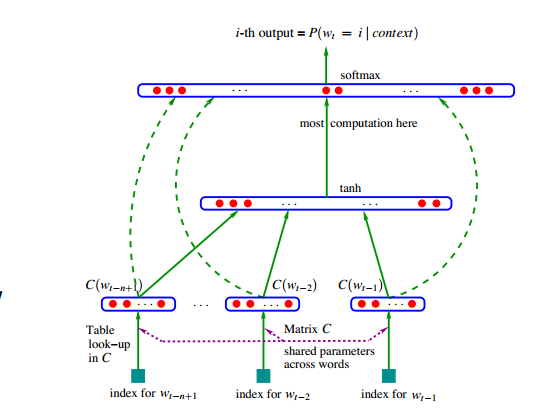
对于大量任务来说非常有用的:可以被用来决定一个序列是否是一个好的/符合文法的翻译或是语音

## 最初的神经语言模型

一个神经概率语言模型

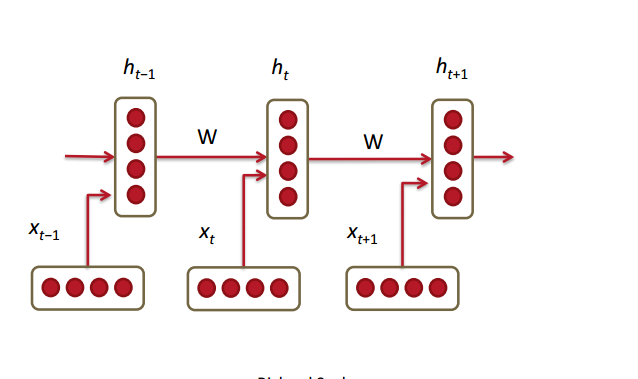




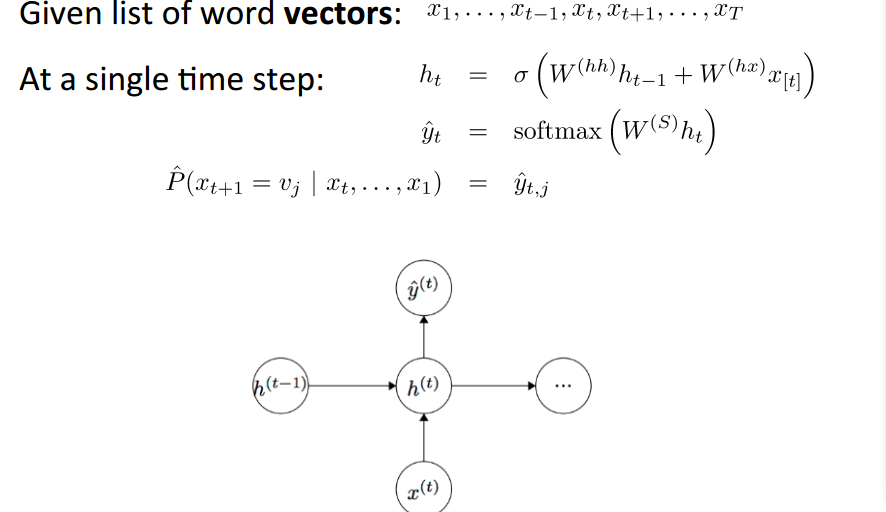


## RNN模型

将神经网络的所有前面出现过的单词和每一次的权重绑定在一起



## RNN神经网络语言模型



主要思想：W权值矩阵在任何一步都一样，不会改变。

## Unsupervised pre--‐training(无监督预处理)

网络越深，会有以下问题：

* 样本多，特征多，参数多。
* 收敛至局部最优
* 梯度扩散

解决办法：

无监督预处理：即训练网络的第一个隐藏层，再训练第二个…最后用这些训练好的网络参数值作为整体网络参数的初始值。

## Stochastic gradient descent and seting learning rates（随机梯度算法和设置学习率）

## 比较梯度下降，随机梯度下降，批量梯度下降

### 一.解决目标及情景假设:

当给定一些数据，输入[http://latex.codecogs.com/gif.latex?x](http://www.codecogs.com/eqnedit.php?latex=x)向量已知，输出[http://latex.codecogs.com/gif.latex?y](http://www.codecogs.com/eqnedit.php?latex=y)也已知，设计一个线性函数[http://latex.codecogs.com/gif.latex?y=h(x)](http://www.codecogs.com/eqnedit.php?latex=y=h(x))去拟合这些数据。  
既然是线性函数，在此不妨设为[http://latex.codecogs.com/gif.latex?h(x)=w_%7b0%7d*x_%7b0%7d+w_%7b1%7d*x_%7b1%7d](http://www.codecogs.com/eqnedit.php?latex=h(x)=w_%7b0%7d*x_%7b0%7d+w_%7b1%7d*x_%7b1%7d)。  
此时我们遇到的问题就是如何确定[http://latex.codecogs.com/gif.latex?w_%7b0%7d](http://www.codecogs.com/eqnedit.php?latex=w_%7b0%7d)和[http://latex.codecogs.com/gif.latex?w_%7b1%7d](http://www.codecogs.com/eqnedit.php?latex=w_%7b1%7d)这两个参数，即[http://latex.codecogs.com/gif.latex?w&space;=(w_%7b0%7d,w_%7b1%7d)](http://www.codecogs.com/eqnedit.php?latex=w&space;=(w_%7b0%7d,w_%7b1%7d))这个向量。  
既然是拟合，则拟合效果可以用误差函数：[http://latex.codecogs.com/gif.latex?E(w)=\sum&space;%5bh(x)-y%5d%5e2/2](http://www.codecogs.com/eqnedit.php?latex=E(w)=\sum&space;%5bh(x)-y%5d%5e2/2)来衡量。  
其中[http://latex.codecogs.com/gif.latex?w](http://www.codecogs.com/eqnedit.php?latex=w)是权重二维向量，[http://latex.codecogs.com/gif.latex?x](http://www.codecogs.com/eqnedit.php?latex=x)是输入二维向量，[http://latex.codecogs.com/gif.latex?x](http://www.codecogs.com/eqnedit.php?latex=x)和[http://latex.codecogs.com/gif.latex?y](http://www.codecogs.com/eqnedit.php?latex=y)都是训练集的数据，即已知。  
至于后面除于2只是为了之后的推导过程中对[http://latex.codecogs.com/gif.latex?E](http://www.codecogs.com/eqnedit.php?latex=E)求导时候可以消除系数，暂时可以不管。  
因为我们解决的目标是找出一个向量[http://latex.codecogs.com/gif.latex?w&space;=(w_%7b0%7d,w_%7b1%7d)](http://www.codecogs.com/eqnedit.php?latex=w&space;=(w_%7b0%7d,w_%7b1%7d))使得[http://latex.codecogs.com/gif.latex?E(x)](http://www.codecogs.com/eqnedit.php?latex=E(x))值最小，即误差最小。  
其实这个问题本质上也是搜索最优解的问题，如果用暴力搜索的话，随机取每个可能的值去让机器每天每夜地跑，显然这是不可能的。

所以此时有一种搜索策略：梯度下降。

### 二. 梯度下降方法：

梯度其实就是高数求导方法，对[http://latex.codecogs.com/gif.latex?E](http://www.codecogs.com/eqnedit.php?latex=E)这个公式针对每个维数[http://latex.codecogs.com/gif.latex?(w_%7b0%7d,w_%7b1%7d)](http://www.codecogs.com/eqnedit.php?latex=(w_%7b0%7d,w_%7b1%7d))求偏导后的向量[http://latex.codecogs.com/gif.latex?\bigtriangledown&space;E(w)=(\partial&space;E/\partial&space;w_%7b0%7d,\partial&space;E/\partial&space;w_%7b1%7d)](http://www.codecogs.com/eqnedit.php?latex=\bigtriangledown&space;E(w)=(\partial&space;E/\partial&space;w_%7b0%7d,\partial&space;E/\partial&space;w_%7b1%7d))  
梯度为最陡峭上升的方向，对应的梯度下降的训练法则为：  
[http://latex.codecogs.com/gif.latex?w=w-\eta&space;\bigtriangledown&space;E(w)](http://www.codecogs.com/eqnedit.php?latex=w=w-\eta&space;\bigtriangledown&space;E(w))  
这里[http://latex.codecogs.com/gif.latex?\eta](http://www.codecogs.com/eqnedit.php?latex=\eta)的代表学习速率，决定梯度下降搜索中的步长 。  
上式的[http://latex.codecogs.com/gif.latex?w](http://www.codecogs.com/eqnedit.php?latex=w)是向量，即可用将该式写成分量形式为：[http://latex.codecogs.com/gif.latex?w_%7bi%7d=w_%7bi%7d-\eta&space;*\partial&space;E/\partial&space;w_%7bi%7d](http://www.codecogs.com/eqnedit.php?latex=w_%7bi%7d=w_%7bi%7d-\eta&space;*\partial&space;E/\partial&space;w_%7bi%7d)  
现在关键就使计算∂E/∂wi：  
(推导过程很简单，书上写的很详细，这里只记录结论:)  
∂E/∂wi=∑（h(x)-y）\*(xi)  
这里的∑是对样本空间，即训练集进行一次遍历，耗费时间较大，可以使用梯度下降的随机近似：

随机梯度下降策略来改进时间。

### 三.随机梯度下降的随机近似：

既然是随机近似，则顾名思义，肯定是用近似方法来改善梯度下降时候的时间复杂度问题。  
正如上所说，在∂E/∂wi=∑（h(x)-y）\*(xi) 的时候∑耗费了大量的时间，特别是在训练集庞大的时候。  
所以肯定有人会猜想，如果把求和去掉如何，即变为∂E/∂wi=（h(x)-y）\*(xi)。  
幸运的是，猜想成立了。  
只是要注意一下标准的梯度下降和随机梯度下降的区别：  
1.标准下降时在权值更新前汇总所有样例得到的标准梯度，随机下降则是通过考察每次训练实例来更新。  
2.对于步长 η的取值，标准梯度下降的η比随机梯度下降的大。  
因为标准梯度下降的是使用准确的梯度，理直气壮地走，随机梯度下降使用的是近似的梯度，就得小心翼翼地走，怕一不小心误入歧途南辕北辙了。

3.当E(w)有多个局部极小值时，随机梯度反而更可能避免进入局部极小值中。

### 四.对比梯度下降和随机梯度下降和批量梯度下降

梯度下降：在梯度下降中，对于[http://latex.codecogs.com/gif.latex?w](http://www.codecogs.com/eqnedit.php?latex=w)的更新，所有的样本都有贡献，也就是参与调整[http://latex.codecogs.com/gif.latex?w](http://www.codecogs.com/eqnedit.php?latex=w)。其计算得到的是一个标准梯度。因而理论上来说一次更新的幅度是比较大的。如果样本不多的情况下，当然是这样收敛的速度会更快啦。

随机梯度下降：可以看到多了随机两个字，随机也就是说我用样本中的一个例子来近似我所有的样本，来调整[http://latex.codecogs.com/gif.latex?w](http://www.codecogs.com/eqnedit.php?latex=w)。

批量梯度下降：其实批量的梯度下降就是一种折中的方法，他用了一些小样本来近似全部的，其本质就是我1个指不定不太准，那我用个30个50个样本那比随机的要准不少了吧，而且批量的话还是非常可以反映样本的一个分布情况的。

### 五.BGD、SGD以及MBGD

参考：http://www.cnblogs.com/maybe2030/p/5089753.html

博客:“梯度下降法的三种形式BGD、SGD以及MBGD”将梯度下降分为：

批量梯度下降法BGD：批量梯度下降法（Batch Gradient Descent，简称BGD）是梯度下降法最原始的形式，它的具体思路是在更新每一参数时都使用所有的样本来进行更新。（对应第四节中的梯度下降）

随机梯度下降法SGD：

小批量梯度下降法MBGD:有上述的两种梯度下降法，其各自均有优缺点，那么能不能在两种方法的性能之间取得一个折衷呢？即，算法的训练过程比较快，而且也要保证最终参数训练的准确率，而这正是小批量梯度下降法（Mini-batch Gradient Descent，简称MBGD）的初衷。（对应第四节中的批量梯度下降）