**一. 记录**

Deconvolution层的kernelsize，stride，padding，计算方法如下：

layer {

name: "upsample", type: "Deconvolution"

bottom: "{{bottom\_name}}" top: "{{top\_name}}"

convolution\_param {

kernel\_size: {{2 \* factor - factor % 2}} stride: {{factor}}

num\_output: {{C}} group: {{C}}

pad: {{ceil((factor - 1) / 2.)}}

weight\_filler: { type: "bilinear" } bias\_term: false

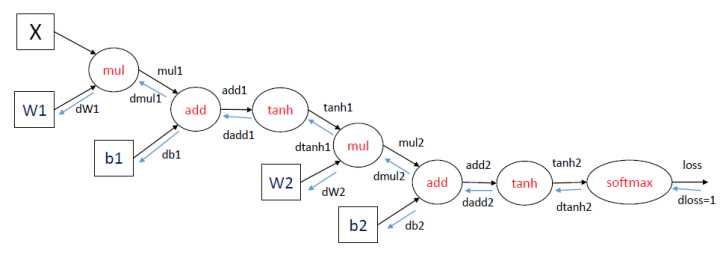
}

param { lr\_mult: 0 decay\_mult: 0 }

}

其中factor是指反卷积之前的所有卷积pooling层的累积采样步长，卷积层使feature map变小，是因为stride，卷积操作造成的影响一般通过padding来消除，因此，累积采样步长factor就等于反卷积之前所有层的stride的乘积。

**二、BP Softmax computation graph**

这里举了一个三层神经网络（一个输入层、一个隐层和一个输出层）的例子，使用了softmax输出层，损失函数使用交叉熵。训练神经网络可以使用梯度下降的方法，重点是计算梯度，也就是损失函数对参数的导数，在图中可以表示为dloss/dW1，dloss/dW2，dloss/db1和dloss/db2。如何计算这些梯度，使用的就是BP算法，其实也就是求导的链式法则。

在每一轮迭代中，首先进行forward propagation，也就是计算computation graph中每个节点的状态：

mul1 = X \* W1

add1 = mul1 + b1

tanh1 = tanh(add1)

mul2 = tanh1 \* W2

add2 = mul2 + b2

tanh2 = tanh(add2)

loss = softmax\_loss(tanh2)

之后进行back propagation，也就是计算computation graph中每个节点相对于损失函数（这里表示为loss）的导数，这里面应用了链式法则。对于dloss/dtanh2, dloss/dadd2等导数，下面省略分子直接表示为dtanh2等。

dloss = 1

dtanh2 = softmax\_loss\_diff(tanh2) \* dloss

dadd2 = tanh\_diff(add2) \* dtanh2

db2 = 1 \* dadd2

dmul2 = 1 \* dadd2

dW2 = tanh1 \* dmul2

dtanh1 = W2 \* dmul2

dadd1 = tanh\_diff(add1) \* dtanh1

db1 = 1 \* dadd1

dmul1 = 1 \* dadd1

dW1 = X \* dmul1

上面的变量都可以用矩阵表示，直接进行矩阵运算。其中dW1，dW2，db1和db2就是我们需要求的参数的梯度。

在编程实现上，每一个计算节点都可以定义两个函数，一个是forward，用于给定输入计算输出；一个是backward，用于给定反向梯度，计算整个表达式（相当于损失函数）相对于这个节点的输入的梯度，应用链式法则就是：这个节点相对于其输入的梯度（直接对输入求导）乘以这个节点接受的反向梯度。

我有一个tutorial，使用Python如何一步一步的实现神经网络，而且可以自定义网络的层数和每层的维度，扩展性很强。其中，抽象出来了gate（AddGate，MulGate），layer（Tanh，Sigmoid）和output（Softmax），你也可以自己实现不同的layer比如ReLu，或不同的output（比如Hinge）。

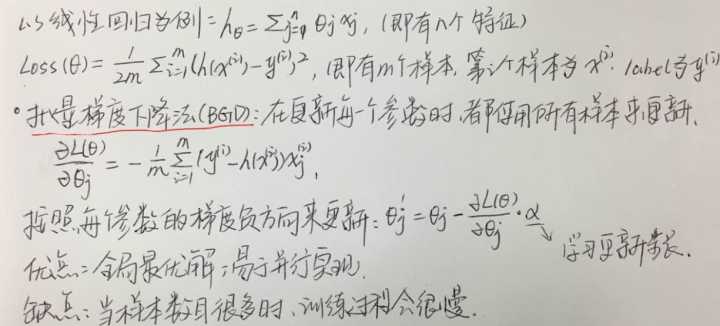
感兴趣的请移步  
[**GitHub - pangolulu/neural-network-from-scratch: Implementing Multiple Layer Neural Network from Scratch**](https://link.zhihu.com/?target=https%3A//github.com/pangolulu/neural-network-from-scratch)

**三、梯度优化算法**

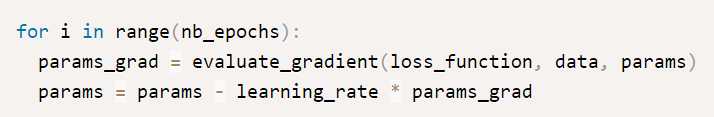
《An overview of gradient descent optimization algorithms》

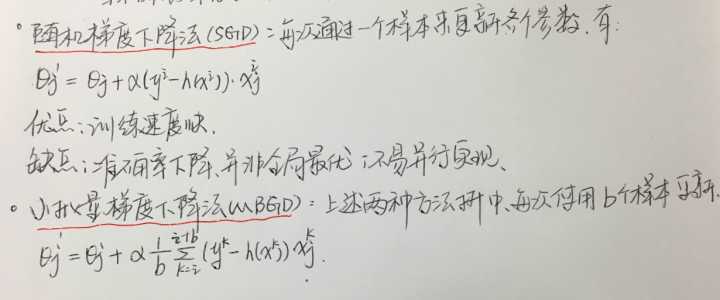
gradient descent基本形式：BGD,SGD,MBGD。

启发式优化算法：Momentum、NAG、Adagrad、Adadelta、RMSprop、Adam、AdaMax、Nadam

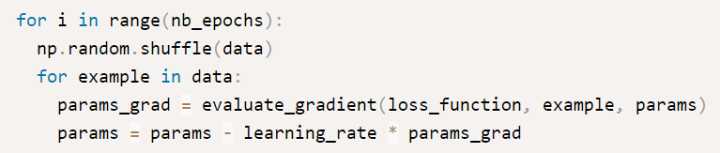


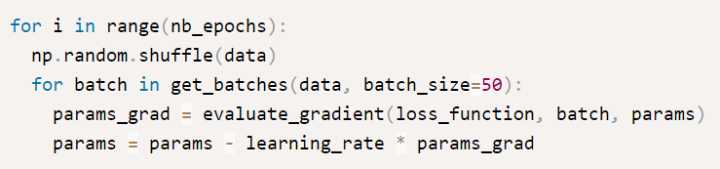
BGD code



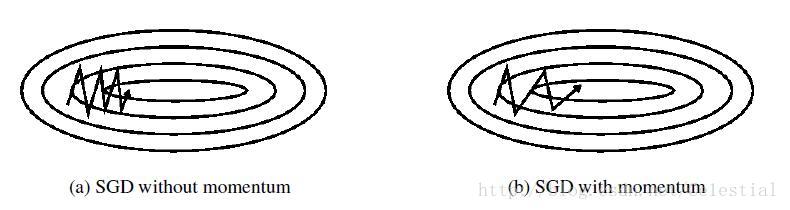


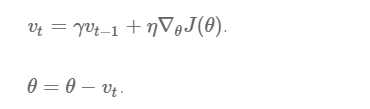
SGD&MBGD code

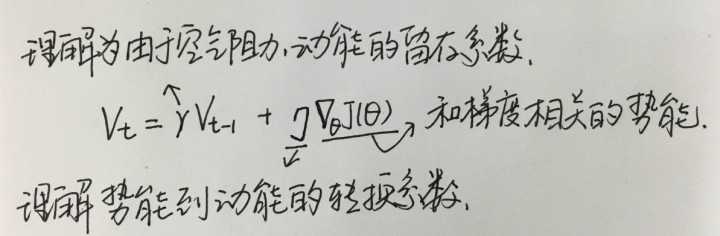




Momentum







即物体的动能 = 上一个时刻的动能 + 上一时刻的势能差。由于有阻力和转换时的损失，所以两者都乘以一个系数。

就像一个小球从坡上向下滚，当前的速度取决于上一个时刻的速度和势能的改变量。

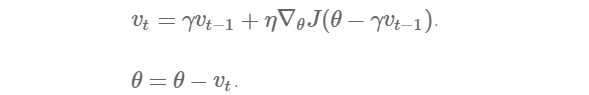
这样在更新参数时，除了考虑到梯度以外，还考虑了上一个时刻参数的历史变更幅度。例如，参数上一次更新幅度较大，并且梯度还不小，那么再更新时是不是得更猛烈些了

**Nesterov accelerated gradient（NAG）**

还是以上面小球的例子来看，momentum方式下小球完全是一种盲目被动的方式滚下的。这样有个缺点就是在临近最优点附近是控制不住速度。我们希望小球很smart，它可以预判后面的"地形"，要是后面地形还是很陡，那就继续坚定不移的大胆走下去；不然的话咱就收敛点~ 当然，小球自己也不知道真正要走到哪里，这里以

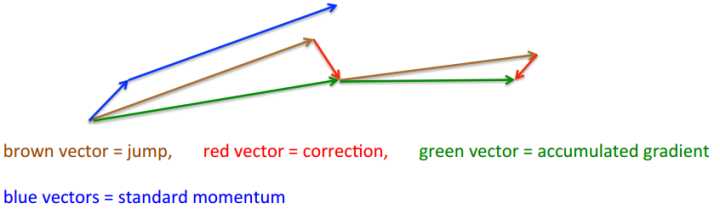
https://pic4.zhimg.com/50/bad8d8b35b1d7094258c545f1adeb2ef_hd.jpg

作为近似下一位置，就有：



相比于之前momentum方式是考虑上一时刻的动能和当前点的梯度，而NAG考虑的是上一时刻的动能和近似下一点的梯度。这使得它可以先看看自己的"前途"，然后慎重一些。

Hinton的slides是这样给出的：



两个blue vectors分别理解为梯度和动能，两个向量和即为momentum方式的作用结果。

而靠左边的brown vector是动能，可以看出它那条blue vector是平行的，但是它预测了下一阶段的梯度是red vector，因此向量和就是green vector，即NAG方式的作用结果。

**Adagrad**

"It adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters."因此，Adagrad对于稀疏数据上有着良好的表现。

之前所讲的方法中所有参数在更新时均使用同一个learning rate，而在Adagrad的每一个参数的每一次更新中都使用不同的learning rate。这样的话，令第t步更新时对第i个参数的梯度为：

https://pic2.zhimg.com/50/200a964f1808444f90721dad68aa31b2_hd.jpg

参数的更新的一般形式为：

https://pic3.zhimg.com/50/6ffa14fefc59dca333ac24c57701efbf_hd.jpg

这里的Gt是一个d\*d维对角矩阵，对角线上的元素（i,i）表示在第t步更新时，历史上θi梯度平方和的积累。ϵ只是一个值很小的防止分母为0的平滑项。

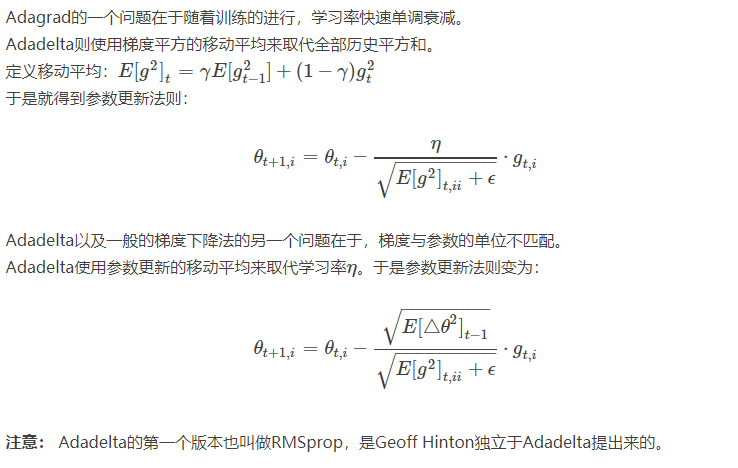
Q：若是这样的话，为什么在step t用对角矩阵保存中间信息呢？使用一维数组不就可以了么，是为了某些方便计算么？

A: 个人感觉是为了矩阵化，如果用对角矩阵，整个更新可以直接用一次矢量运算完成，节约循环时间。

Q：为何带上平方根符号？据说这样的效果更好。

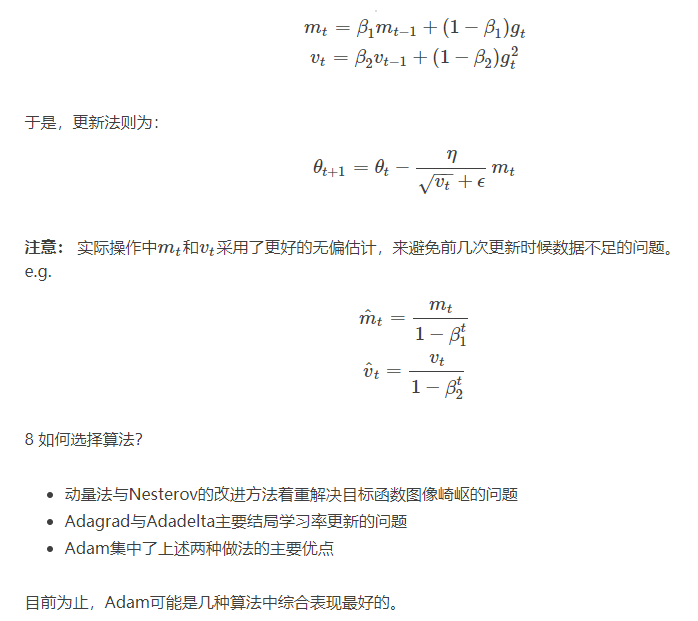
A: 个人感觉，巨大多数自适应的learning rate都是有根据变化幅度而更改，并且具有逐渐收敛的特性。所以这种启发式方法仅是一种可行性方式，并不是唯一的，也绝不是最好的。把握其本质即可。

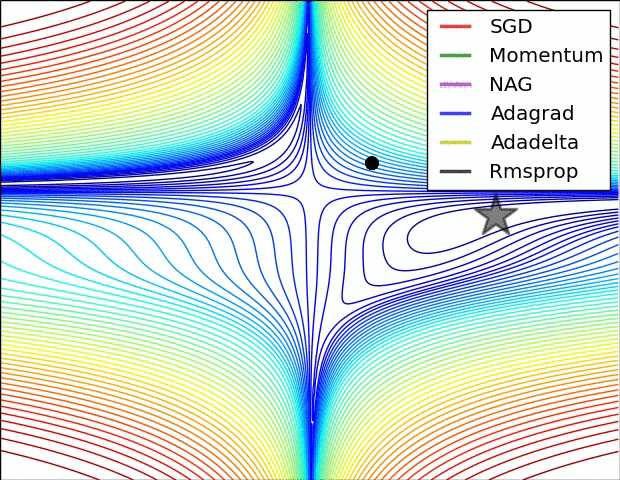
**Adadelta**

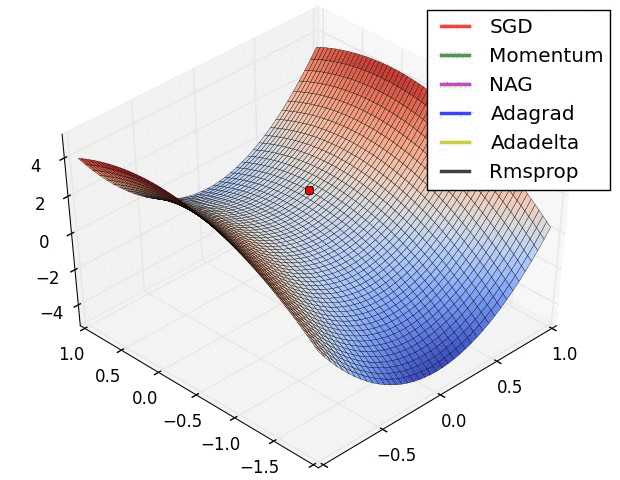
****

**Adam（结合了动量法和Adadelta算法）**

如果把Adadelta里面梯度的平方和看成是梯度的二阶矩，那么梯度自身的求和就是一阶矩。Adam算法在Adadelta的二阶矩基础上又引入了一阶矩。   
而一阶矩，其实就类似于动量法里面的动量。







**一些tricks**

1. 为避免bias，做shuffle。但有时在某种背景下，也可刻意地事先order training examples。
2. Batch normalization。
3. Early stopping。若指标趋近平稳，及时终止。
4. Gradient noise。引入一个符合高斯分布的noise项，使得在poor initialization时具有更好的鲁棒性。形式如下：

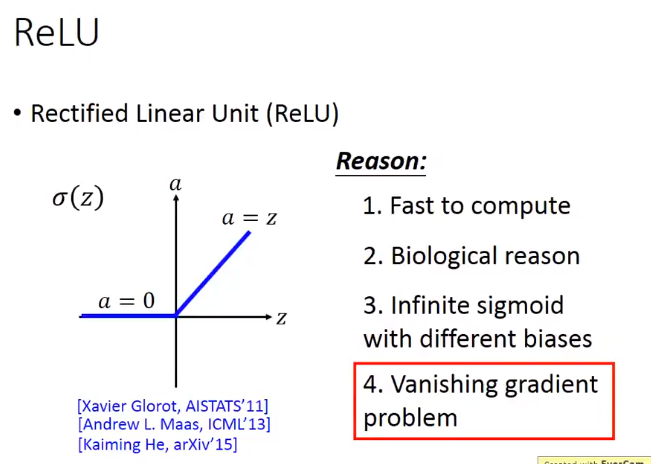
https://pic4.zhimg.com/50/8af790801973da67682a761a663a5846_hd.jpg

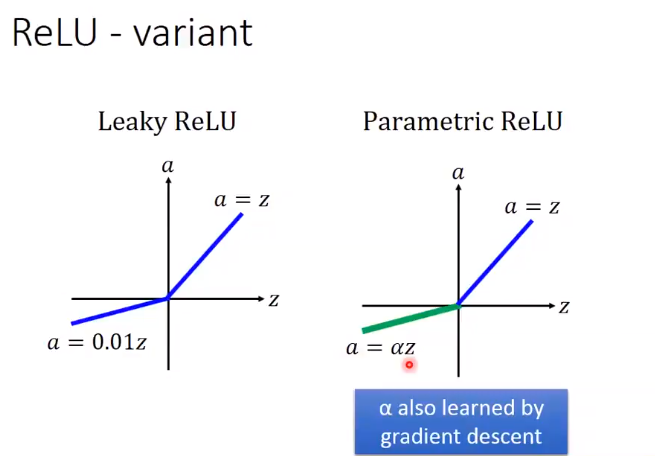
同时对方差退火，因为随着训练进行越来越稳定，noise也随之减弱，退火项如下：

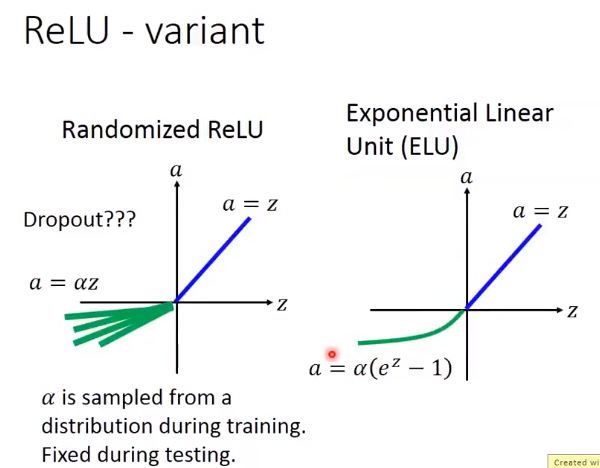
https://pic4.zhimg.com/50/96b3903d4928edcf8ee44f4150c9bfd0_hd.jpg

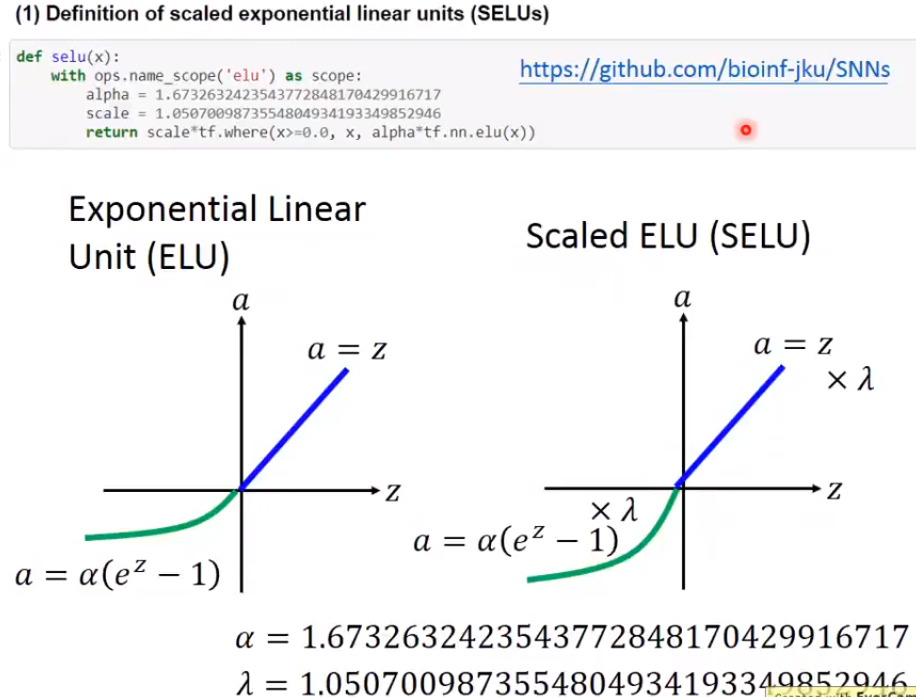
**四、激活函数**

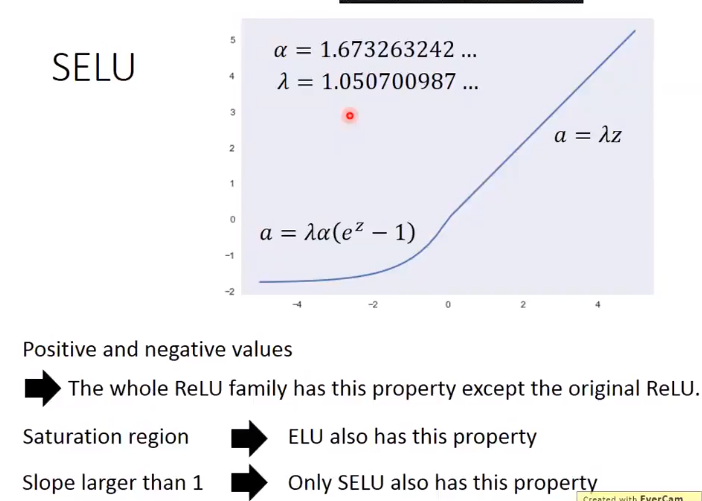
Sigmoid、tanh、Maxout、ReLU、Leaky ReLU、Parametric ReLU、Randomized ReLU、ELU、SELU(需要将输入X和权重W都初始化为均值0，方差1)、Swish

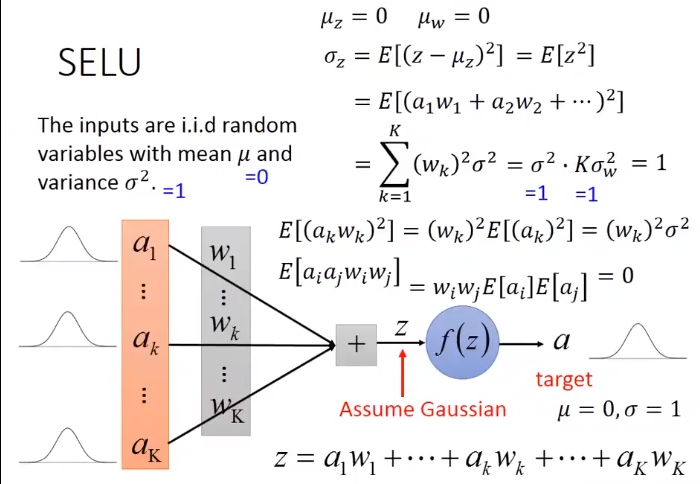


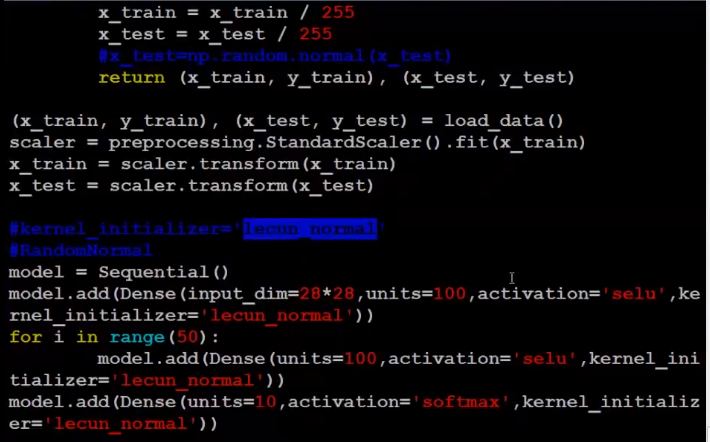


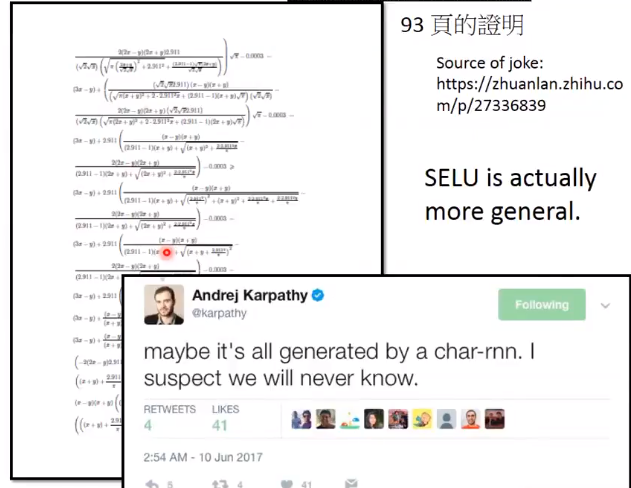


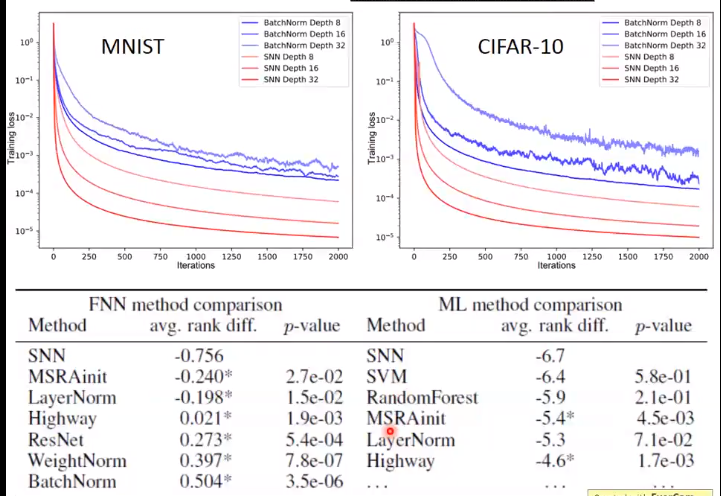


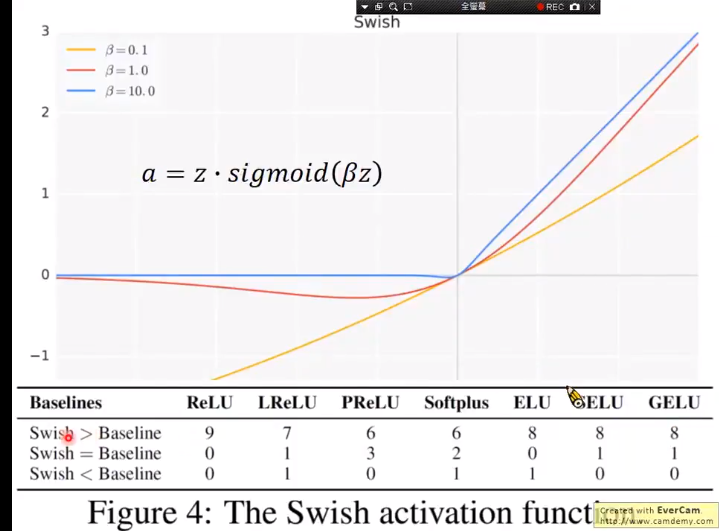












**五、损失函数**

欧式距离损失函数（Euclidean Loss）

这个loss的具体含义就是所有样本估计值和预测值的欧式距离平方的均值，也就是均方根误差(MSE)。

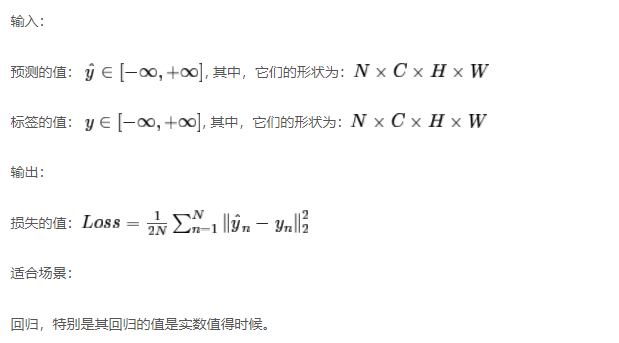
http://album.sina.com.cn/pic/004j58Jbzy6NjXF5YgG42

假设模型结果与测量值 误差满足，均值为0，方差为1的高斯分布，即正态分布。这个假设是靠谱的，符合一般客观统计规律。则数据x与y的条件概率为：

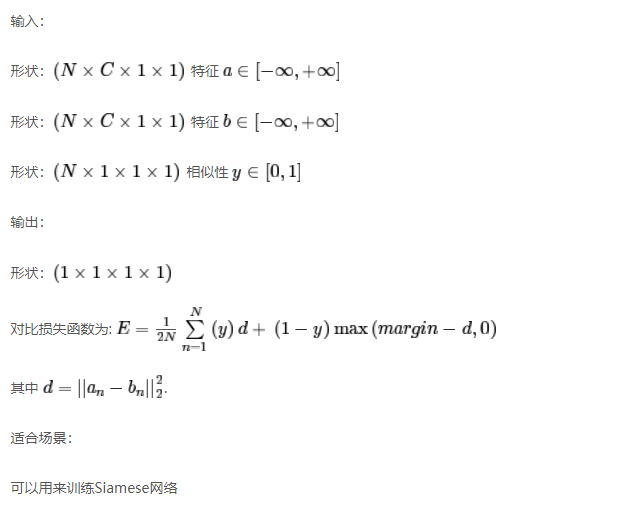
http://album.sina.com.cn/pic/004j58Jbzy6NjXFqUu2c6

若使 模型与测量数据最接近，那么其概率积就最大。概率积，就是概率密度函数的连续积，这样，就形成了一个最大似然函数估计。对最大似然函数估计进行推导，就得出了求导后结果： 平方和最小公式。

MSE误差的优点是执行简单，较容易理解，缺点就是强制预测和标注要exactly的匹配，也就是一个非0即1的概念，本节点和其他节点是独立的，这样带来的问题是会导致最后的train出来的model有可能并不是那么准确。



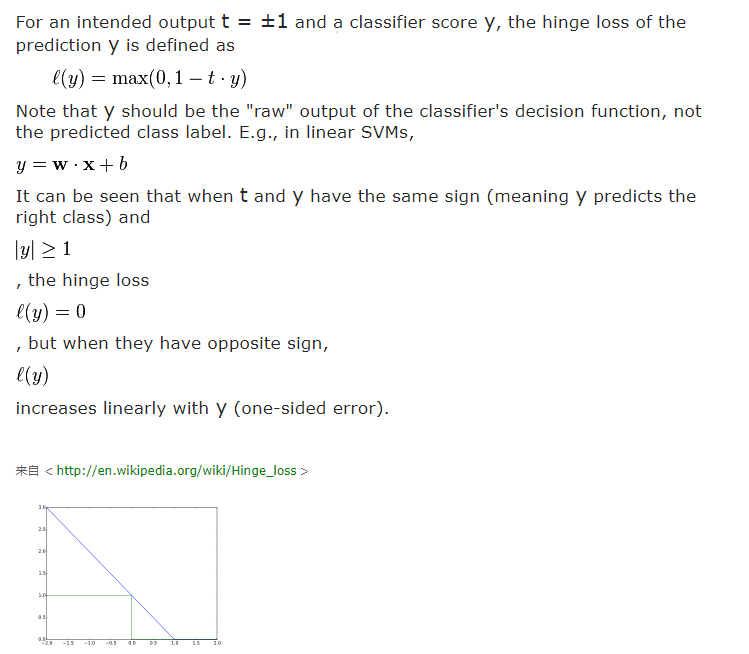
对比损失函数（Contrastive loss）

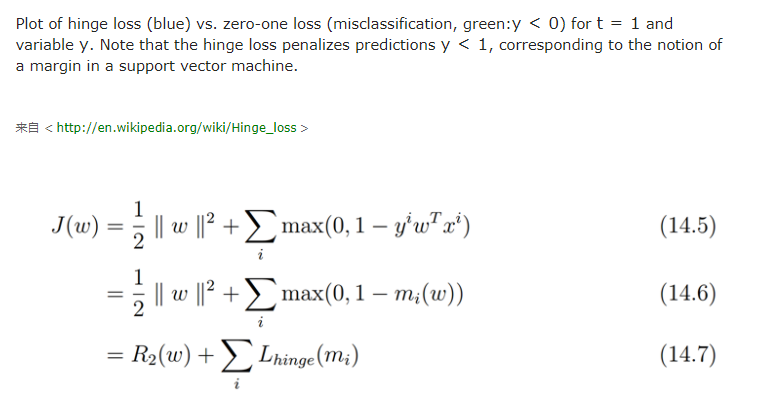


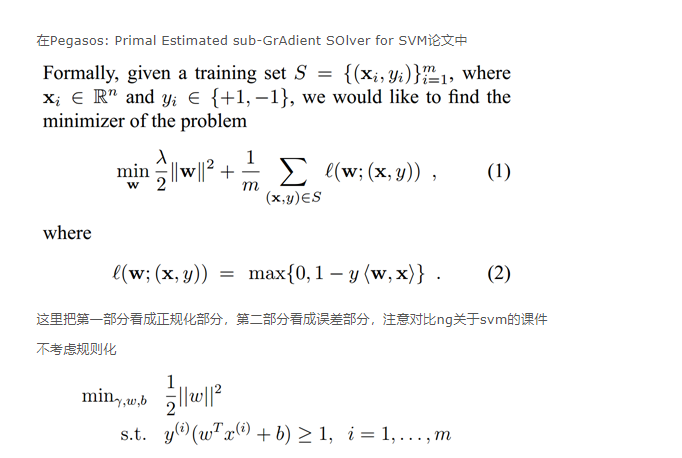
辛基（折页）损失函数（Hinge Loss）

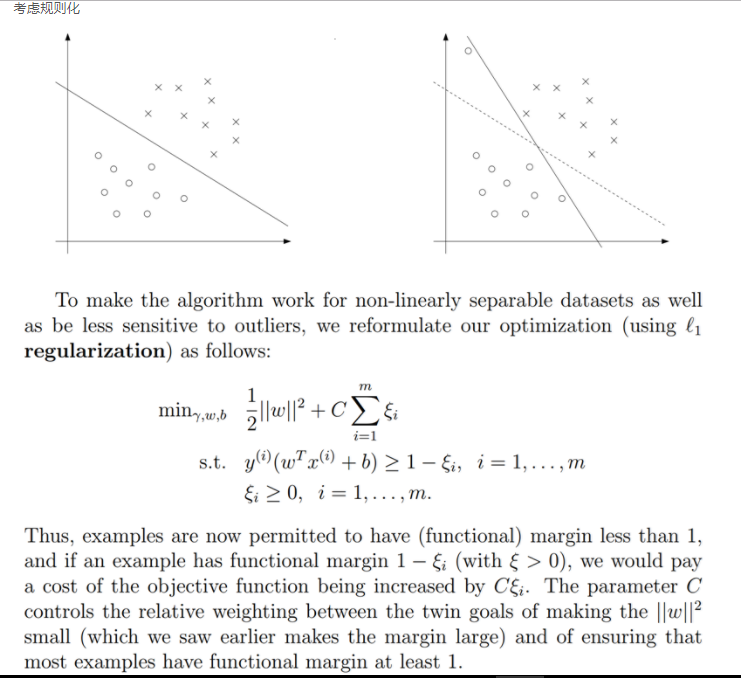
http://album.sina.com.cn/pic/004j58Jbzy6NjYex6BVdf

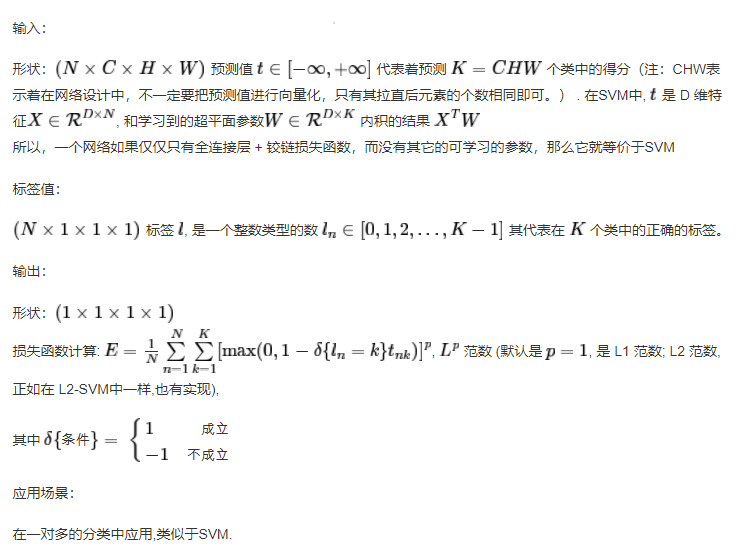
求和后便成了Hinge loss  
     
这个loss就是SVM用到的loss。Hinge loss就是0-1 loss的改良版，这个改良主要在两个方面，一个是在t.y在【0 1】之间不再是采用hard的方式，而是一个soft的方式。另外一个就是在【-inf，0】之间不再采用固定的1来定义能量的损失，而是采用一个线性函数对于错误分类的情况进行惩罚。





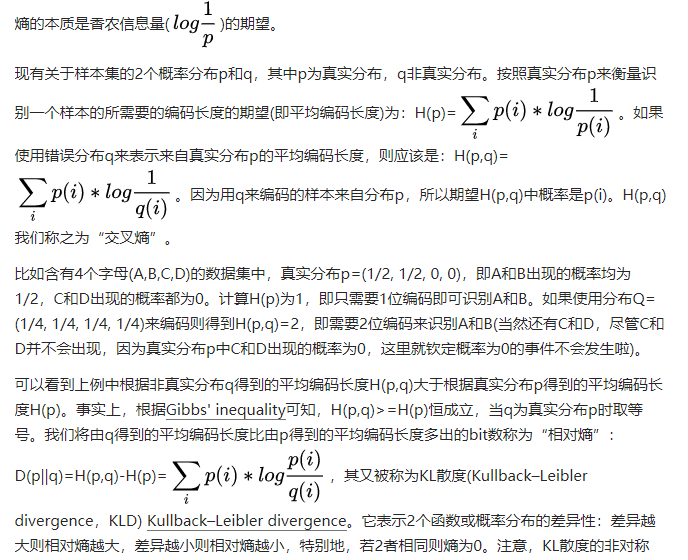


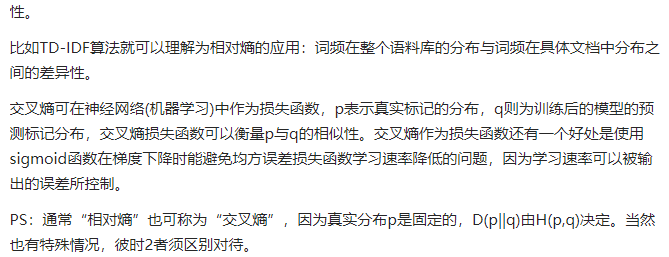


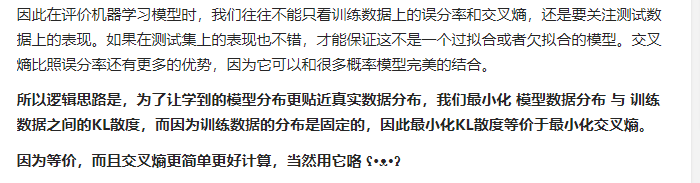


信息增益损失函数（InformationGain Loss）

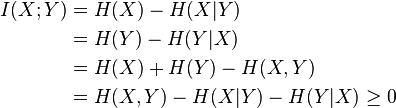
信息熵是对于信息的一种量化，是对于某种系统信息的一种量度，熵就是系统的平均信息量，也可以理解为某种信息出现的概率。

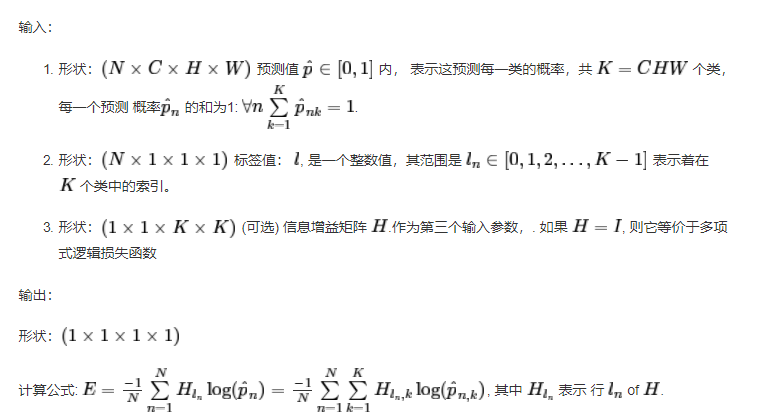






信息增益是在加入一个条件后，所得到的熵和原来的熵的差，具体形式如下：

z



多项式逻辑损失函数（Multinomial Logistic Loss）

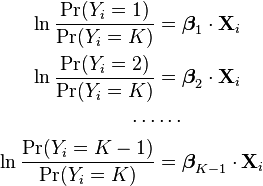
如果一个事件发生的概率为：



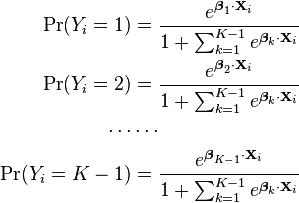
那么事件几率的概念是指的发生的概率与不发生的概率的比值，取对数则称为对数几率(log odds)。  
the logit ：



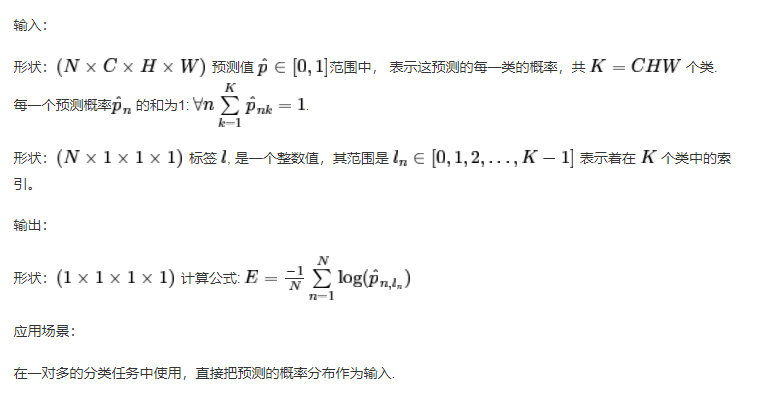
对于对元的分布，采用base-line logit，如下所示。(从某种意义上二元的logit是多元的一个特例)



通过概率和为1这样一个约束，可以得到Pr(Y=K)的表示，从而推导出K-1个分布的公式，如下所示：

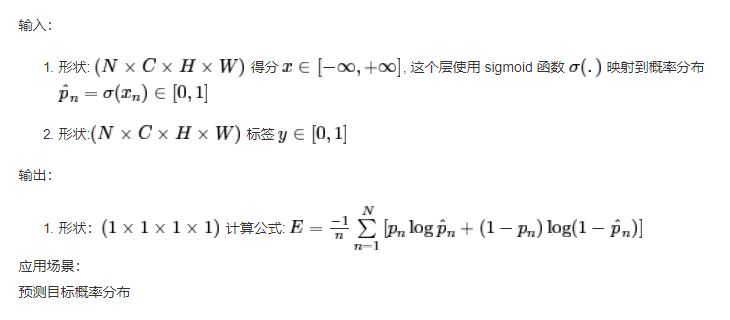


所有样本的概率和就是最后使用到的MULTINOMIAL\_LOGISTIC\_LOSS

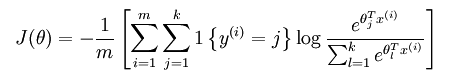


Sigmoid 交叉熵损失函数（Sigmoid Cross Entropy Loss）





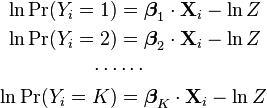
**Softmax+损失函数(Softmax With Loss)**



下面看一下基本的softmax的推导过程  
 在概率论中，一个归一化常数的作用是所有的密度函数满足和为1这样一个条件。

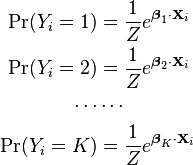


此处采用log的形式描述，其中Z为归一化的常数



C:\Users\abadcandy\Desktop\1.png

得到



根据和为1这样的约束可以得到

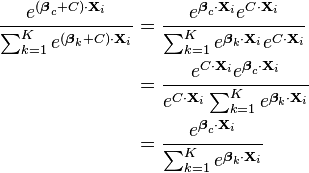


softmax 则采用了连续函数来进行函数的逼近，最后采用概率的形式进行输出，这样就弱化了EUCLIDEAN\_LOSS中带来的问题。

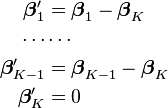


softmax中，各个节点的输出是一个归一化后的概率值，这个值随着每次迭代是动态变化的。

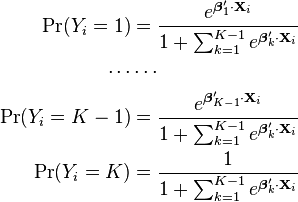
**softmax和multinomial logistic的统一性**  
最后得到softmax的形式，在所有的beta参数中，只有K-1个是独立的，对于所有添加一个常量C不改变softmax的值，证明如下：



如果这个把最后一个第K个样本的beta设置为常数，得到如下的变换：



带入原来的公式，可以得到：



这个输出就是multinomial logistic.。

**Center Loss损失函数**

**Keras中categorical crossentropy(softmax loss)的问题**

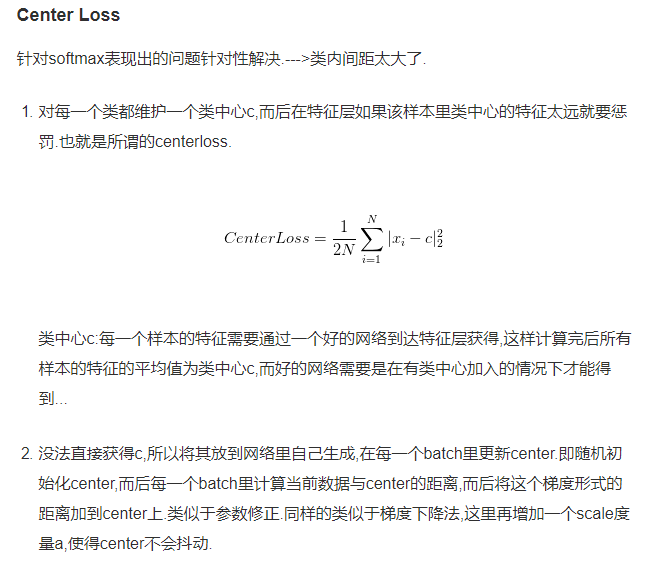
缺点:

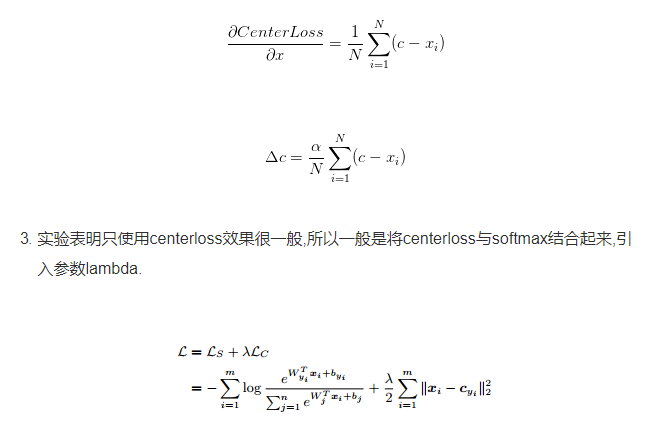
1. 从聚类角度看,提取的特征并不好.很多情况下类内间距甚至要大于类间间距.我们期望特征不仅可分，而且要求差异大，特征学习需要保证提取的特征有识别度。
2. 占据的面积有点大.通常情况下,我们希望每一类只占很小一部分.因为手写字符很多啊,这些数字就占了这么大地方,如果新来了英文字母呢...也就是我们期望模型能识别出在训练数据的标签中没有的分类。特征学习需要保证提取的特征具有普适性.
3. softmax会使得模型过度自信,分类结果基本非1即0,上图里有些点在边界但是softmax认为已经可以了,根本没必要再修正.同时softmax这种特性使得基本上没有办法去设置诸如可信度等度量.(场景?)

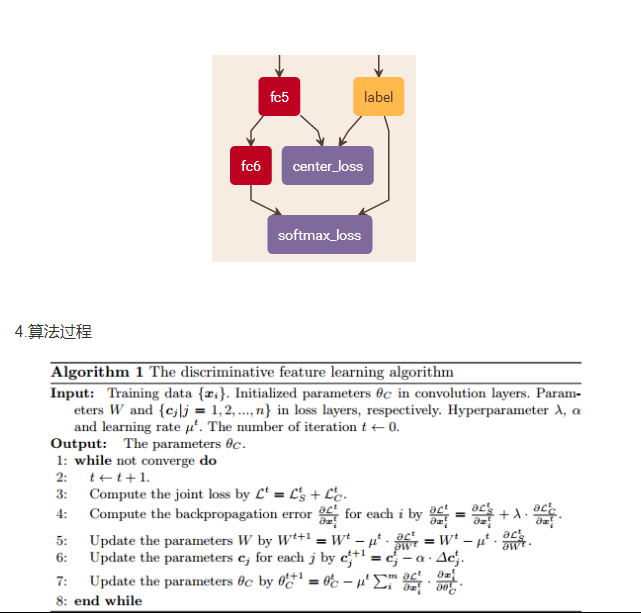
原因?举例:

最后一层全连接层输出V=[x1,x2,x3],真实标签是[1,0,0].那么假设V=[x1,x2,x3]是[3.1,3,3],那么softmax的公式使得其只需要V的模长增加倍数即可以降低loss损失.这太容易(只需要增大参数即可)使得网络往往就是这样做的.而不是我们通常想要的那样去努力降低x2,x3的相对于x1的值如[3.1,1,1]这样.这也是所以L2正则会缓解过拟合的一个原因.

解决办法:很多,如故意让softmax也去模拟下均匀分布输出而不仅仅是one\_hot.这里只涉及其中一种也就是centerloss.

****

****

****

**六、Ensemble**

数据集大：划分成多个小数据集，学习多个模型进行组合

数据集小：利用Bootstrap方法进行抽样，得到多个数据集，分别训练多个模型再进行组合

**Bagging**

Bagging是bootstrap aggregating的简写。先说一下bootstrap，bootstrap也称为自助法，它是一种有放回的抽样方法，目的为了得到统计量的分布以及置信区间。具体步骤如下

采用重抽样方法（有放回抽样）从原始样本中抽取一定数量的样本

根据抽出的样本计算想要得到的统计量T

重复上述N次（一般大于1000），得到N个统计量T

根据这N个统计量，即可计算出统计量的置信区间

在Bagging方法中，利用bootstrap方法从整体数据集中采取有放回抽样得到N个数据集，在每个数据集上学习出一个模型，最后的预测结果利用N个模型的输出得到，具体地：分类问题采用N个模型预测投票的方式，回归问题采用N个模型预测平均的方式。

例如随机森林（Random Forest）就属于Bagging。随机森林简单地来说就是用随机的方式建立一个森林，森林由很多的决策树组成，随机森林的每一棵决策树之间是没有关联的。

在我们学习每一棵决策树的时候就需要用到Bootstrap方法。在随机森林中，有两个随机采样的过程：对输入数据的行（数据的数量）与列（数据的特征）都进行采样。对于行采样，采用有放回的方式，若有N个数据，则采样出N个数据（可能有重复），这样在训练的时候每一棵树都不是全部的样本，相对而言不容易出现overfitting；接着进行列采样从M个feature中选择出m个（m<<M）。最近进行决策树的学习。

预测的时候，随机森林中的每一棵树的都对输入进行预测，最后进行投票，哪个类别多，输入样本就属于哪个类别。这就相当于前面说的，每一个分类器（每一棵树）都比较弱，但组合到一起（投票）就比较强了。

**Boosting**

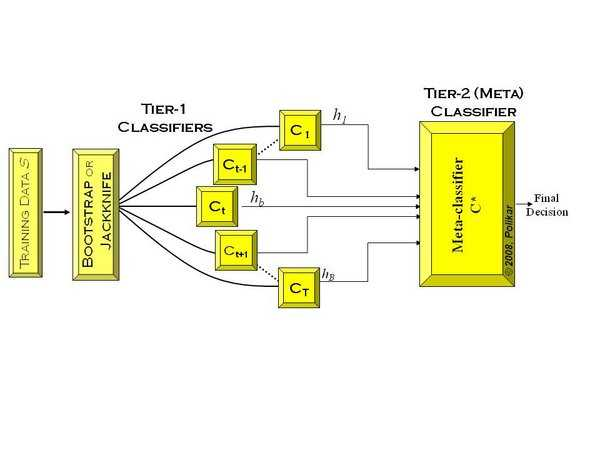
提升方法（Boosting）是一种可以用来减小监督学习中偏差的机器学习算法。主要也是学习一系列弱分类器，并将其组合为一个强分类器。Boosting中有代表性的是AdaBoost（Adaptive boosting）算法：刚开始训练时对每一个训练例赋相等的权重，然后用该算法对训练集训练t轮，每次训练后，对训练失败的训练例赋以较大的权重，也就是让学习算法在每次学习以后更注意学错的样本，从而得到多个预测函数。具体可以参考《统计学习方法》。

之前提到过的GBDT（Gradient Boost Decision Tree)也是一种Boosting的方法，与AdaBoost不同，GBDT每一次的计算是为了减少上一次的残差，GBDT在残差减少（负梯度）的方向上建立一个新的模型。可以参考Gradient Boosting - 知乎专栏。

**Stacking**

Stacking方法是指训练一个模型用于组合其他各个模型。首先我们先训练多个不同的模型，然后把之前训练的各个模型的输出为输入来训练一个模型，以得到一个最终的输出。理论上，Stacking可以表示上面提到的两种Ensemble方法，只要我们采用合适的模型组合策略即可。但在实际中，我们通常使用logistic回归作为组合策略。

如下图，先在整个训练数据集上通过bootstrap抽样得到各个训练集合，得到一系列分类模型，称之为Tier 1分类器（可以采用交叉验证的方式学习），然后将输出用于训练Tier 2 分类器。



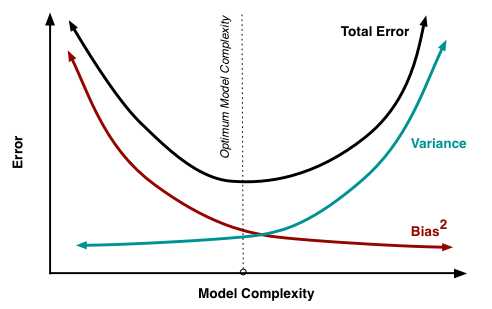
**Bagging与Boosting**

Bagging和Boosting采用的都是采样-学习-组合的方式，但在细节上有一些不同，如

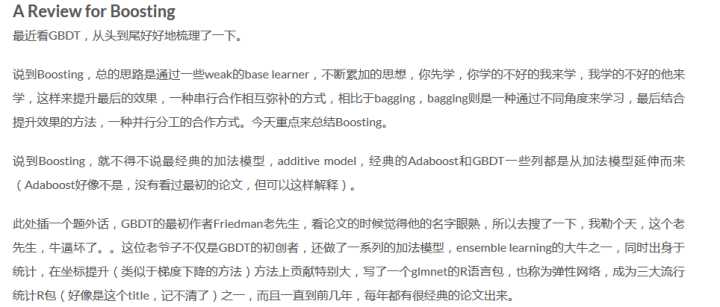
Bagging中每个训练集互不相关，也就是每个基分类器互不相关，而Boosting中训练集要在上一轮的结果上进行调整，也使得其不能并行计算

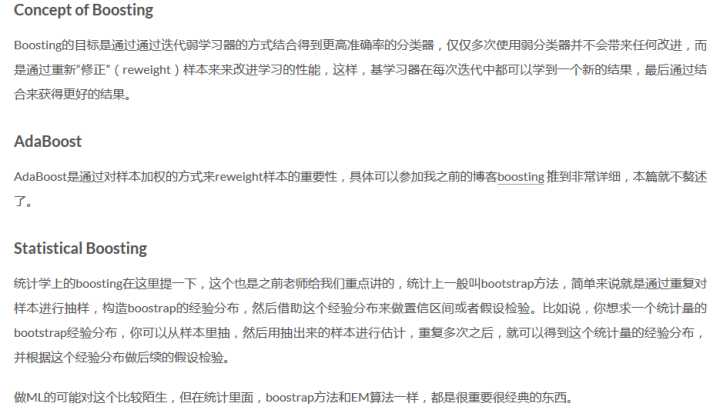
Bagging中预测函数是均匀平等的，但在Boosting中预测函数是加权的

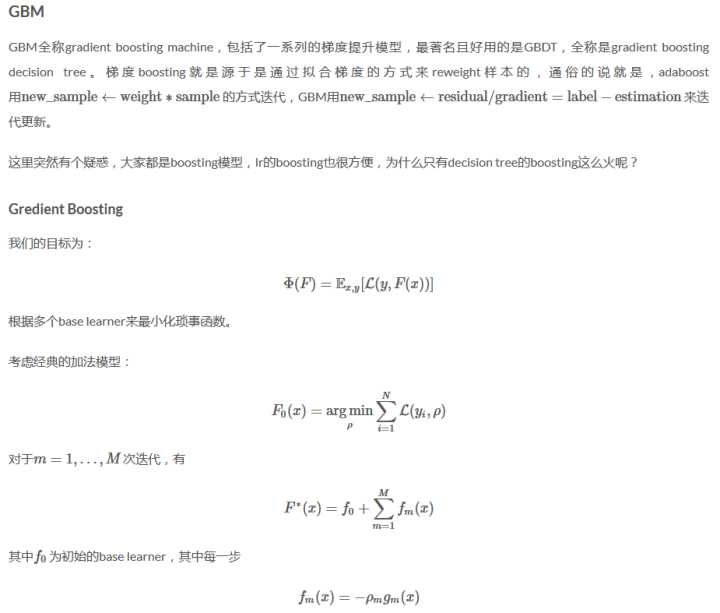
在算法学习的时候，通常在bias和variance之间要有一个权衡。bias与variance的关系如下图，因而模型要想达到最优的效果，必须要兼顾bias和variance，也就是要采取策略使得两者比较平衡。

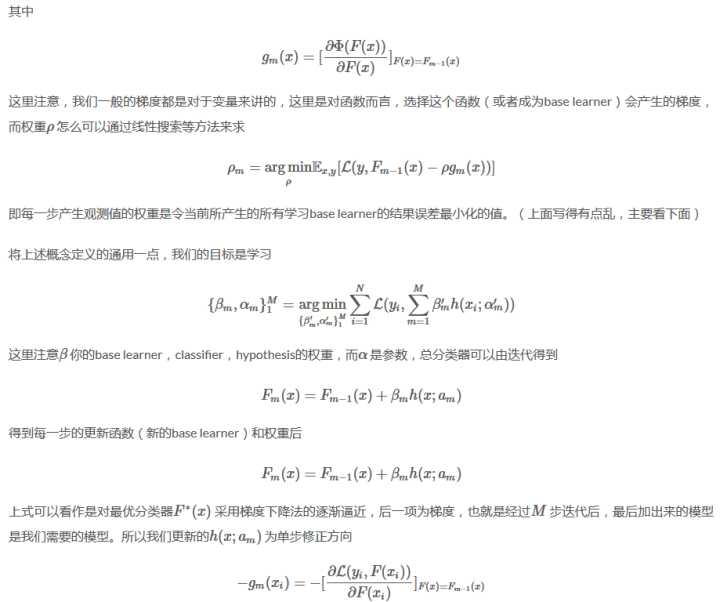


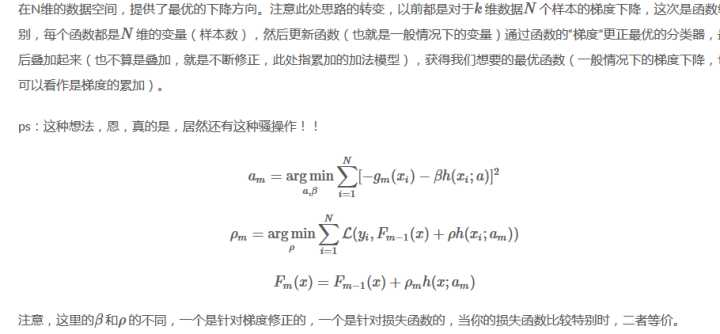
从算法来看，Bagging关注的是多个基模型的投票组合，保证了模型的稳定，因而每一个基模型就要相对复杂一些以降低偏差（比如每一棵决策树都很深）；而Boosting采用的策略是在每一次学习中都减少上一轮的偏差，因而在保证了偏差的基础上就要将每一个基分类器简化使得方差更小。

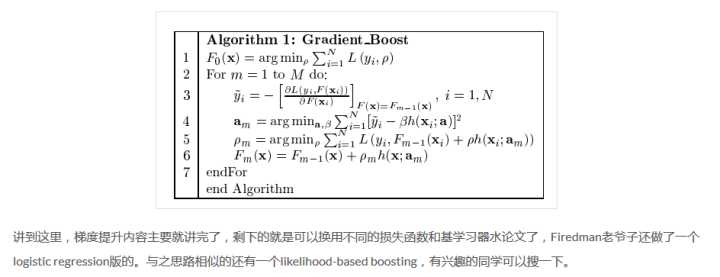


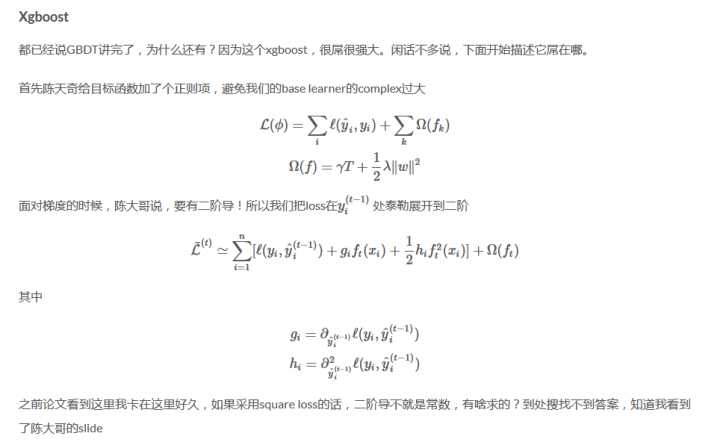


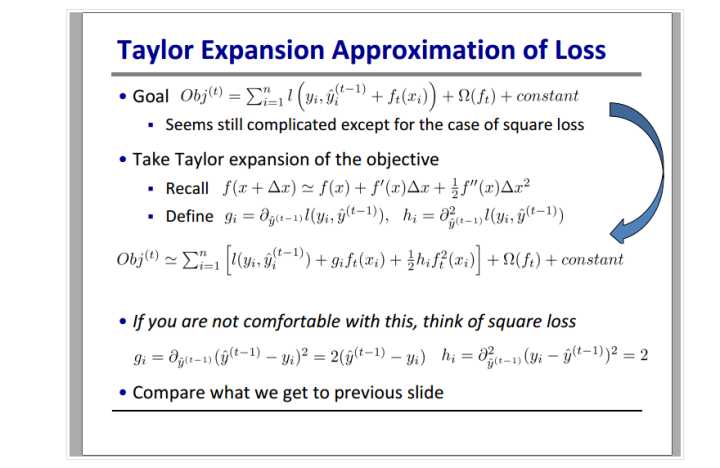










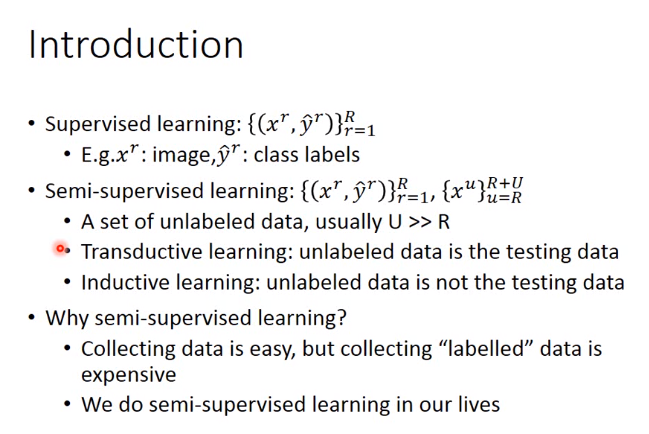








**七、Semi-supervised Learning**

****

**Transductive表示用到了测试数据的特征，但没有用它的label。**

**八、Unsupervised Learning**

**Autoencoder系列、RBM/DBN**