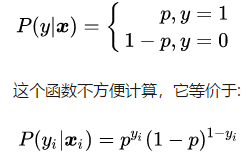
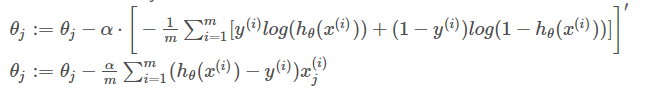
* **LR**

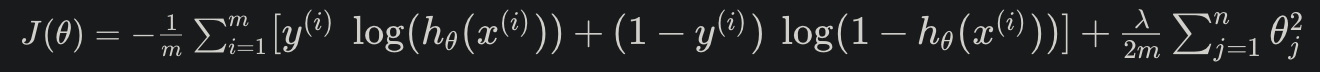
逻辑回归是在数据服从伯努利分布的假设下，通过极大似然的方法，运用梯度下降法来求解参数，从而达到将数据二分类的目的。y 值大于这个阈值的是一类，y 值小于这个阈值的是另外一类。一般情况下选取 0.5 作为阈值。

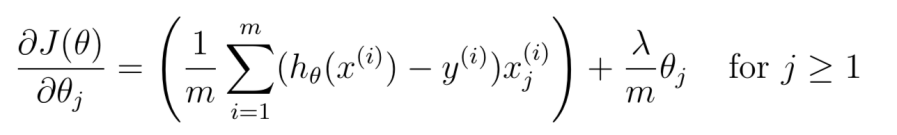
 

**LR损失函数**



**正则化的损失函数**





**为什么可以用梯度下降法？**

因为逻辑回归的损失函数L是一个连续的凸函数。这样的函数的特征是，它只会有一个全局最优的点，不存在局部最优。对于GD跟SGD最大的潜在问题就是它们可能会陷入局部最优。然而这个问题在逻辑回归里面不存在。

**为什么不适用平方损失函数？**

在使用 Sigmoid 函数作为正样本的概率时，同时将平方损失作为损失函数，这时所构造出来的**损失函数是非凸的，不容易求解，容易得到其局部最优解**。

使用极大似然，其目标函数就是对数似然函数，该损失函数是关于未知参数的**高阶连续可导的凸函数，便于求其全局最优解。**

**LR优缺点**

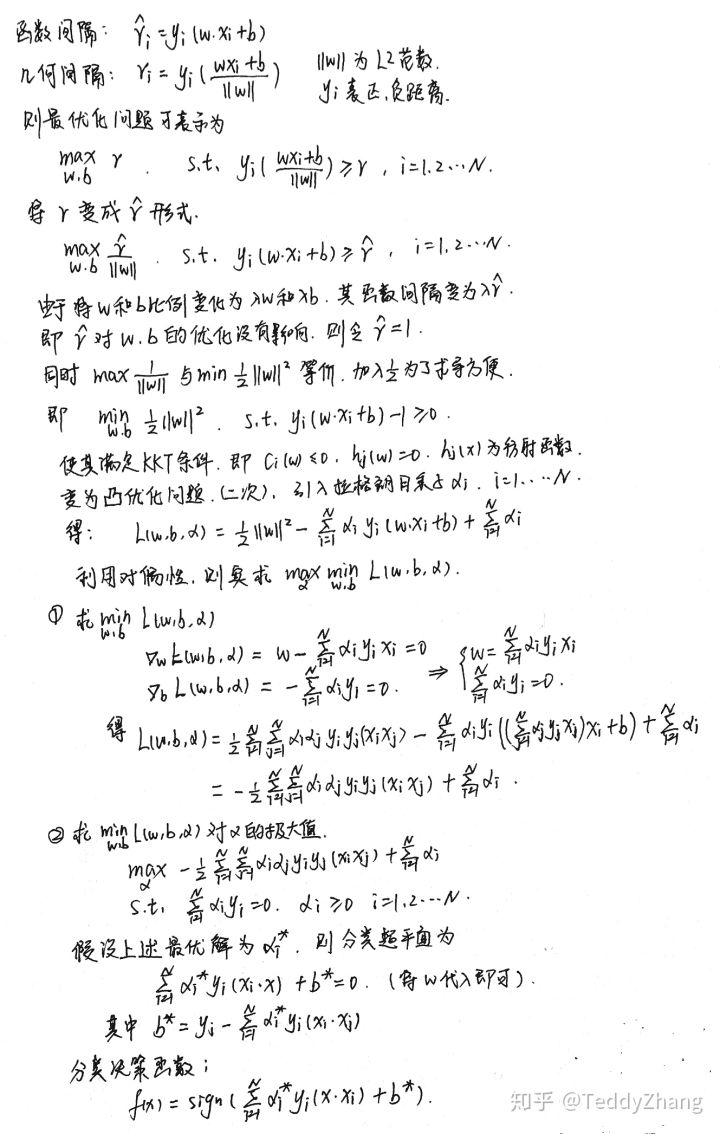
**优点：**

1. 形式简单，模型的可解释性非常好。从特征的权重可以看到不同的特征对最后结果的影响，某个特征的权重值比较高，那么这个特征最后对结果的影响会比较大。
2. 模型效果不错。在工程上是可以接受的（作为 baseline），如果特征工程做的好，效果不会太差，并且特征工程可以并行开发，大大加快开发的速度。
3. 训练速度较快。分类的时候，计算量仅仅只和特征的数目相关。并且逻辑回归的分布式优化 SGD 发展比较成熟。

**缺点：**

1. 准确率欠佳。因为形式非常的简单，而现实中的数据非常复杂，因此，很难达到很高的准确性。
2. 很难处理数据不平衡的问题。如果正负样本比例 10000:1，把所有样本都预测为正也能使损失函数的值比较小。但是作为一个分类器，它对正负样本的区分能力不会很好。

* SVM



SVM的目的：寻找到一个超平面使样本分成两类，并且间隔最大。

**为什么要将求解 SVM 的原始问题转换为其对偶问题**

1. 对偶问题将原始问题中的约束转为了对偶问题中的等式约束
2. 方便核函数的引入
3. 改变了问题的复杂度。由求特征向量w转化为求比例系数a，在原始问题下，求解的复杂度与样本的维度有关，即w的维度。在对偶问题下，只与样本数量有关。

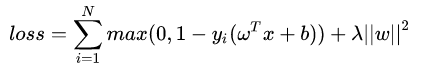
**SVM怎么防止过拟合**

引入松弛变量，既希望松弛变量存在以解决异常点问题，又不希望松弛变量太大导致分类解决太差。

在损失函数中加上正则项

**SVM损失函数**

合页损失函数 Hinge Loss



第一项为经验风险，度量了模型对训练数据的拟合程度

第二项为结构风险，也称正则化项，度量了模型自身的复杂度，可以降低过拟合风险

**核函数**

SVM可以通过核方法进行非线性分类。常见的核函数有：多项式核、径向基函数核、拉普拉斯核、Sigmoid核。

**KKT条件**

KKT条件将Lagrange乘数法所处理涉及等式的约束优化问题推广至不等式。若满足KKT条件，则可以将原问题转化为对偶问题。

**LR和SVM比较**

联系：

1、LR和SVM一般都用于处理线性二分类问题（在改进的情况下可以处理多分类问题）

2、两个方法都可以增加不同的正则化项，如l1、l2等等。

3、LR和SVM都可以用来做非线性分类，只要加核函数就好。

4、LR和SVM都是线性模型，当然这里我们不要考虑核函数

5、都属于判别模型

区别：

1、LR是参数模型，SVM是非参数模型。

2、从目标函数来看，区别在于逻辑回归采用的是logistical loss，SVM采用的是hinge loss

3、逻辑回归相对来说模型更简单，大规模线性分类时比较方便。而SVM的理解和优化相对来说复杂一些，SVM转化为对偶问题后,分类只需要计算与少数几个支持向量的距离,这个在进行复杂核函数计算时优势很明显,能够大大简化模型和计算。

4、SVM不直接依赖数据分布，而LR则依赖，因为SVM只支持向量有关，而LR和所有点都有关系。

5、SVM本身是结构风险最小化模型，而LR是经验风险最小化模型

怎么选模型：

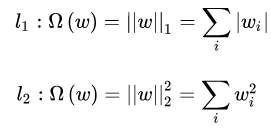
1. 如果Feature的数量很大，跟样本数量差不多，这时候选用LR或者是Linear Kernel的SVM

2. 如果Feature的数量比较小，样本数量一般，不算大也不算小，选用SVM+Gaussian Kernel

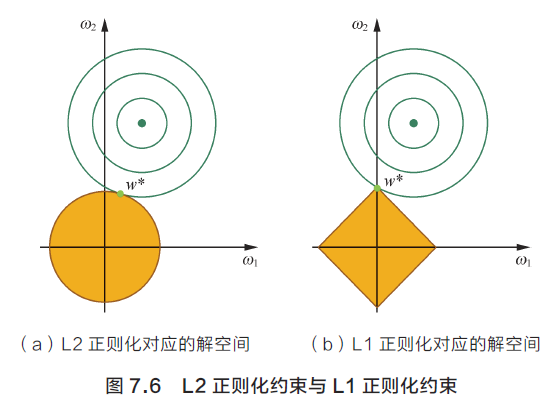
3. 如果Feature的数量比较小，而样本数量很多，需要手工添加一些feature变成第一种情况

* **正则化**

**L1和L2正则化**

正则化（Regularization）是机器学习中一种常用的技术，其主要目的是控制模型复杂度，减小过拟合。最基本的正则化方法是在原目标（代价）函数 中添加惩罚项，对复杂度高的模型进行“惩罚”。

**L1和l2正则，为什么l1产生稀疏矩阵，l2可以防止过拟合**

黄色的部分是L2 和L1 正则项约束后的解空间，绿色的等高线是凸优化问题中目标函数的等高线，如图7.6 所示。L2 正则项约束后的解空间是圆形，而L1 正则项约束的解空间是多边形。多边形的解空间更容易在尖角处与等高线碰撞出稀疏解。

“带正则项”和“带约束条件”是等价的。为了约束w 的可能取值空间从而防止过拟合，为该最优化问题加上一个约束，就是w 的L2 范数的平方不能大于m。

L2 正则化相当于为参数定义了一个圆形的解空间

L1 正则化相当于为参数定义了一个棱形的解空间。

约束条件下的最优解一定是在解空间的边界上，而L1“棱角分明”的解空间显然更容易与目标函数等高线在角点碰撞，从而产生稀疏解。

L1正则化的解具有稀疏性，可用于特征选择。一定程度上也可以防止过拟合。一般而言，只有少部分特征对模型有贡献，大部分特征对模型没有贡献或者贡献很小，稀疏参数的引入，使得一些特征对应的参数是0，所以就可以剔除可以将那些没有用的特征，从而实现特征选择。

L2正则化的解都比较小，抗扰动能力强。

L2通常倾向让权值尽可能小，最后构造一个所有参数都比较小的模型。**因为一般认为参数值小的模型比较简单，能适应不同的数据集，也在一定程度上避免了过拟合现象**。参数足够小，数据偏移得多一点也不会对结果造成什么影响，可以说“抗扰动能力强”，模型也就能够适应不同的数据集，也就是泛化能力强，所以一定程度上避免过拟合。

* 评估指标

**True positive (TP)**

· 真实值为Positive，预测正确（预测值为Positive）

**True negative (TN)**

· 真实值为Negative，预测正确（预测值为Negative）

**False positive (FP)**

· 真实值为Negative，预测错误（预测值为Positive），第一类错误， Type I error。

**False negative (FN)**

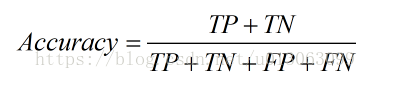
· 真实值为Positive，预测错误（预测值为 Negative），第二类错误， Type II error。

精度 Acc ：预测正确的样本的占总样本的比例

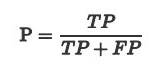
缺陷：

对于有倾向性的问题（两种情况重要性不同），往往不能用精度指标来衡量。

对于样本类别数量严重不均衡的情况，也不能用精度指标来衡量。



准确率 Precision：预测的正样本中预测正确的比例



召回率 Recall：预测正确的正样本占所有正样本的比例



F-Score

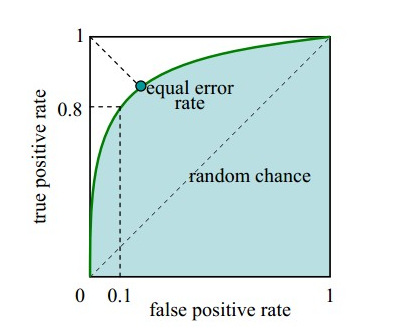
Precision和Recall 是互相影响的，理想情况下肯定是做到两者都高，但是一般情况下Precision高、Recall 就低， Recall 高、Precision就低。为了均衡两个指标，我们可以采用Precision和Recall的加权调和平均（weighted harmonic mean）来衡量。



**ROC曲线**

x轴为假阳率预测为正但实际为负的样本占所有负例样本的比例

y轴为真阳率预测为正且实际为正的样本占所有正例样本的比例



ROC曲线越靠拢（0,1）点，越偏离45度对角线越好。

**AUC值**

ROC曲线下的面积，AUC更大的分类器效果更好

**AUC的物理意义正样本的预测结果大于负样本的预测结果的概率。**

**为什么说 ROC 和AUC都能应用于非均衡的分类问题？**

ROC曲线只与横坐标 (FPR) 和 纵坐标 (TPR) 有关系 。我们可以发现TPR只是正样本中预测正确的概率，而FPR只是负样本中预测错误的概率，**和正负样本的比例没有关系。**因此 ROC 的值与实际的正负样本比例无关，因此既可以用于均衡问题，也可以用于非均衡问题。而 AUC 的几何意义为ROC曲线下的面积，因此也和实际的正负样本比例无关。

* **过拟合与欠拟合**

**如何解决过拟合**

过拟合表现在训练好的模型在训练集上效果很好，但是在测试集上效果差，也就是模型的泛化能力弱。

过拟合主要有两个原因造成的，数据太少和模型太复杂。

1. 数据集扩增
2. 改进模型
   1. early stopping：运行优化方法直到若干次在验证集上的验证误差没有提升时候停止。
   2. 正则化：L1 / L2 正则化
   3. Dropout
   4. Batch Normalization
3. 模型融合，Bagging / Boosting

**交叉验证只能检验过拟合，不能抑制过拟合**

**解决欠拟合**

1. 添加其他特征。有时候模型出现欠拟合的时候是因为特征项不够导致的，可以添加其他特征项来很好地解决。
2. 添加多项式特征
3. 减少正则化参数
4. 尝试更高级更复杂的模型

**不平衡数据如何处理**

1. 使用正确的评估指标
   1. F1 score
   2. AUC
2. 采样
   1. 欠采样：通过减少冗余类的大小来平衡数据集，在多数类中随机选择相等数量的样本
   2. 过采样：增加稀有样本的数量来平衡数据集
   3. 分配权重
   4. SMOTE采样：在局部区域通过K-近邻生成了新的样本，可以降低过拟合风险，缺点是运算开销大
   5. 对多数类进行聚类，只保留质心（数量与少数样本一样）

**判别式模型**直接对 P(Y|X) 建模，直接根据X特征来对Y建模训练，对所有样本只构建一个模型，确定总体判别边界。LR、SVM、感知机、神经网络等

**生成式模型**在训练阶段只对P(X,Y) 建模，然后对新数据计算 P(Y|X) ，导出Y。对每个label 都需要建模，最终选择最优概率的label为结果，所以没有什么判别边界。NB、LDA等

**交叉验证**

1. 留一法：每次取出一个数据作为测试集的唯一元素，而其他n-1个数据都作为训练集用于训练模型和调参。结果就是我们最终训练了n个模型，每次都能得到一个MSE。而计算最终test MSE则就是将这n个MSE取平均。
2. K折交叉验证：将数据集分为K组，每次取一组做测试集，其余数据做训练集。对得到的K个MSE取平均。

K越大，每次投入的训练集的数据越多，模型的Bias越小，每次选取的训练集差异越小，模型的Variance越大

K一般选择5或10

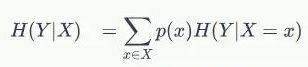
* **决策树**

决策树是一种用于分类和回归的算法。决策树算法的核心是如何选择最优属性。

**熵**度量了事物的不确定性，越不确定的事物，它的熵就越大。



**条件熵**：在已知随机变量X的条件下随机变量Y的不确定性。



**信息增益**：得知特征X的信息而使得类Y的信息的不确定性减少的程度。信息增益大的特征具有更强的分类能力。

互信息：熵H（Y）与条件熵H（Y|X）之差。信息增益等价于训练数据集中类与特征的互信息。



**信息增益比**：数据集的熵大，则信息增益大。可以使用信息增益比进行校正。



ID3算法的核心是在决策树各个结点上应用信息增益准则选择特征

C4.5算法是应用信息增益比来选择特征

**决策树防止过拟合**

1. 预剪枝：在生成决策树的过程中，如果信息增益或信息增益比小于阈值，就停止分裂。

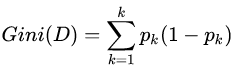
缺点是，阈值难以选择，容易造成欠拟合。

1. 后剪枝：先生成一颗完全生长的决策树，再自底向上用节点替换子树

后剪枝的效果更好

CART 决策树全称为分类回归树，CART是一棵二叉树，每次把数据切成两份，分别进入左子树、右子树。

**CART分类时**，使用基尼指数（Gini）来选择最好的数据分割的特征，gini描述的是纯度，与信息熵的含义相似。CART中每一次迭代都会降低GINI系数，候选集合中选择使得划分后基尼指数最小的属性作为最优化分属性。

k代表类别

回归树的分裂使用最小化均方误差为标准

**随机森林**

随机森林，是用随机的方式建立一个森林，森林里面有很多的决策树组成，随机森林的每一棵决策树之间是没有关联的。在得到森林之后，当有一个新的输入样本进入的时候，就让森林中的每一棵决策树分别进行一下判断，对于分类问题进行投票表决，对于回归问题计算平均值

**采样：**随机森林对输入的数据要进行行、列的采样。由于随机采样保证了随机性，就算不剪枝也不会出现过拟合

**优点：**

1. **不容易陷入过拟合**
2. **抗噪能力强**
3. **可以并行，训练速度快**
4. **可以处理高维数据**

* 集成学习

**Bagging / Boosting / Stacking**

集成学习的思路是通过合并多个模型来提升机器学习性能，相比于单个模型通常能够获得更好的预测结果。

集成学习可以分为三大类：

用于减少方差的bagging

用于减少偏差的boosting

用于提升预测结果的stacking

**Boosting**

提升方法就是从弱学习算法出发，使用全部数据反复学习，得到一系列弱分类器（基本分类器），然后组合这些弱分类器，构成一个强分类器。Adaboost、GBDT

**Bagging**

每轮训练从原始样本集中抽取训练集，训练出不同的模型，对分类问题使用投票，对回归问题求平均。随机森林

**Stacking**是通过一个元分类器或者元回归器来整合多个分类模型或回归模型的集成学习技术。基础模型利用整个训练集做训练，元模型将基础模型的特征作为特征进行训练。

1. 有T个不同模型，根据数据，各自独立训练。每个模型输出结果为ht，其中t= 1,2,3...T
2. 根据每个模型预测，创建新的『训练集』，训练集的数据（X）是预测结果,Y是实际结果
3. 再训练一个classifier，这里叫元分类器，来从各个预测中再生成最后预测。

**Boosting和Bagging的区别**

1. 样本选择

Boosting 每轮的训练集不变，但是样本的权重会根据上轮的结果调整

Bagging从训练集中有放回的抽取，每轮训练集之间的独立的

1. 样本权重

Boosting 样本的权重会根据上轮的结果调整，错误率越大，权重越大

Bagging 均匀采样，每个样本的权值相等

1. 并行计算

Boosting 每个基分类器只能顺序生成

Bagging 可以并行生成

**Bagging + 决策树 = 随机森林**

**AdaBoost + 决策树 = 提升树**

**Gradient Boosting + 决策树 = GBDT**

stacking和Bagging的区别：

1. Bagging每个弱学习器是同一种模型，stacking使用的是不同的模型
2. Bagging每轮使用部分数据训练，stacking每个学习器都用了全部训练数据

* AdaBoost

利用前一轮迭代弱学习器的误差率来更新训练集的权重调整错分的分类器权重进行学习，形成“基分类器”加性模型。

**Adaboost算法流程**

1. 初始化数据集的权重，所有数据的权值相同

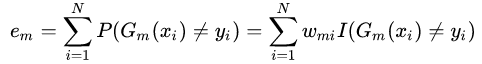


1. 假设训练轮次为M（也是基分类器的个数）

2.1使用具有权值分布的训练数据集学习，得到本轮的基分类器



2.2得到分类误差率



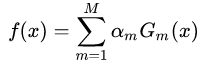
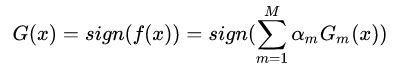
2.3基分类器系数，在本轮分类误差率越小的基本分类器在最终分类器中的作用越大。



2.4更新数据集权重，误分类样本的权值得以扩大，而被正确分类样本的权值缩小。



1. 构建基分类器的线性组合，得到最终的分类器

* GBDT

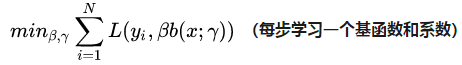
每轮迭代产生一个弱分类器，每个分类器在上一轮分类器的残差基础上进行训练。

对弱分类器的要求一般是足够简单，并且是低方差和高偏差的，因为训练的过程是通过降低偏差来提高最终分类器的精度。

弱分类器一般会选择为 CART。

采用加法模型与前向分步算法并以决策树作为基函数的提升方法。

**前向分步算法**：因为学习的是加法模型，从前向后，每一步只学习一个基函数及其系数，逐步去逼近上述的目标函数式，可简化优化的复杂度。每一步只需优化如下损失函数：



使用均方误差时，残差即为负梯度，基于残差的负梯度对异常值过于敏感

**GBDT和随机森林**

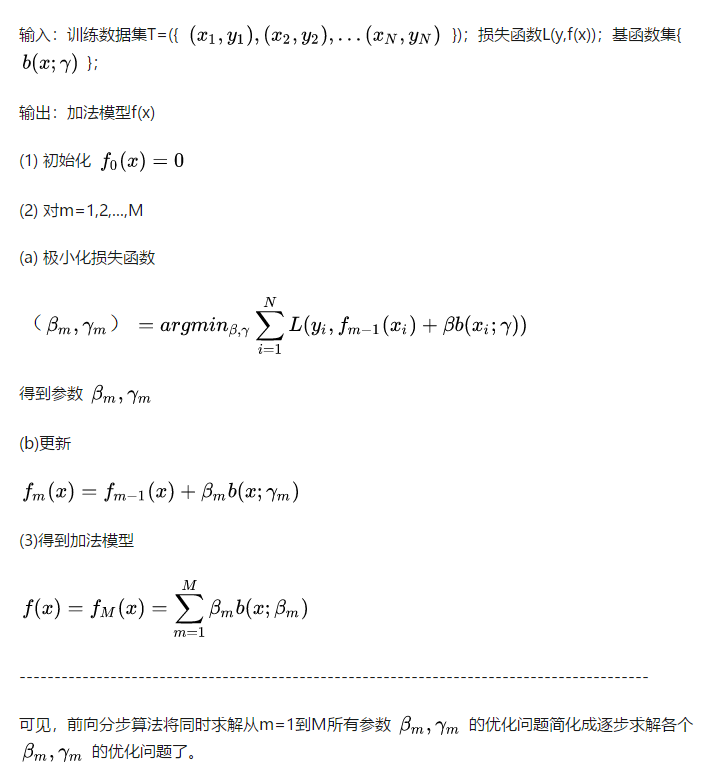
**相同点：**

1. 都是由多棵树组成
2. 最终的结果都是由多棵树一起决定

**不同点：**

1. 组成随机森林的树可以分类树也可以是回归树，而GBDT只由回归树组成
2. 组成随机森林的树可以并行生成，而GBDT是串行生成
3. 随机森林的结果是多数表决表决的，而GBDT则是多棵树累加之和
4. 随机森林对异常值不敏感，而GBDT对异常值比较敏感
5. 随机森林是通过减少模型的方差来提高性能，而GBDT是减少模型的偏差来提高性能的
6. 随机森林不需要进行数据预处理，即特征归一化。而GBDT则需要进行特征归一化

**GBDT算法流程**

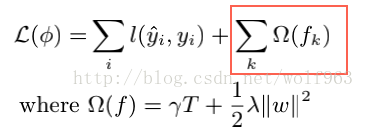


**二阶泰勒展开**



**GBDT和xgboost的区别**

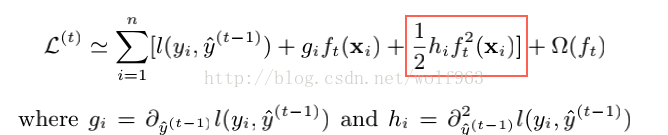
1. XGB加了正则项，普通GBDT没有。为了防止过拟合



T为叶子节点的数量，W为叶子的权重。

Y hat 为预测值，Y为目标值。gamma ,delta 为参数

1. GBDT 是一阶泰勒展开，xgboost损失函数是误差部分是二阶泰勒展开。因此损失函数近似的更精准。



1. shrinkage，对每颗子树增加一个参数，类似于学习率，使得每颗子树的权重降低，防止过拟合。xgboost在进行完一次迭代后，会将叶子节点的权重乘上该系数，主要是为了削弱每棵树的影响，让后面有更大的学习空间。
2. 列抽样，对特征进行降采样，灵感来源于随机森林，除了能降低计算量外，还能防止过拟合。
3. 对缺失值的处理。对于特征的值有缺失的样本，xgboost可以自动学习出它的分裂方向。
4. xgboost工具支持并行，特征并行。这是xgboost比一般GBDT更快的一个重要原因
5. xgboost在训练之前，预先对数据进行了排序，然后保存为block结构，使得在寻找最佳分裂点的时候能够并行化计算。
6. GBDT使用CART作为基分类器，xgboost还支持线性分类器

**lightGBM与xgboost**

基本原理与XGBoost一样，只是在框架上做了一优化

区别：

1. xgboost采用的是level-wise的分裂策略，而lightGBM采用了leaf-wise的策略，区别是xgboost对每一层所有节点做无差别分裂，可能有些节点的增益非常小，对结果影响不大，但是xgboost也进行了分裂，带来了务必要的开销。 leaft-wise的做法是在当前所有叶子节点中选择分裂收益最大的节点进行分裂，如此递归进行，很明显leaf-wise这种做法容易过拟合，因为容易陷入比较高的深度中，因此需要对最大深度做限制，从而避免过拟合。
2. lightgbm使用了基于直方图的决策树算法，直方图算法在内存和计算代价上都有不小优势。

（1）内存上优势：对特征分桶后只需保存特征离散化之后的值，而xgboost既要保存原始feature的值，也要保存这个值的顺序索引。

（2）计算上的优势，预排序算法在选择好分裂特征计算分裂收益时需要遍历所有样本的特征值，,而直方图算法只需要遍历桶就行了

3. 直方图做差加速，一个子节点的直方图可以通过父节点的直方图减去兄弟节点的直方图得到，从而加速计算。

4. 多线程优化，数据并行和特征并行

**怎么理解偏差方差的平衡的？bias variance tradeoff**

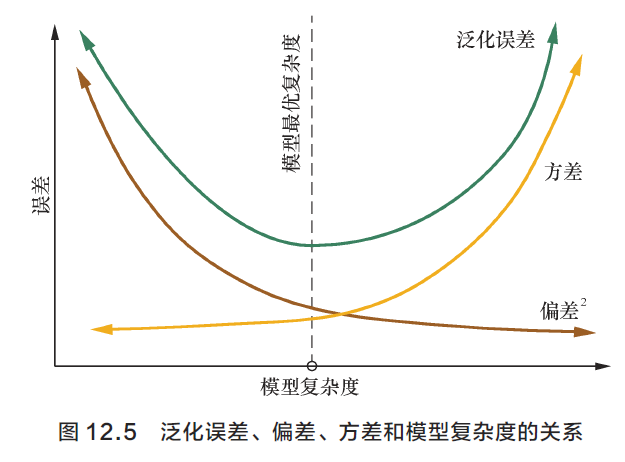
任何模型出现的误差可以分为偏差、方差、噪声之和

**偏差**是指预测结果与真实值之间的差异，排除噪声的影响，偏差是模型无法准确表达数据关系导致，比如模型过于简单，非线性的数据关系采用线性模型建模，偏差较大的模型是错的模型；

**方差**不是针对某一个模型输出样本进行判定，而是指多个模型输出的结果之间的离散差异。方差是由训练集的数据不够导致，一方面数据量不够，有限的数据集过度训练导致模型复杂，另一方面样本质量不行，测试集中的数据分布未在训练集中，导致每次抽样训练模型时，模型参数不同，输出的结果都无法准确预测出正确结果；

**残差**是指预测结果与真实值之间的差异，与偏差的区别是偏差是由于拟合度不够导致，而残差是模型准确，但仍然与真实值有一定的差异，这里可以理解成噪声，噪声是随机的，意味着不可预测，而偏差不是随机产生的，可通过一定的特征工程进行预测；

高偏差误差意味着我们的模型表现不太好，因为没有抓到重要的趋势。另一方面，方差量化了在同一个观察上进行的预测是如何彼此不同的。高方差模型会过度拟合你的训练集，而在训练集以外的数据上表现很差。

**从方差、偏差角度解释 Boosting 和 Bagging**

Bagging能够提高弱分类器性能的原因是降低了方差，Boosting能够提升弱分类器性能的原因是降低了偏差。

1. Bagging是再抽样，然后在每个样本上训练出来的模型取平均。

对n 个独立不相关的模型预测结果取平均，方差是原来单个模型的1/n。

在随机森林算法中，每次选取节点分裂属性时，会随机抽取一个属性子集，而不是从所有属性中选取最优属性，这就是为了避免弱分类器之间过强的相关性。通过训练集的重采样也能够带来弱分类器之间的一定独立性，从而降低Bagging 后模型的方差。

1. Boosting先训练好一个弱分类器，计算弱分类器的错误或者残差，作为下一个分类器的输入。这个过程本身就是在不断减小损失函数，使得模型偏差不断降低。但Boosting 的过程并不会显著降低方差。这是因为Boosting 的训练过程使得各弱分类器之间是强相关的，缺乏独立性，所以并不会对降低方差有作用。

方差和偏差是相辅相成，矛盾又统一的，二者并不能完全独立的存在。

如果模型复杂度过低，虽然方差很小，但是偏差会很高

如果模型复杂度过高，虽然偏差降低了，但是方差会很高。

所以需要综合考虑偏差和方差选择合适复杂度的模型进行训练。

**梯度下降法和牛顿法的区别与对比**

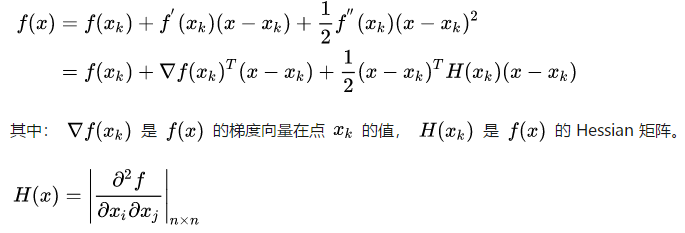
1. 梯度下降法是一阶优化算法，牛顿法是二阶优化算法
2. 牛顿法的收敛速度相比梯度下降法常常较快
3. 牛顿法每次需要更新一个二维矩阵，计算代价很大，实际使用中常使用拟牛顿法
4. 牛顿法对初始值有一定要求，在非凸优化问题中（如神经网络训练），牛顿法很容易陷入鞍点（牛顿法步长会越来越小），而梯度下降法则很容易逃离鞍点（因此在神经网络训练中一般使用梯度下降法，高维空间的神经网络中存在大量鞍点）
5. 梯度下降法在靠近最优点时会震荡，因此步长调整在梯度下降法中是必要的，具体有adagrad, adadelta, rmsprop, adam等一系列自适应学习率的方法

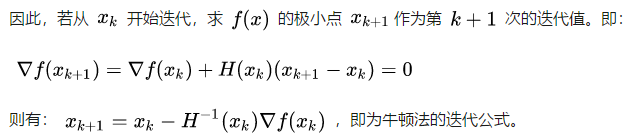
**牛顿法和拟牛顿法**

牛顿法是迭代算法，每一步需要求解目标函数的 Hessian 矩阵的逆矩阵，计算过程复杂。

拟牛顿法通过正定矩阵近似 Hessian 矩阵的逆矩阵或 Hessian 矩阵，简化计算过程。

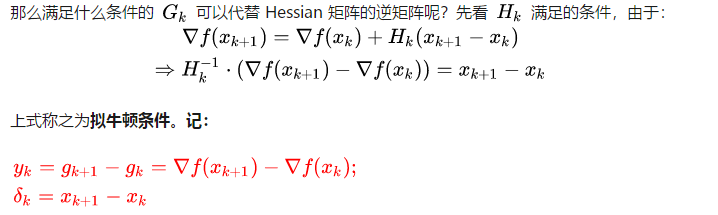
**牛顿法**

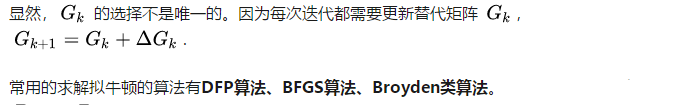




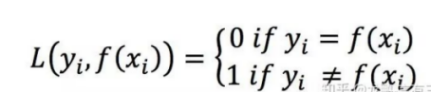
**拟牛顿法**

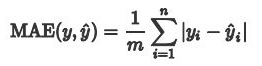




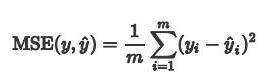


**损失函数**

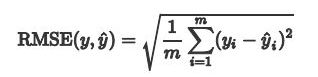
0-1 loss 

平均绝对误差 

绝对值的存在导致函数不光滑，在某些点上不能求导，可以考虑将绝对值改为残差的平方，这就是均方误差。

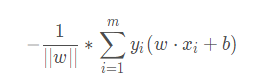
均方误差 

MSE与我们的目标变量的量纲不一致，为了保证量纲一致性，我们需要对MSE进行开方 。

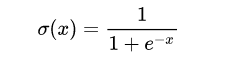
均方根误差 

分类损失

Hinge Loss IMG_256

交叉熵 Cross Entropy IMG_256

感知机的损失函数：误分类点到超平面的总距离

**激活函数**

sigmoid

输入实数值并将其“挤压”到0到1范围内，适合输出为概率的情况

缺陷：

**Sigmoid**函数饱和使梯度消失。当神经元的激活在接近0或1处时会饱和，在这些区域梯度几乎为0，这就会导致梯度消失，几乎就有没有信号通过神经传回上一层。

Sigmoid函数的输出不是零中心的。因为如果输入神经元的数据总是正数，那么关于IMG_256的梯度在反向传播的过程中，将会要么全部是正数，要么全部是负数，这将会导致梯度下降权重更新时出现z字型的下降。

**tanh**

将实数值压缩到[-1,1]之间

缺陷：

Tanh解决了Sigmoid的输出是不是零中心的问题，但仍然存在饱和问题。

为了防止落入饱和区，现在主流的做法会在激活函数前多做一步batch normalization，尽可能保证每一层网络的输入具有均值较小的、零中心的分布。

**ReLU**

优势：

因为不会落入饱和区，对于随机梯度下降的收敛有巨大的加速作用

sigmoid和tanh在求导时含有指数运算，而ReLU求导几乎不存在任何计算量。

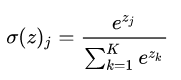
缺陷：由于负数部分恒为0，会导致一些神经元无法激活

单侧抑制，ReLU单元比较脆弱并且可能“死掉”，而且是不可逆的，因此导致了数据多样化的丢失。

**Leaky ReLU**

ε是很小的负数梯度值

使负轴信息不会全部丢失，解决了ReLU神经元“死掉”的问题。



**Softmax**：用于多分类神经网络输出

Softmax是Sigmoid的扩展，当类别数k＝2时，Softmax回归退化为Logistic回归。

**BP算法**

1) 正向传播FP(求损失).在这个过程中,我们根据输入的样本,给定的初始化权重值W和偏置项的值b, 计算最终输出值以及输出值与实际值之间的损失值.如果损失值不在给定的范围内则进行反向传播的过程; 否则停止W,b的更新。

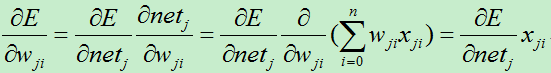
反向传播BP.将输出以某种形式通过隐层向输入层逐层反传,并将误差分摊给各层的所有单元，从而获得各层单元的误差信号,此误差信号即作为修正各单元权值的依据。

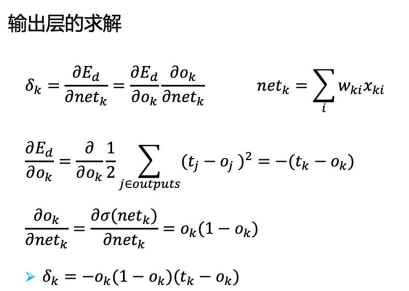
损失函数：

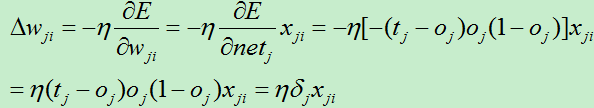


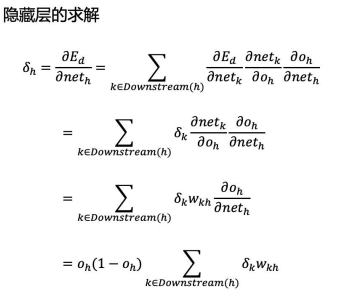
D个样本，k个输出，okd 为输出值，tkd为真实值

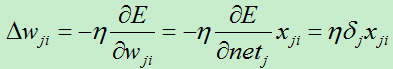
假设损失函数是均方误差，激活函数是sigmoid











缺点：

1. 由于BP神经网络中的参数众多，每次都需要更新数量较多的阈值和权值，故会导致收敛速度过慢。
2. 很容易陷入局部最小值的问题。
3. 当网络的训练次数过多时，可能会出现过拟合的情况。

**梯度消失和梯度爆炸的根本原因**

前层上的梯度的计算来自于后层上梯度的乘积（链式法则）。当层数很多时，就容易出现不稳定。

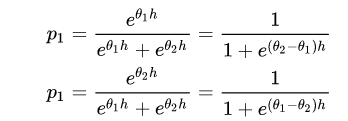
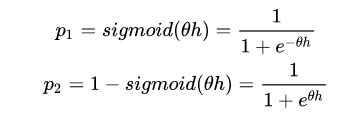
如果激活函数求导后与权重相乘的积大于1，那么层数增多的时候，最终的求出的梯度更新信息将以指数形式增加，即发生梯度爆炸，如果此部分小于1，那么随着层数增多，求出的梯度更新信息将会以指数形式衰减，即发生了梯度消失。

**softmax和交叉熵**

二分类问题中，将得到的数值输入sigmoid，得到某个类别的概率

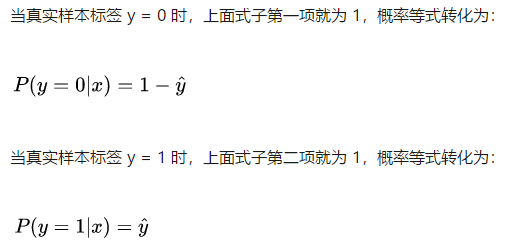
多分类问题中，将得到的数值输入softmax，得到某个类别的概率

**sigmoid是softmax的特殊形式，softmax是sigmoid的推广**



**sigmoid**



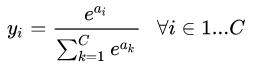


**二分类损失函数**



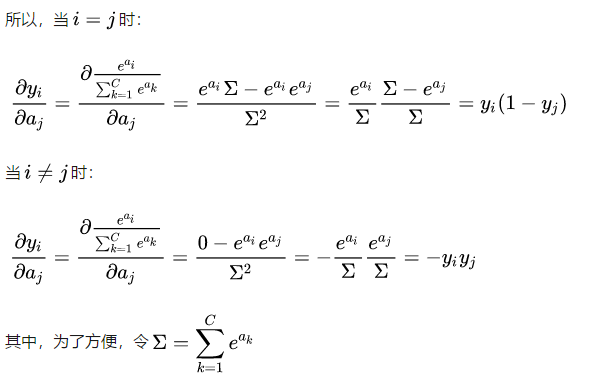


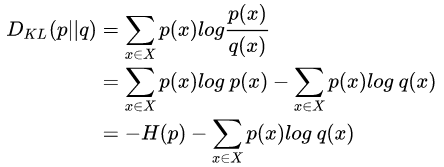
**多分类softmax**



**softmax求导** https://blog.csdn.net/yangwohenmai1/article/details/96741328

第i项输出对第j项输入的偏导

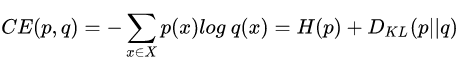


相对熵又称KL散度，两个概率分布P和Q差别的非对称性的度量

**交叉熵**

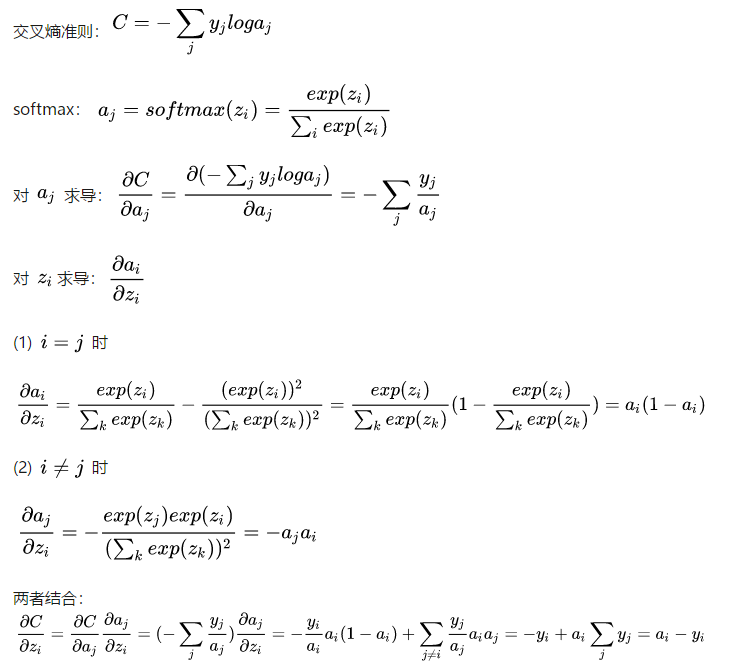
交叉熵主要用于度量两个概率分布间的差异性信息

交叉熵用来判定实际的输出与期望的输出的接近程度，交叉熵越小说明两者之间越接近。



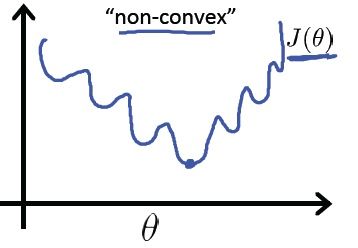
真实标签分布和模型预测标签分布的关系可以用交叉熵表示，形式与对数似然一样

损失函数求导（yi 是标签值，0或1 ，ai 是线性激活后得到的值）



**分类问题为什么选择交叉熵二不使用均方差**

如果用 MSE 计算 loss， 输出的曲线是波动的，有很多局部的极值点。 即，非凸优化问题 (non-convex)



cross entropy 计算 loss，则依旧是一个凸优化问题，用梯度下降求解时，凸优化问题有很好的收敛特性。

分类问题，都用 one-hot + cross entropy

training 过程中，分类问题用 cross entropy，回归问题用 mean squared error。

**为什么均方误差不适合分类问题？**

当sigmoid函数和MSE一起使用时会出现梯度消失。

1. MSE对参数的偏导

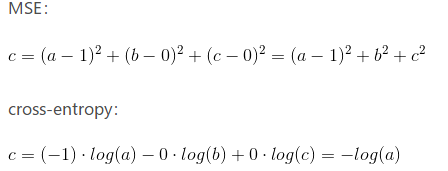
1. 交叉熵对参数的偏导

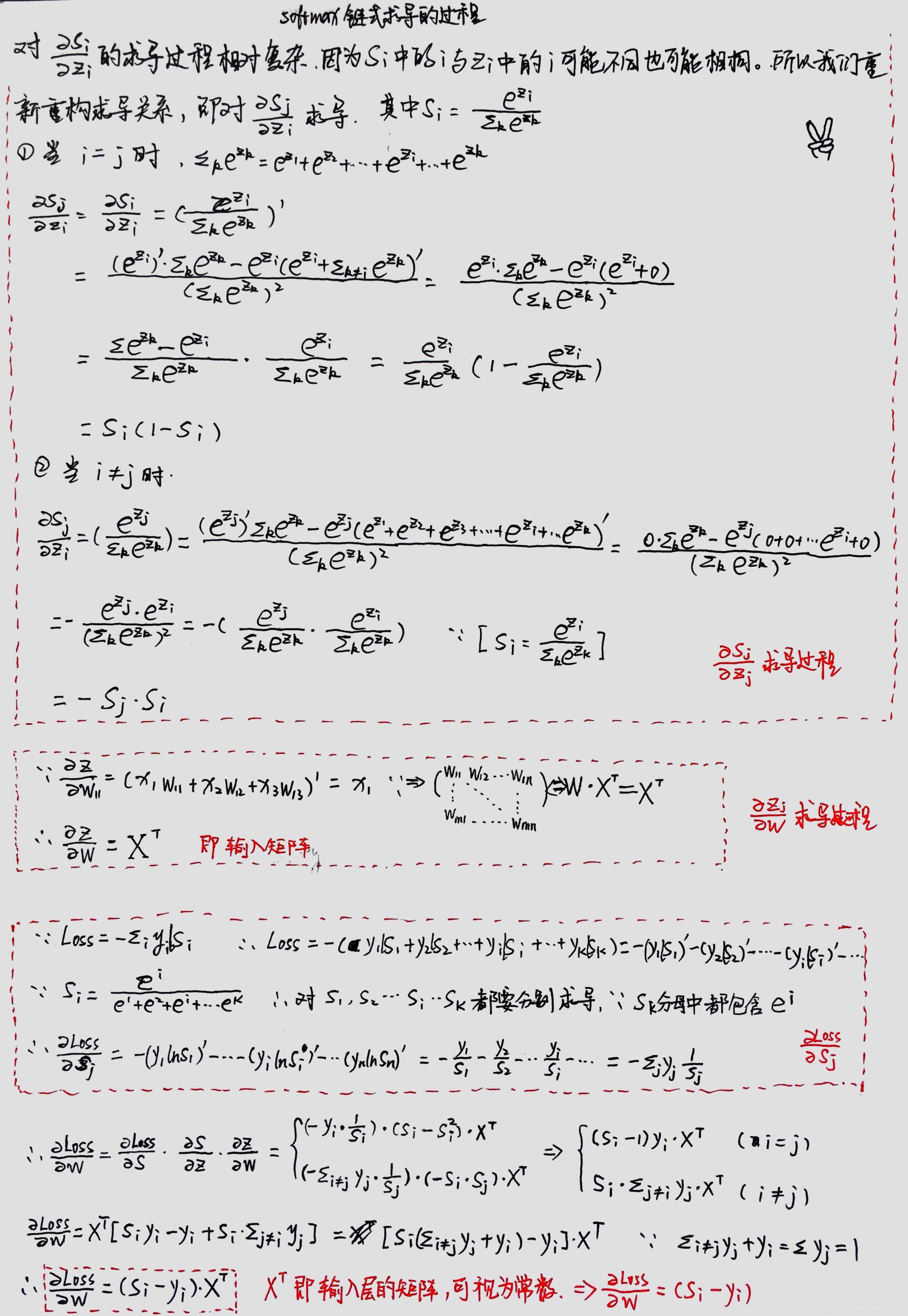
在使用MSE时，w、b的梯度均与sigmoid函数对z的偏导有关系，而sigmoid函数的偏导在自变量非常大或者非常小时，偏导数的值接近于零，这将导致w、b的梯度将不会变化，也就是出现所谓的梯度消失现象。而使用cross-entropy时，w、b的梯度就不会出现上述的情况。所以MSE不适用于分类问题。

**为什么交叉熵不适合回归问题？**

在一个三分类模型中，模型的输出结果为（a,b,c)，而真实的输出结果为(1,0,0)，那么MSE与cross-entropy相对应的损失函数的值如下：



**交叉熵的损失函数只和分类正确的预测结果有关系，而MSE的损失函数还和错误的分类有关系，该分类函数除了让正确的分类尽量变大，还会让错误的分类变得平均**，但实际在分类问题中这个调整是没有必要的。但是对于回归问题来说，这样的考虑就显得很重要了。所以，回归问题熵使用交叉上并不合适。



**CNN**

神经网络每两层之间的所有结点都是有边相连的，称为全连接层网络结构。

对于卷积神经网络，相邻两层之间只有部分节点相连，卷积神经网络的输入输出以及训练流程与全连接神经网络也基本一致。

一个卷积神经网络主要由以下5种结构组成：

1. 输入层。输入层是整个神经网络的输入，在处理图像的卷积神经网络中，它一般代表了一张图片的像素矩阵，处理文本时，就是词向量矩阵。
2. 卷积层。卷积层中每一个节点的输入只是上一层神经网络的一小块，称为卷积核（kernel）。这个小块常用的大小有3×3或者5×5。一般来说，通过卷积层处理过的节点矩阵会变得更深。
3. 池化层（Pooling）。池化层神经网络不会改变三维矩阵的深度，但是它可以缩小矩阵的大小。可以进一步缩小最后全连接层中节点的个数，从而达到减少整个神经网络中参数的目的。
4. 全连接层。在特征提取完成之后，仍然需要使用全连接层来完成分类任务。
5. Softmax层。通过Softmax层，可以得到当前样例属于不同种类的概率分布情况。

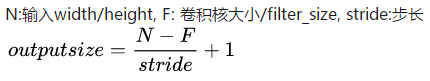
在卷积神经网络中，每一个卷积层中使用的过滤器中的参数都是一样的。共享每一个卷积层中过滤器中的参数可以巨幅减少神经网络上的参数。

**池化层可以非常有效地缩小矩阵的尺寸**，**从而减少最后全连接层中的参数**。使用池化层既可以加快计算速度也有防止过拟合问题的作用。

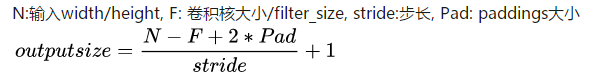
池化层前向传播的过程也是通过移动一个类似过滤器的结构完成的。不过池化层过滤器中的计算不是节点的加权和，而是采用更加简单的最大值或者平均值运算。最大池化层（max pooling），平均池化层（average pooling）。

池化层的过滤器也需要人工设定过滤器的尺寸、是否使用全0填充以及过滤器移动的步长等设置，而且这些设置的意义也是一样的。

**不考虑padding**



**考虑padding**



# **1\*1的卷积核和全连接层有什么异同？**

当输入的feature map的尺寸是1×1时，两者从数学原理上来看，没有区别。假设输入为c×1×1，输出为n×1×1，那么全连接可以认为是一个c维的向量和n×c大小的矩阵相乘。卷积层可以理解为n个c×1×1的卷积核，分别与输入做内积，跟计算矩阵向量乘没有区别。

当输入为c×w×h时，卷积层和全连接层的输出尺寸就不一样了，1×1的卷积输出为n×w×h，全连接的输出是n×1×1。此时，全连接可以等价于n个c×w×h卷积核的卷积层。

全连接层和卷积层最大的区别就是输入尺寸是否可变，全连接层的输入尺寸是固定的，卷积层的输入尺寸是任意的。

**RNN**

循环神经网络的主要用途是处理和预测序列数据。全连接神经网络或卷积神经网络中，层与层之间是全连接或部分连接的，每层之间的节点是无连接的。

RNN的隐藏层之间的节点是有连接的，隐藏层的输入不仅包括输入层的输出，还包括上一时刻隐藏层的输出。

正如卷积神经网络在不同的空间位置共享参数，循环神经网络是在不同时间位置共享参数，从而能够使用有限的参数处理任意长度的序列。某些情况下存在 ot=ht 的特例

展开之后可以看成是有N个中间层的前馈神经网络，训练方法 BPTT（沿时间反向传播），以tanh 函数作为激活函数，输出序列接一个全连接层。

RNN在处理long term memory的时候存在缺陷，因此LSTM应运而生。

LSTM的细胞状态会决定哪些状态应该被留下来，哪些状态应该被遗忘。

**LSTM 长短时记忆网络**

与一般RNN 不同的是，LSTM 有三个“门”机构，输入门，遗忘门和输出门。“门”的结构就是一个使用sigmoid 神经网络和一个按位做乘法的操作，得到一个0到1之间的数值，描述当前输入有多少信息量可以通过这个结构。【1：全部信息通过，0：全部不通过】

“遗忘门”会根据当前的输入Xt和上一时刻输出 ht-1 决定哪一部分记忆需要被遗忘。在循环神经网络“忘记”了部分之前的状态后，它还需要从当前的输入补充最新的记忆。这个过程就是“输入门”完成的。“输出门”会根据最新的状态Ct 、上一时刻的输出ht-1 和当前的输入Xt 来决定该时刻的输出ht。

具体LSTM每个“门”的公式定义如下：

z = tanh(Wz[ht-1 , xt]) （输入值）

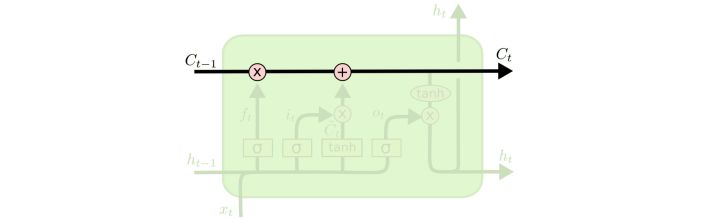
i = sigmoid(Wt[ht-1 , xt]) （输入门）

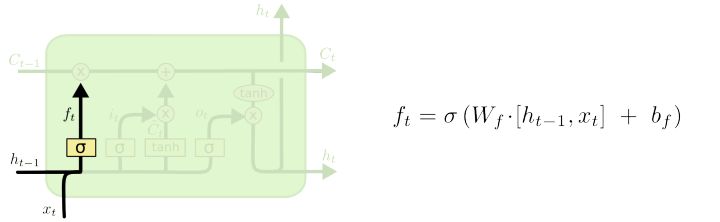
f =sigmoid(Wf［ht-1 , xt]) （遗忘门）

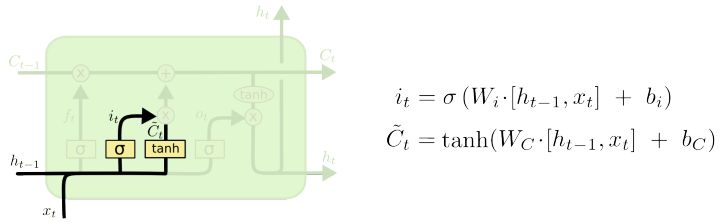
o =sigmoid（Wo［ht-1 , xt]) （输出门）

Ct = f · Ct-1 + i · z （新状态）

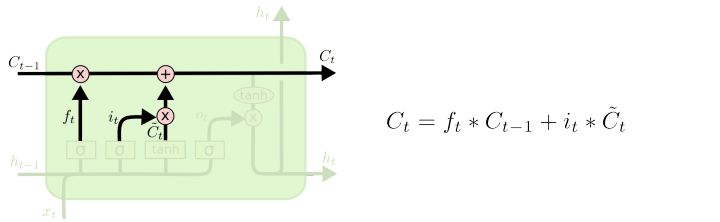
ht= o · tanh ct （输出）

LSTM的记忆干流就是Ct，上一个单元的输入作用于它，并且与它一起形成这个单元的输出：

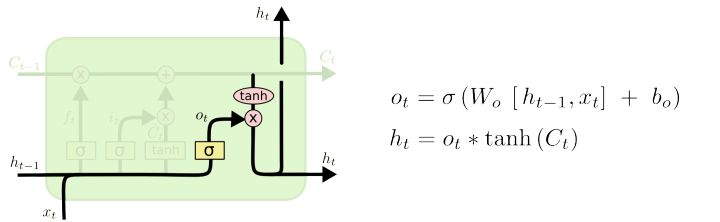
LSTM则包含了一个叫遗忘门的单元，用来判断上一单元传过来的哪些是可以忘记的，哪些需要记下

对本单元的输入信息进行提取

调整当前的记忆



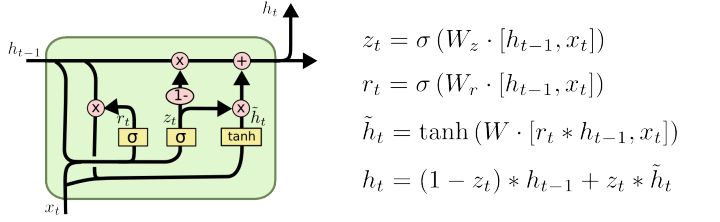
完成本单元的输出



GRU

GRU是LSTM的一种简化，它组合了遗忘门和输入门到一个单独的“更新门”中。它也合并了cell state和hidden state。只有两个门，reset gate IMG_256 和update gate IMG_257。

一般来说那些具有短距离依赖的单元reset gate比较活跃（如果 IMG_256rt 为1，而 ztIMG_257 为0 那么相当于变成了一个标准的RNN，能处理短距离依赖），具有长距离依赖的单元update gate比较活跃。



**一方面GRU的参数更少，因而训练稍快或需要更少的数据来泛化。另一方面，如果有足够的数据，LSTM的强大表达能力可能会产生更好的结果。**

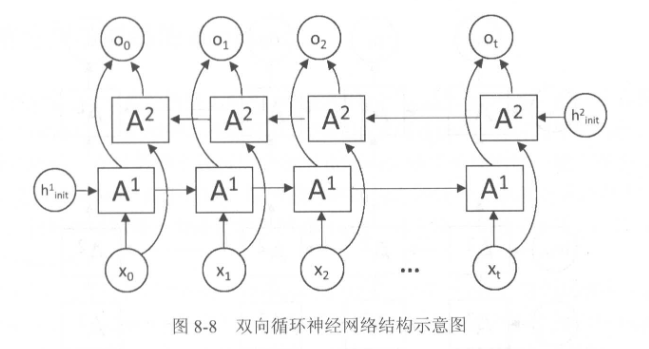
BiRNN

双向循环神经网络的主体结构就是两个单向循环神经网络的结合。在每一个时刻t，输入会同时提供给这两个方向相反的循环神经网络。两个网络独立进行计算，各自产生该时刻的新状态和输出，而双向循环网络的最终输出是这两个单向循环神经网络的输出的简单拼接。两个循环神经网络除方向不同以外，其余结构完全对称。

简单的RNN、LSTM均可以作为双向循环网络的循环体。

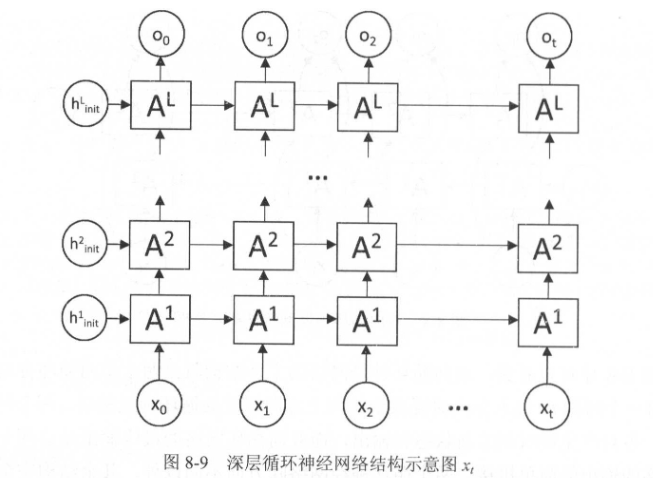
tensorflow 中，先定义好cell和Dropout Wrapper之后，使用 bidirectional\_dynamic\_rnn 实现

dynamic\_mn 接受两个参数，数据内容和序列长度，会根据每个 batch 的最大长度动态展开到需要的层数



深层循环神经网络（DeepRNN）是循环神经网络的另外一种变种，每层循环网络的输出传给下一层进行处理，每一层的循环体中参数是一致的，而不同层中的参数可以不同。

TensorFlow中先定义好cell 和Dropout Wrapper之后，提供了MultiRNNCell来实现



**Dropout**

在机器学习的模型中，如果模型的参数太多，而训练样本又太少，训练出来的模型很容易产生过拟合的现象。在训练神经网络的时候经常会遇到过拟合的问题，过拟合具体表现在：模型在训练数据上损失函数较小，预测准确率较高；但是在测试数据上损失函数比较大，预测准确率较低。

过拟合是很多机器学习的通病。如果模型过拟合，那么得到的模型几乎不能用。为了解决过拟合问题，一般会采用模型集成的方法，即训练多个模型进行组合。此时，训练模型费时就成为一个很大的问题，不仅训练多个模型费时，测试多个模型也是很费时。

综上所述，训练深度神经网络的时候，总是会遇到两大缺点：

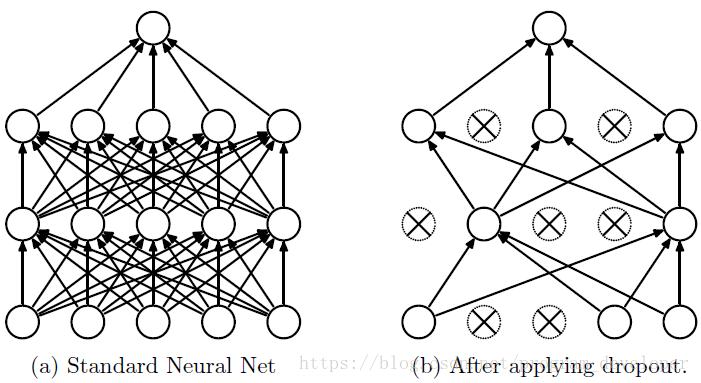
（1）容易过拟合

（2）费时

Dropout可以比较有效的缓解过拟合的发生，在一定程度上达到正则化的效果。

在每个训练批次中，通过忽略一半的特征检测器（让一半的隐层节点值为0），可以明显地减少过拟合现象。这种方式可以减少特征检测器（隐层节点）间的相互作用，检测器相互作用是指某些检测器依赖其他检测器才能发挥作用。

Dropout说的简单一点就是：我们在前向传播的时候，让某个神经元的激活值以一定的概率p停止工作，这样可以使模型泛化性更强，因为它不会太依赖某些局部的特征，如图所示。



使用Dropout之后，过程变成如下：

（1）首先随机（临时）删掉网络中一半的隐藏神经元，输入输出神经元保持不变

（2） 然后把输入x通过修改后的网络前向传播，然后把得到的损失结果通过修改的网络反向传播。一小批训练样本执行完这个过程后，在没有被删除的神经元上按照随机梯度下降法更新对应的参数（w，b）。

（3）然后继续重复这一过程：

* 恢复被删掉的神经元（此时被删除的神经元保持原样，而没有被删除的神经元已经有所更新）
* 从隐藏层神经元中随机选择一个一半大小的子集临时删除掉（备份被删除神经元的参数）。
* 对一小批训练样本，先前向传播然后反向传播损失并根据随机梯度下降法更新参数（w，b） （没有被删除的那一部分参数得到更新，删除的神经元参数保持被删除前的结果）。

不断重复这一过程。

**Dropout 在训练和测试时都需要嘛？**

Dropout 在训练时采用，是为了减少神经元对部分上层神经元的依赖，类似将多个不同网络结构的模型集成起来，减少过拟合的风险。

而在测试时，应该用整个训练好的模型，因此不需要dropout。

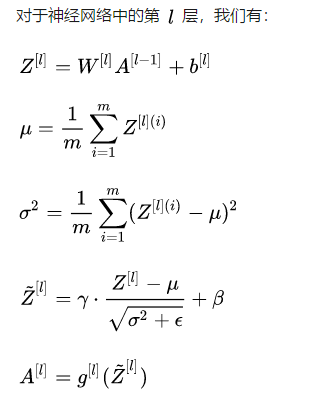
**为什么说Dropout可以解决过拟合？**

**（1）取平均的作用**： 先回到标准的模型即没有dropout，我们用相同的训练数据去训练5个不同的神经网络，一般会得到5个不同的结果，此时我们可以采用 “5个结果取均值”或者“多数取胜的投票策略”去决定最终结果。例如3个网络判断结果为数字9,那么很有可能真正的结果就是数字9，其它两个网络给出了错误结果。这种“综合起来取平均”的策略通常可以有效防止过拟合问题。因为不同的网络可能产生不同的过拟合，取平均则有可能让一些“相反的”拟合互相抵消。**dropout掉不同的隐藏神经元就类似在训练不同的网络**，**随机删掉一半隐藏神经元导致网络结构已经不同，整个dropout过程就相当于对很多个不同的神经网络取平均**。而不同的网络产生不同的过拟合，一些互为“反向”的拟合相互抵消就可以达到整体上减少过拟合。

**（2）减少神经元之间复杂的共适应关系**： **因为dropout程序导致两个神经元不一定每次都在一个dropout网络中出现。这样权值的更新不再依赖于有固定关系的隐含节点的共同作用，阻止了某些特征仅仅在其它特定特征下才有效果的情况 。**迫使网络去学习更加鲁棒的特征 ，这些特征在其它的神经元的随机子集中也存在。换句话说假如我们的神经网络是在做出某种预测，它不应该对一些特定的线索片段太过敏感，即使丢失特定的线索，它也应该可以从众多其它线索中学习一些共同的特征。从这个角度看dropout就有点像L1，L2正则，减少权重使得网络对丢失特定神经元连接的鲁棒性提高。

**（3）Dropout类似于性别在生物进化中的角色**：物种为了生存往往会倾向于适应这种环境，环境突变则会导致物种难以做出及时反应，性别的出现可以繁衍出适应新环境的变种，有效的阻止过拟合，即避免环境改变时物种可能面临的灭绝。

**Batch Normalization**

机器学习领域有个很重要的假设：IID独立同分布假设，就是**假设训练数据和测试数据是满足相同分布的**，这是通过训练数据获得的模型能够在测试集获得好的效果的一个基本保障。那**Batch Norm**的作用是什么呢？**Batch Norm**就是在深度神经网络训练过程中使得每一层神经网络的输入保持相同分布的。

Internal Covariate Shift定义：**在深层网络训练的过程中，由于网络中参数变化而引起内部结点数据分布发生变化。**

**Internal Covariate Shift会带来什么问题？**

1. 上层网络需要不停调整来适应输入数据分布的变化，导致网络学习速度的降低
2. 下层输入的变化可能趋向于变大或者变小，导致上层落入饱和区，使得学习过早停止。
3. 每层的更新都会影响到其它层，因此每层的参数更新策略需要尽可能的谨慎

**Batch Normalization 位置？**

BN在非线性激活函数之前。

Internal Covariate Shift 的产生是由于线性变换将输入数据的分布改变了，产生的结果是使梯度落在激活函数的饱和区，导致梯度更新变慢。BN将分布修正之后，再交给激活函数处理。所以在激活函数之前。

**BN就是通过一定的规范化手段，把每层神经网络任意神经元这个输入值的分布强行拉回到均值为0方差为1的标准正态分布**，其实就是把越来越偏的分布强制拉回比较标准的分布，这样**使得激活输入值落在非线性函数对输入比较敏感的区域**，**这样输入的小变化就会导致损失函数较大的变化，意思是这样让梯度变大，避免梯度消失问题产生，而且梯度变大意味着学习收敛速度快，能大大加快训练速度。**

**BN为了保证非线性的获得，对变换后的满足均值为0方差为1的x又进行了scale加上shift操作(y=scale\*x+shift)**，每个神经元增加了两个参数scale和shift参数，这两个参数是通过训练学习到的。由于归一化后的xi基本会被限制在正态分布下，使得网络的表达能力下降。**核心思想应该是想找到一个线性和非线性的较好平衡点，既能享受非线性的较强表达能力的好处，又避免太靠非线性区两头使得网络收敛速度太慢。**

**输入就只有一个实例，看不到Mini-Batch其它实例，那么这时候怎么对输入做BN呢？**

每次做Mini-Batch训练时，都会有那个Mini-Batch里m个训练实例获得的均值和方差，保存每个Mini-Batch的均值和方差，然后对这些均值和方差求其对应的数学期望即可得出全局统计量。

**Batch Normalization的优势**

1. **BN使得网络中每层输入数据的分布相对稳定，加速模型学习速度**

BN通过规范化与线性变换使得每一层网络的输入数据的均值与方差都在一定范围内，使得后一层网络不必不断去适应底层网络中输入的变化，从而实现了网络中层与层之间的解耦，允许每一层进行独立学习，有利于提高整个神经网络的学习速度。

1. **BN使得模型对网络中的参数不那么敏感，简化调参过程，使得网络学习更加稳定**

在使用Batch Normalization之后，抑制了参数微小变化随着网络层数加深被放大的 问题，使得网络对参数大小的适应能力更强，此时我们可以设置较大的学习率而不用过于担心模型divergence的风险。

1. **BN允许网络使用饱和性激活函数（sigmoid，tanh等），缓解梯度消失问题**

通过normalize操作可以让激活函数的输入数据落在梯度非饱和区，缓解梯度消失的 问题；另外通过自适应学习参数β、γ又让数据保留更多的原始信息。

1. **BN具有一定的正则化效果**

不同mini-batch的均值与方差会有所不同，这就为网络的学习过程中增加了随机噪音，与Dropout通过关闭神经元给网络训练带来噪音类似，在一定程度上对模型起到了**正则化的效果。**

**BatchNorm**目前基本已经成为各种网络（RNN除外）的标配，主要是因为效果好，比如可以加快模型收敛速度，不再依赖精细的参数初始化过程，可以调大学习率等各种方便，同时引入的随机噪声能够起到对模型参数进行正则化的作用，有利于增强模型泛化能力。

**BN的局限性**

1. **如果Batch Size太小，则BN效果明显下降。**
2. **对于有些像素级图片生成任务来说，BN效果不佳；**
3. **RNN等动态网络使用BN效果不佳且使用起来不方便**

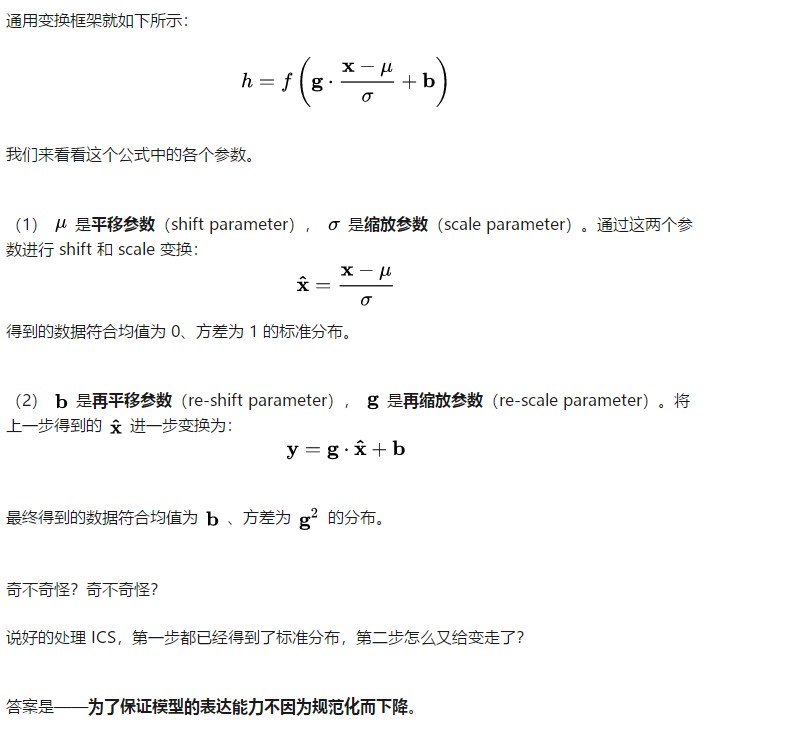
因为输入的Sequence序列是不定长的，这源自同一个Mini-Batch中的训练实例有长有短。对于类似RNN这种动态网络结构，BN使用起来不方便，因为要应用BN，那么RNN的每个时间步需要维护各自的统计量，而Mini-Batch中的训练实例长短不一，这意味着RNN不同时间步的隐层会看到不同数量的输入数据。

1. **训练时和推理时统计量不一致**

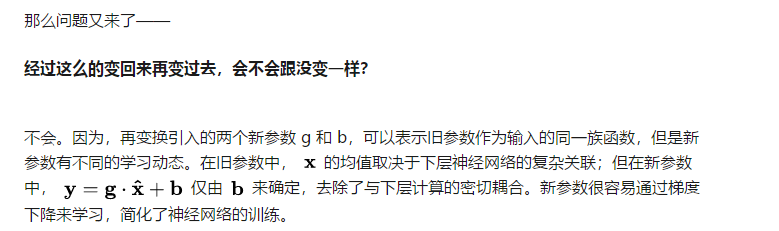
对于BN来说，采用Mini-Batch内实例来计算统计量，这在训练时没有问题，但是在预测的时候，是单实例的，不存在Mini-Batch，所以就无法获得BN计算所需的均值和方差

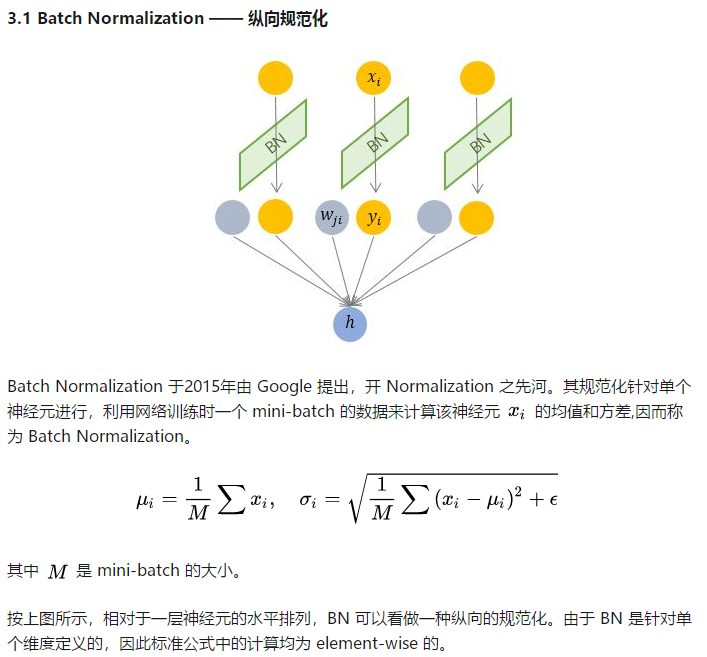
一般解决方法是采用训练时刻记录的各个Mini-Batch的统计量的数学期望，以此来推算全局的均值和方差，在线推理时采用这样推导出的统计量。实际使用并没大问题，但是确实存在训练和预测时统计量计算方法不一致的问题。

**Layer Normalization**这种在同隐层内计算统计量的模式就比较符合RNN这种动态网络，目前在RNN中貌似也只有LayerNorm相对有效，但Layer Normalization目前看好像也只适合应用在RNN场景下，在CNN等环境下效果是不如BatchNorm或者GroupNorm等模型的。



经过第一步操作后，Normalization有可能降低神经网络的非线性表达能力，所以会以此方式来补偿Normalization操作后神经网络的表达能力。

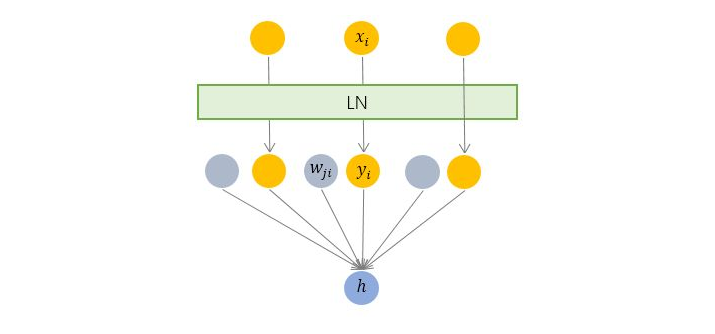


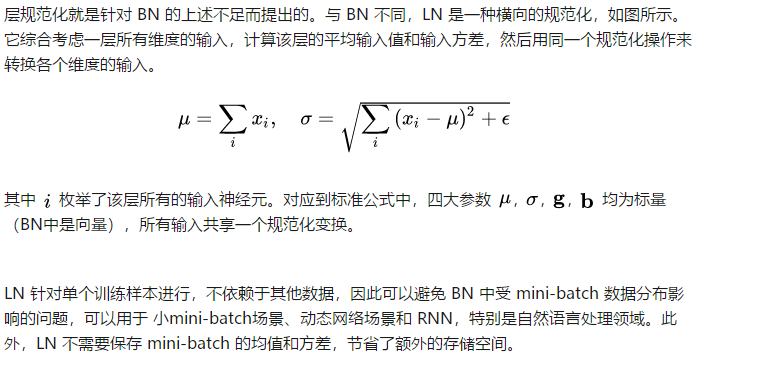


BN 比较适用的场景是：每个 mini-batch 比较大，数据分布比较接近。在进行训练之前，要做好充分的 shuffle. 否则效果会差很多。

目前BN在这些基础网络结构都有尝试，总体而言，BN在MLP和CNN是非常成功的，在RNN上效果不明显。

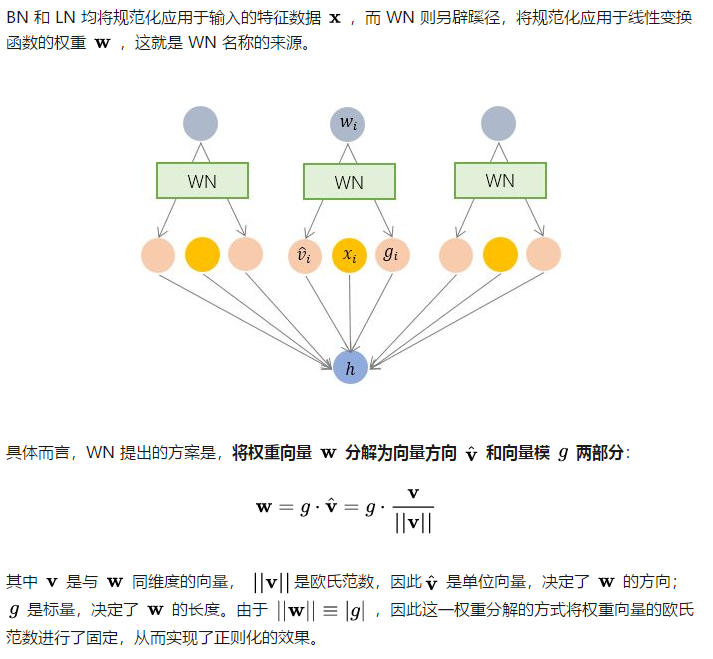
**3.2 Layer Normalization —— 横向规范化**





但是，BN 的转换是针对单个神经元可训练的——不同神经元的输入经过再平移和再缩放后分布在不同的区间，而 LN 对于一整层的神经元训练得到同一个转换——所有的输入都在同一个区间范围内。如果不同输入特征不属于相似的类别（比如颜色和大小），那么 LN 的处理可能会降低模型的表达能力。

**3.3 Weight Normalization —— 参数规范化**



****Batch Normalization为何有效?****

原始的BN论文给出的解释是BN可以解决神经网络训练过程中的ICS（Internal Covariate Shift）问题，所谓ICS问题，指的是由于深度网络由很多隐层构成，在训练过程中由于底层网络参数不断变化，导致上层隐层神经元激活值的分布逐渐发生很大的变化和偏移，而这非常不利于有效稳定地训练神经网络。

但是能够解决ICS问题其实并不是BN为何有效背后真正的原因，最近有一些研究对此作了探讨。那么ICS问题真实存在吗？ICS问题在较深的网络中确实是普遍存在的，但是这并非导致深层网络难以训练的根本原因。另外，BN解决了ICS问题了吗？其实也没有。实验一方面证明：**即使是应用了BN，网络隐层中的输出仍然存在严重的ICS问题**；另一方面也证明了：**在BN层输出后人工加入噪音模拟ICS现象，并不妨碍BN的优秀表现**。这两方面的证据互相佐证来看的话，其实侧面说明了**BN和ICS问题并没什么关系。**

那么BN有效的真正原因到底是什么呢？这还要从深度网络的损失曲面（Loss Surface）说起，在深度网络叠加大量非线性函数方式来解决非凸复杂问题时，损失曲面形态异常复杂，大量空间坑坑洼洼相当不平整，也有很多空间是由平坦的大量充满鞍点的曲面构成，训练过程就是利用SGD在这个复杂平面上一步一步游走，期望找到全局最小值，也就是曲面里最深的那个坑。所以在SGD寻优时，在如此复杂曲面上寻找全局最小值而不是落入局部最小值或者被困在鞍点动弹不得，可想而知难度有多高。

有了损失曲面的基本概念，我们回头来看为何BN是有效的。研究表明，BN真正的用处在于：通过上文所述的Normalization操作，使得网络参数重整（Reparametrize），它对于非线性非凸问题复杂的损失曲面有很好的平滑作用，参数重整后的损失曲面比未重整前的参数损失曲面平滑许多。**BN通过参数重整确实起到了平滑损失曲面及梯度的作用。**

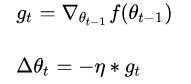
**Normalization通过对激活值进行正态分布化的参数重整，产生参数Re-Scaling不变的效果，因此缓解梯度消失或梯度爆炸问题，与其对应的重整后的损失曲面及梯度也因此变得更平滑，更有利于SGD寻优找到问题好的解决方案。**

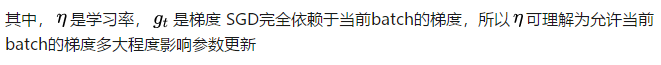
------------------------------------------------------------------------------------------------------------------------------

**优化算法**

SGD mini-batch gradient descent

SGD就是每一次迭代计算mini-batch的梯度，然后对参数进行更新，是最常见的优化方法了。即：



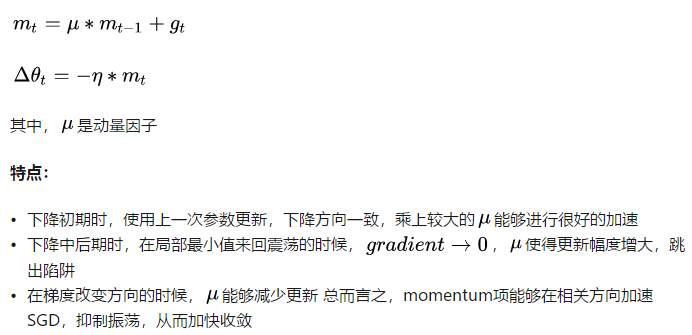


缺点：

1. 选择合适的learning rate比较困难 - 对所有的参数更新使用同样的learning rate。 对于稀疏数据或者特征，有时我们可能想更新快一些对于不经常出现的特征，对于常出 现的特征更新慢一些，这时候SGD就不太能满足要求了
2. SGD容易收敛到局部最优，并且在某些情况下可能被困在鞍点

**Momentum**

momentum是模拟物理里动量的概念，积累之前的动量来替代真正的梯度。



**区别： SGD每次都会在当前位置上沿着负梯度方向更新（下降，沿着正梯度则为上升），并不考虑之前的方向梯度大小等等。而动量（moment）通过引入一个新的变量去积累之前的梯度（通过指数衰减平均得到），得到加速学习过程的目的。**

**最直观的理解就是，若当前的梯度方向与累积的历史梯度方向一致，则当前的梯度会被加强，从而这一步下降的幅度更大。若当前的梯度方向与累积的梯度方向不一致，则会减弱当前下降的梯度幅度。**

**Nesterov (牛顿动量)**

nesterov项在梯度更新时做一个校正，避免前进太快，同时提高灵敏度。Nesterov的改进就是让之前的动量直接影响当前的动量。







momentum项和nesterov项都是为了使梯度更新更加灵活，对不同情况有针对性。但是，人工设置一些学习率总还是有些生硬，以下是几种自适应学习率的方法

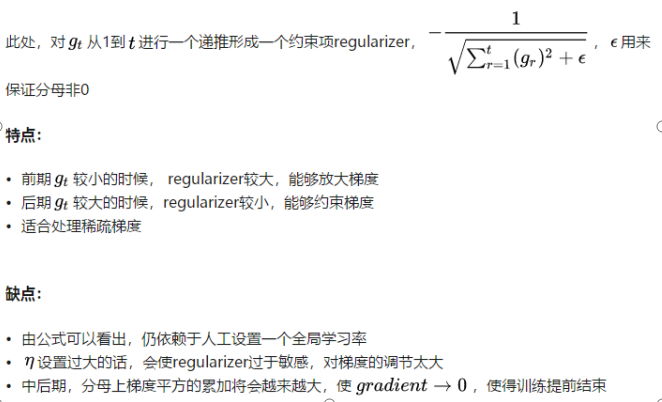
**Adagrad**

对学习率进行了约束





学习率除以根号下每个维度梯度的平方和，使每个维度的学习率不同。平方和是递增的，会导致学习率提前趋向于0。下降速度很快，但是也很快就停止训练了。



****RMSprop****（root mean square prop）

为了防止梯度过早下降为0,使学习不充分，将之前的梯度进行衰减ρ，0<ρ<1。使得历史梯度对学习率的影响递减，但是仍然要选择学习率。

**Adadelta**



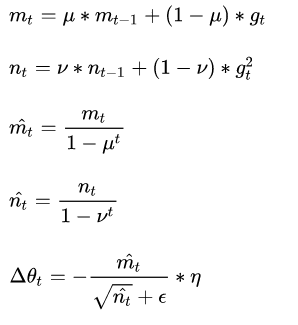
Adadelta对不同的**ε**，得到的结果更稳定，SGD，Momentum，Adagrad对不同的**ε**，结果不稳定。

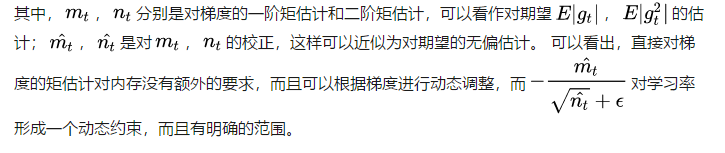
从多个数据集情况来看，AdaDelta在训练初期和中期，具有非常不错的加速效果。

但是到训练后期，进入局部最小值雷区之后，AdaDelta就会反复在局部最小值附近抖动。

**Adam**

Adam(Adaptive Moment Estimation)，它利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率。Adam的优点主要在于经过偏置校正后，每一次迭代学习率都有个确定范围，使得参数比较平稳。公式如下：





**特点：**

**学习率可以从梯度和梯度平方，两个角度自适应的调节，而不是由当前梯度直接决定**

结合了Adagrad善于处理稀疏梯度和RMSprop善于处理非平稳目标的优点

为不同的参数计算不同的自适应学习率

也适用于大多非凸优化 - 适用于大数据集和高维空间

数据稀疏主要指的是数据特征的稀疏性，高维特征中仅有少数非零特征；

模型稀疏主要指的是模型参数的稀疏性，高维参数中仅有少数非零参数。

**概率图模型**

贝叶斯网络是有向的，适合为有单向依赖的数据建模。

马尔科夫网络是无向，适合实体之间互相依赖的建模。

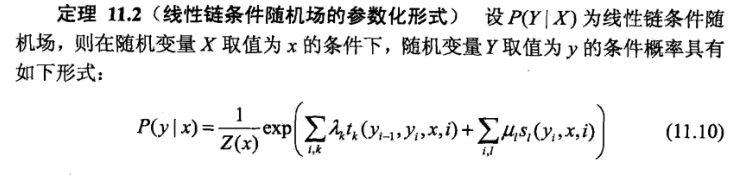
**条件随机场**是**马尔科夫随机场**。隐马尔科夫模型时贝叶斯网络。

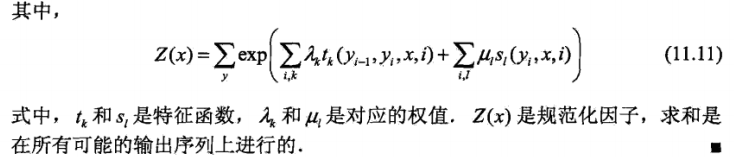
对于标注问题，使用的是线性链条件随机场，这时，问题变成了**由输入序列对输出序列的判别模型**。形式为对数线性模型，学习方法是极大似然估计或正则化的极大似然估计。

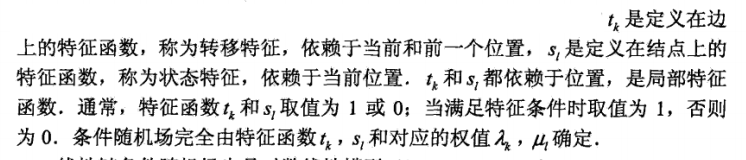
概率无向图模型，又称马尔科夫随机场，是一个可以由无向图表示的联合概率分布。无向图是指边没有方向的图。

无向图中任何两个节点均有边连接的节点子集称为团。不能再加入一个节点，使其称为更大的团，称为最大团。

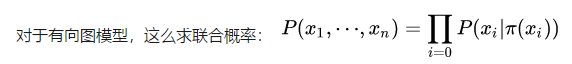
概率无向图模型的联合概率分布，是其最大团的随机变量C的函数的乘积。





表示给定输入序列x ，对输出序列y 预测的条件概率。

对于crf的训练，自然是更新每种特征的权重，使得表现最好，使得P(y|x)最大



**HMM 和 CRF 的区别**

CRF模拟的是条件概率,是判别模型，HMM模拟的是联合概率,是生成模型

CRF是概率无向图，HMM是概率有向图

CRF不对observation之间的关系做假设，HMM要求observation互相独立

CRF用梯度优化求解，HMM用动态规划求解

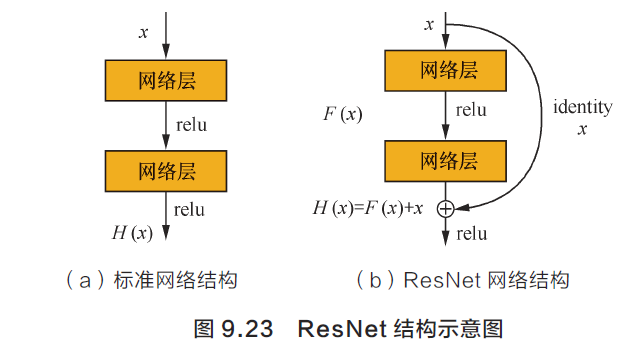
**ResNet的提出背景和核心理论**

ResNet的提出背景是解决或缓解深层的神经网络训练中的梯度消失问题。假设有一个L 层的深度神经网络，如果我们在上面加入一层， 直观来讲得到的L+1 层深度神经网络的效果应该至少不会比L 层的差。因为我们简单地设最后一层为前一层的拷贝（用一个恒等映射即可实现），并且其他层维持原来的参数即可。然而在进行反向传播时，我们很难找到这种形式的解。实际上，通过实验发现，层数更深的神经网络反而会具有更大的训练误差。这很大程度上归结于深度神经网络的梯度消失问题。

ResNet过调整网络结构来解决上述问题。

ResNet 把网络结构调整为，既然离输入近的神经网络层较难训练，那么我们可以将它短接到更靠近输出的层。输入x经过两个神经网络的变换得到F(x)，同时也短接到两层之后，最后这个包含两层的神经网络模块输出H(x)=F(x)+x。

这样一来，F(x) 被设计为只需要拟合输入x 与目标输出 IMG_264 的残差IMG_265，残差网络的名称也因此而来。如果某一层的输出已经较好的拟合了期望结果，那么多加入一层不会使得模型变得更差，因为该层的输出将直接被短接到两层之后，相当于直接学习了一个恒等映射，而跳过的两层只需要拟合上层输出和目标之间的残差即可。

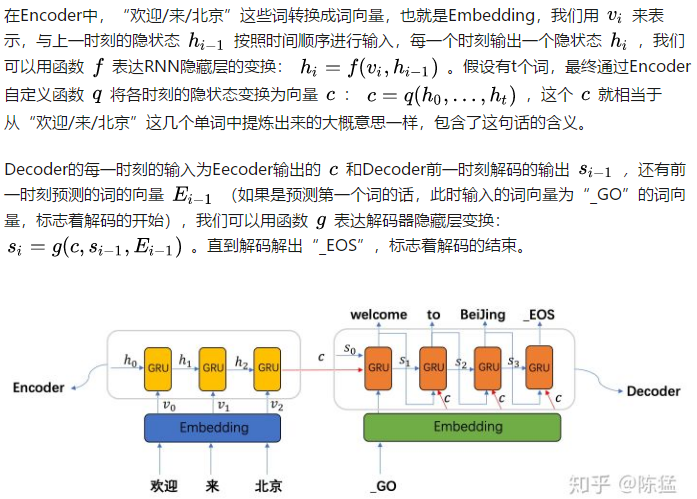


ResNet 可以有效改善深层的神经网络学习问题，使得训练更深的网络成为可能，如图9.24 所示。图9.24（a）展示的是传统神经网络的结果，可以看到随着模型结构的加深训练误差反而上升；而图9.24（b） 是ResNet 的实验结果，随着模型结构的加深，训练误差逐渐降低，并且优于相同层数的传统的神经网络。

**seq2seq**

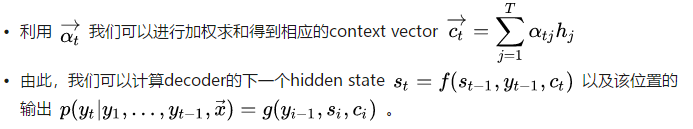
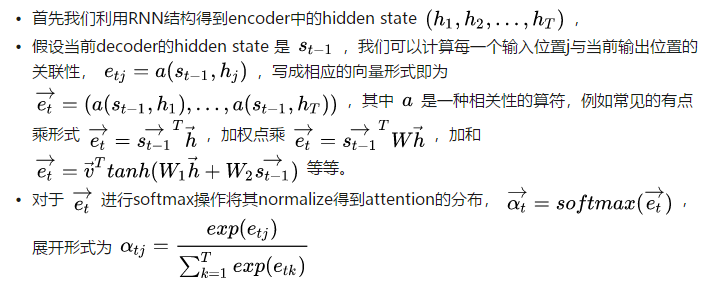
目前Seq2Seq模型在机器翻译，语音识别，文本摘要，问答系统等领域取得了巨大的成功。

Encoder和Decoder一般都是RNN，通常为LSTM或者GRU



**Attention**

让Encoder编码出的c向量跟Decoder解码过程中的每一个输出进行加权运算，在解码的每一个词时，调整权重取到不一样的 IMG_256 c向量。



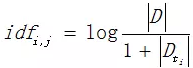
RNN由于其顺序结构训练速度常常受到限制，既然Attention模型本身可以看到全局的信息， 那么一个自然的疑问是我们能不能去掉RNN结构，仅仅依赖于Attention模型呢，这样我们可以使训练并行化，同时拥有全局信息？

**Self-Attention 和 Transformer**

相距较远时RNN的效果常常较差，且由于其顺序性处理效率也较低。

Self-Attention则利用了Attention机制，计算每个单词与其他所有单词之间的关联，计算Attention score。利用这些Attention score就可以得到一个加权的表示，然后再放到一个前馈神经网络中得到新的表示，这一表示很好的考虑到上下文的信息。距离为1。

**TF-IDF**

**预训练**

**统计语言模型**

广泛应用于语音识别、机器翻译、分词、词性标注等任务

利用贝叶斯公式，每个单词出现的概率与之前所有的单词相关



字典大小为N，句子长度为T，则参数规模为 TNT ，还需要保存计算好的概率，以后使用

目标函数



**N-Gram**

假设每个词出现的概率只与前n-1个词相关

参数规模 TNn

平滑：统计词频时要在分母加1，避免为0或1的情况（两个词没有共现或每次都共现）

**神经概率语言模型 NNLM**

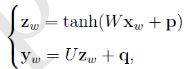
用神经网络训练词向量，开始使用词向量

结构：输入层---投影层---隐藏层---输出层

输入one-hot向量

在投影层首尾拼接

使用前n-1个词预测当前词



再接softmax计算概率值，取概率最大的一个

优势：

词之间的相似性可以通词向量体现

词向量自带平滑功能（概率值不可能为0或1）

**Word2Vec**

输入每个词的one-hot编码，累加之后，经过Word Embedding矩阵Q，得到词向量，Word2Vec是无监督学习

优点：

词向量相对于one-hot维度减少

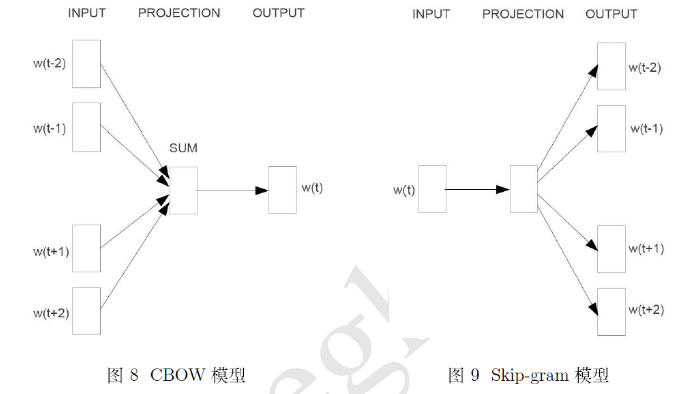
距离近的词向量意义相似

word2vec 缺点：多义词问题。不管上下文是什么，每个词有固定的词向量，无法区分多义词的语义



目标函数

模型结构（没有隐藏层）：



CBOW模型的投射层是将2c个向量相加，所以向量长度还是one-hot长度

skip-gram 的投影层是恒等变换，是多余的，没有任何操作，这么表示只是为了与CBOW比较

哈夫曼树

词频越大的词离根节点越近

可以约定左节点的值更大或更小

权值是叶结点的值，每个结点的值等于左右子树值的和

**结点的带权路径长度**：从根节点到该节点的路径长度与该节点权值的乘积

**树的带权路径长度**：所有叶结点的带权路径长度之和

**哈夫曼树**：带权路径长度最小的二叉树

**前缀编码**：一个字符的编码不能是另一个字符编码的前缀

**哈夫曼编码**：利用哈夫曼树设计的二进制前缀编码

频率低的编码长，频率高的编码短

**分层softmax**

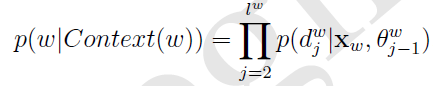
NNLM输出的是概率最大的词，基于分层softmax的CBOW输出的是哈夫曼树

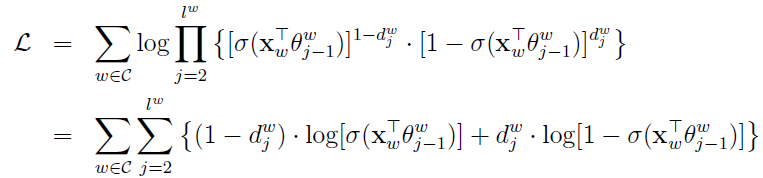
word2vec输出的哈夫曼树左子节点权值更大编码为1，右子节点编码为0

相当于在非叶结点做二分类，复杂度由O(n) 降为 O(log n)

使用sigmoid作非线性激活，正例为1，负例为0，djw 为词的哈夫曼编码（0或1）

**CBOW**

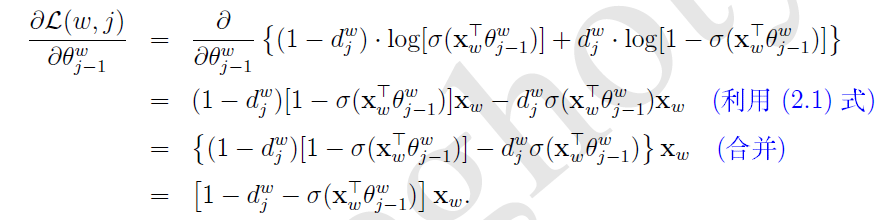
 



为方便起见，将求和符号中的部分取出

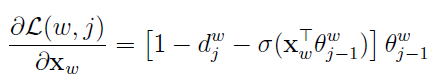


采用梯度上升法求极大值

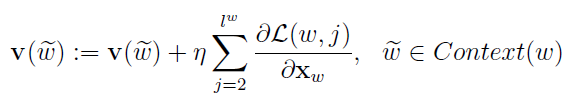




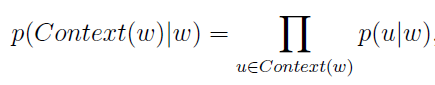
根据对称性，Xw是输入各词的累加

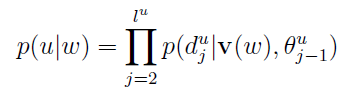


单个词向量的更新将加上累加和的更新

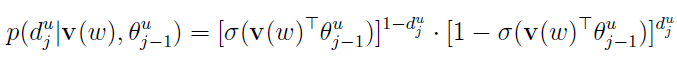


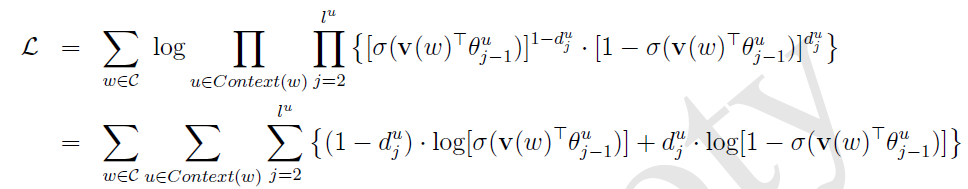
**skip-gram**





其中

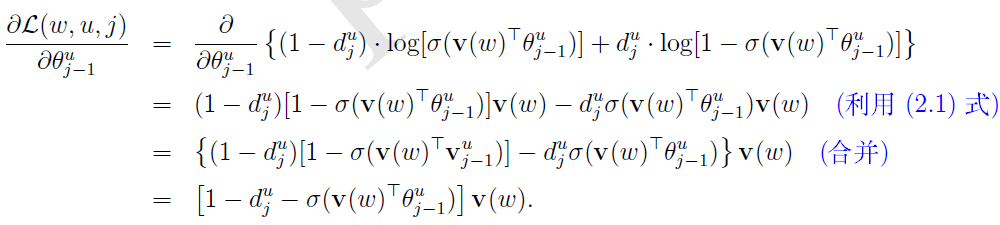




取求和函数中间的部分



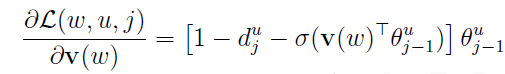
梯度上升法



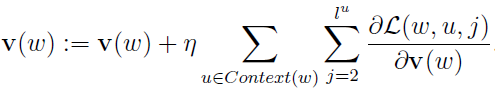
更新公式



根据对称性



更新公式

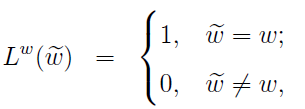


**负采样 Negative Sampling**

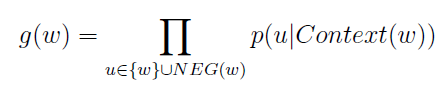
使用随机负采样代替复杂的哈夫曼树，目的是提高训练速度并改善所得词向量的质量

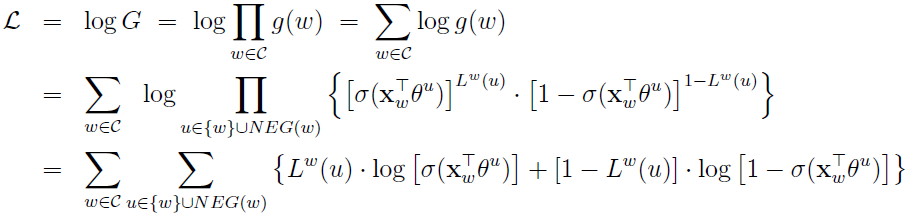
**CBOW**

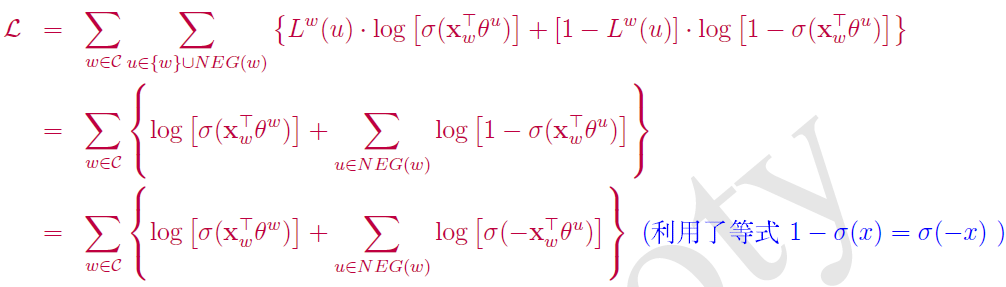
对于给定的Context(w)，词w是一个正样本，其他的词就是负样本



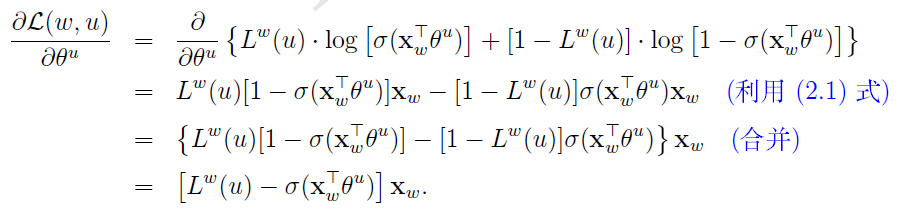
对于给定的正样本，希望最大化







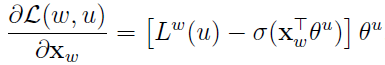




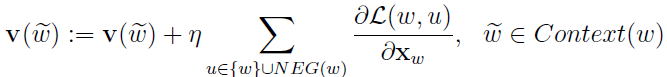
更新公式



根据对称性



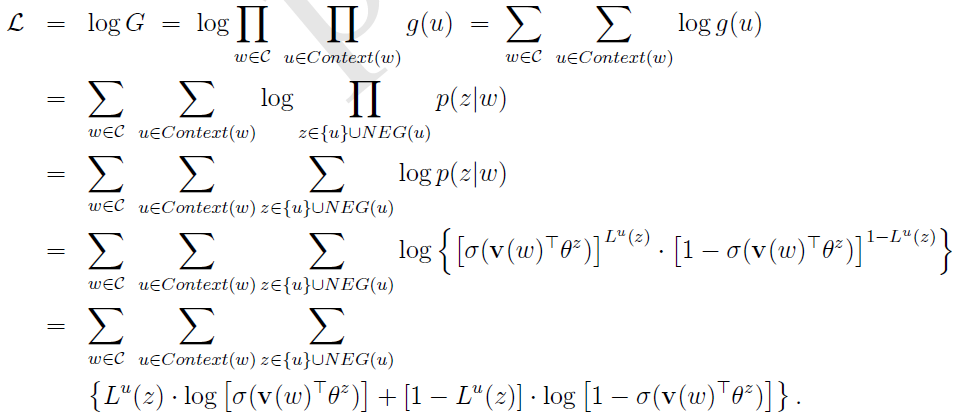
更新公式



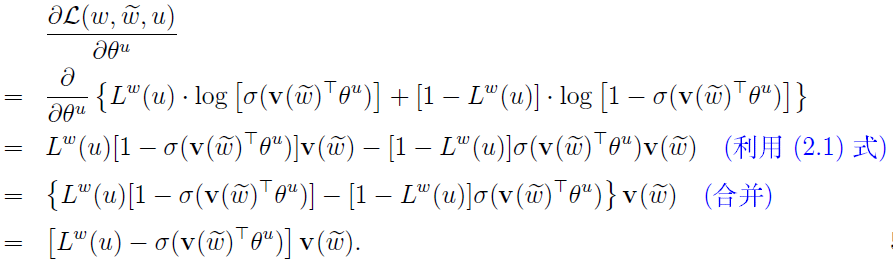
**skip-gram**





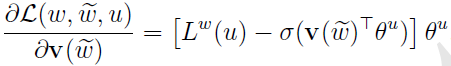


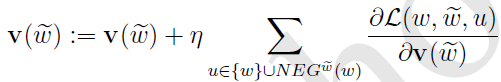


更新公式



根据对称性

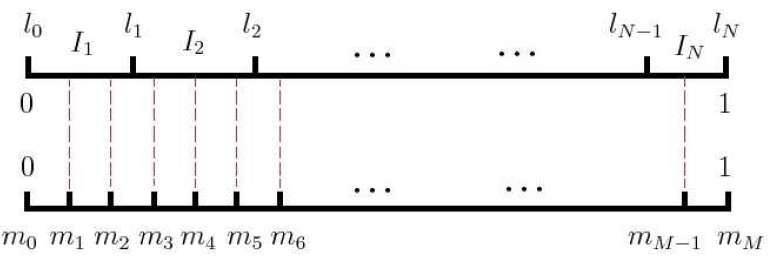




**负采样**

**如何生成NEG(w)？**

语料C中词频有高有低，高频词被选中的概率高，所以是一个**带权采样问题**。



m为等距剖分，n为非等距剖分（长度比例为词频比例）

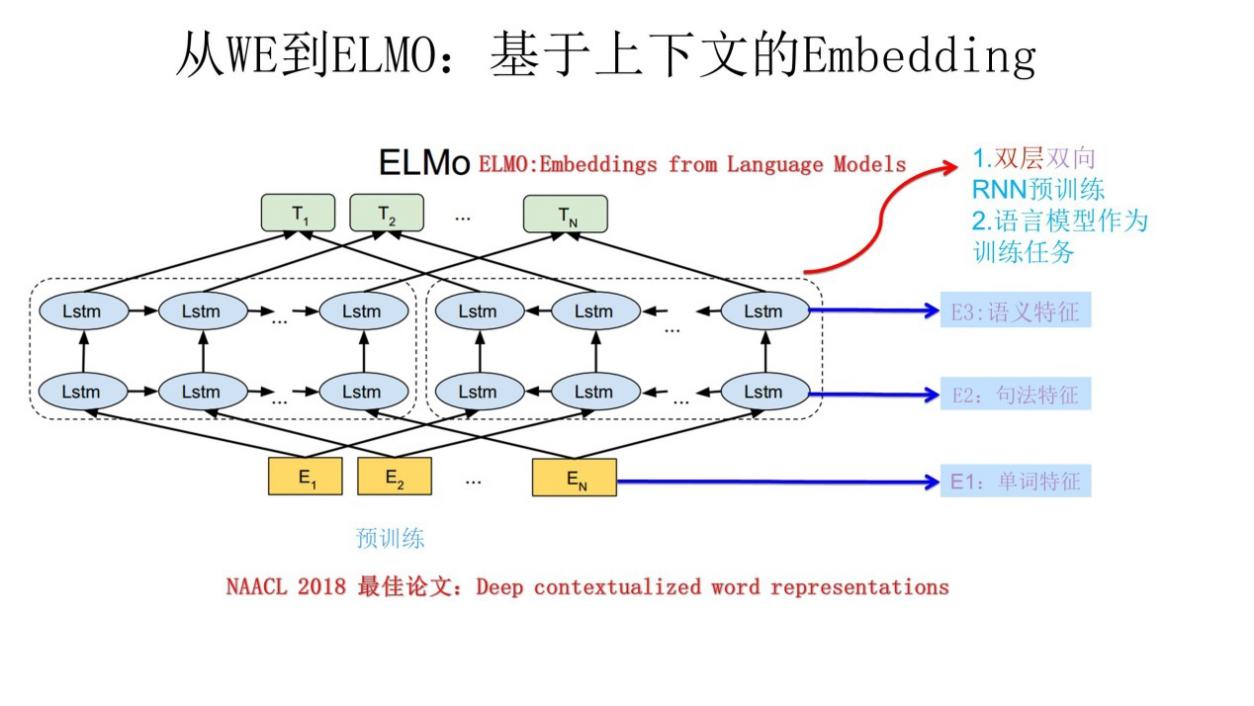
每次生成[1,M - 1]之间的随机数，对应n中的一个词即为负采样的词，如果选到自己，则重新选择

**ELMO**

事先用语言模型学好单词的Word Embedding，对多义词无法区分，不过这没关系。使用Word Embedding的时候，单词已经具备了特定的上下文了，此时再根据当前上下文对Word Embedding动态调整，解决了多义词的问题

ELMO采用了典型的两阶段过程，第一个阶段是利用语言模型进行预训练；第二个阶段是在做下游任务时，从预训练网络中提取对应单词的网络各层的Word Embedding作为新特征补充到下游任务中。

预训练过程，它的网络结构采用了双层双向LSTM，左端的前向双层LSTM代表**正向编码器**，输入的是从左到右顺序的除了预测单词外 的上文Context-before；右端的逆向双层LSTM代表**反向编码器**，输入的是从右到左的逆序的句子下文Context-after；每个编码器的深度都是两层LSTM叠加。



句子中每个单词都能得到对应的三个Embedding:

最底层是单词的Word Embedding，

第一层双向LSTM中对应单词位置的Embedding，句法信息更多一些；

第二层双向LSTM中对应单词位置的Embedding，语义信息更多一些。

给予这三个Embedding中的每一个Embedding一个权重a，这个权重可以学习得来，根据各自权重累加求和，将三个Embedding整合成一个。

优点：ELMO训练的词向量包含了语法和语义的信息，根据上下文动态调整后的embedding解决了多义词问题

缺点：

ELMO使用了LSTM而不是新贵Transformer，Transformer提取特征的能力远强于LSTM

ELMO采取双向拼接这种融合特征的能力比Bert一体化的融合特征方式弱

**GPT**

GPT采用两阶段过程，第一个阶段是利用语言模型进行预训练，第二阶段通过Fine-tuning的模式解决下游任务。

与ELMO主要不同在于两点：

1. 特征抽取器不是用的RNN，而是用的Transformer，它的特征抽取能力要强于RNN；
2. ELMO同时使用了Context-before和Context-after，GPT只使用了Context-before。这个选择现在看不是个太好的选择。

**Attention机制**

并不是所有的词对当前词都重要，所以引入Attention

注意力可以理解成是表示重要性的权重向量

为预测一个元素，使用注意力权重来估计其他元素与其相关性，并将注意力权重加权求和作为计算最终目标的特征

原始的seq2seq 在预测每个目标词时，使用同一个context vector

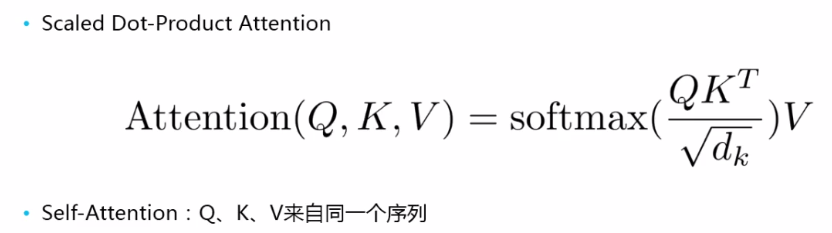
seq2seq 加入attention机制之后，在预测每个目标词时，使用的context vector 都不一样

**第一步求相似度：**计算相似性得到的向量称为对齐向量，可以使用cosine、前馈神经网络等多种计算方法

**第二步使用softmax得到归一化的权重**（self-attention 也被称为 intra-attention）

**Transformer**

**attention**



QKT的操作，可以使用点乘，也可以拼接后输入前馈神经网络，维度低（低于100）时，效果差不多，高维度的时候，使用拼接的效果更好。

用于缩放，数值非常大的时候，分子趋向于0或1。梯度在数值比较高的时候，趋向于0。

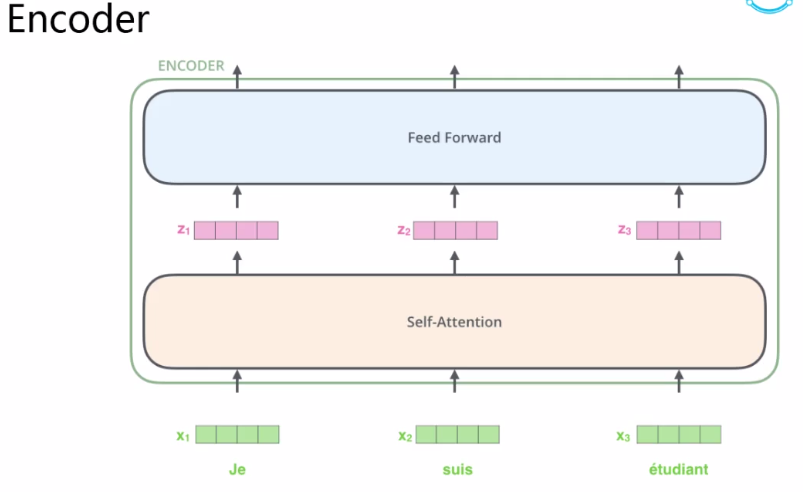
Q K V 由输入分别与一个矩阵相乘得到

Q1 与 K中每一个进行比较（点乘），除以缩放系数，再进行softmax。得到score1

score1 乘以 V1，score2 乘以V2... 加权求和得到Q1 对应的sum 向量 Z1

Q1 与 Z1 的维度可能不一样

将得到的所有向量拼接，得到矩阵Z



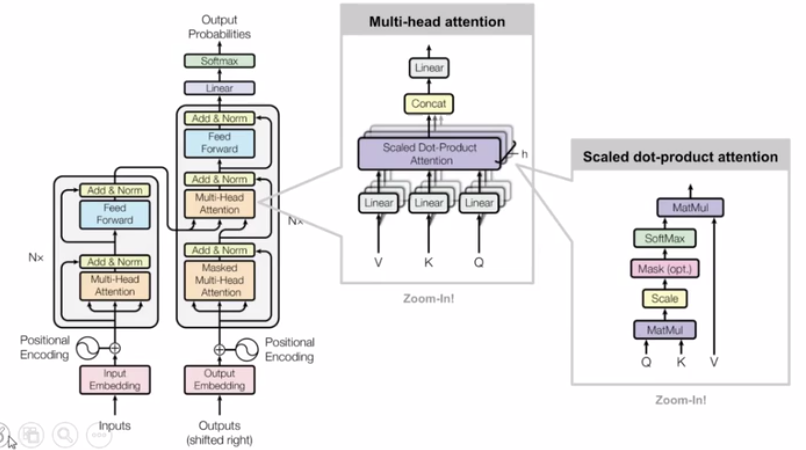
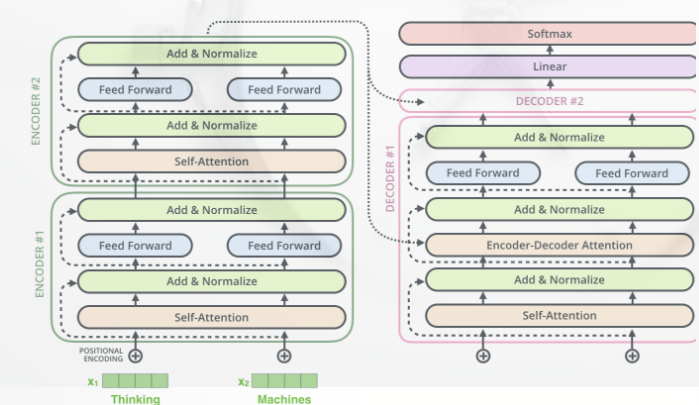
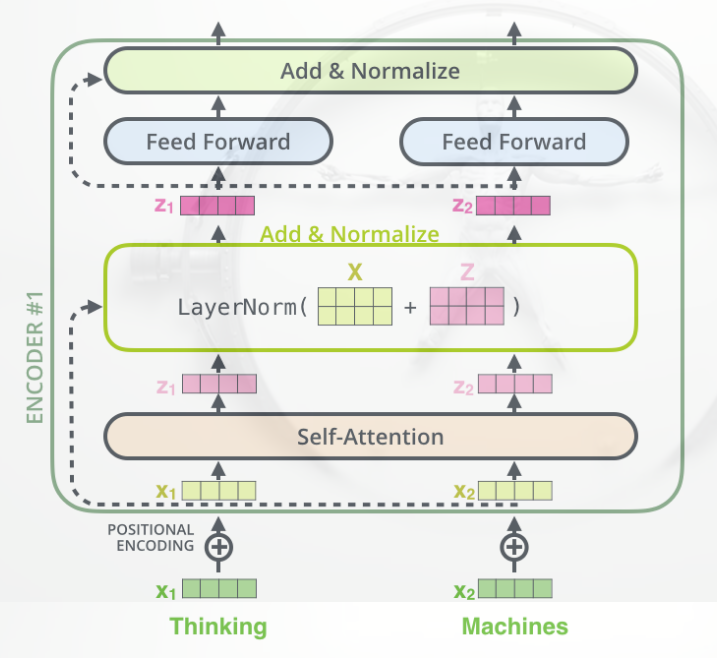
**Multi-Head Attention**

多个self-attention ，将得到的矩阵拼接在一起，再进行一个变换，得到矩阵Z

Encoder中输入经过self-attention 得到的数据长度不变

feed forward参数共享，对每一个Z进行相同的变换

编码器中，多个Encoder相连，论文中是6个Encoder 相连，可以尝试别的层数，Add & Normalization中必须使输入X 和self-attention 的输出Z 维度相同 ，因为要把Z 和X 相加再进行LayerNorm，Encoder 和 Decoder都可以重复多次



Decoder中，**经过Masked Multihead Attention之后作为Q，接入Encoder的输出作为K和V。**

Decoder的输出经过一个线性映射，接softmax做分类

Q K V 都是不带位置信息的词袋模型，在输入中加入位置信息 Positional Embedding，得到位置编码

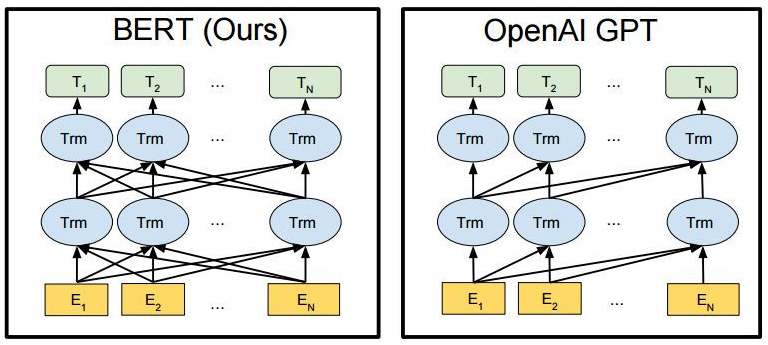
Mask 操作是把之后所有的词去掉，只关注到输出序列中较前的位置。这是通过在自注意力计算的softmax步骤之前用掩码mask遮罩序列中后面的位置（将它们设置为为负无穷）来实现的。

**Bert**

word embedding (如word2vec) 的预训练表示的是上下文无关的，对于多义词问题无法解决

Bert 是一种预训练语言表示的方法，与上下文相关

Bert 的改进是引入了masked language model



GPT第一个提出预训练加微调方案

GPT在预测词的时候，只预测下一个词，因此只能用到上文的信息，无法利用到下文的信息。BERT是预测文中抠掉的词，可以充分利用到上下文的信息，这使得模型有更强的表达能力，这也是BERT中Bidirectional的含义。

BERT有了第二个任务，就能够很好的捕捉句子之间的关系。

Bert预训练包括了两个任务，第一个任务是随机地扣掉15%的单词，用一个掩码MASK代替，让模型去猜测这个单词；第二个任务是，每个训练样本是一个上下句，有50%的样本，下句和上句是真实的，另外50%的样本，下句和上句是无关的，模型需要判断两句的关系。

**多任务过程**

1. Masked双向语言模型：Masked LM借鉴了CBOW的思想，随机选择语料中15%的单词，把它抠掉，也就是用[Mask]掩码代替原始单词，然后要求模型去正确预测被抠掉的单词。
2. Next Sentence Prediction：分两种情况选择两个句子，一种是选择语料中真正顺序相连的两个句子；另外一种是随机选择一个句子拼到第一个句子后面。判断第二个句子是不是真的是第一个句子的后续句子。

Bert的输入部分：

输入部分是个线性序列，两个句子通过分隔符分割，最前面和最后增加两个标识符号。

每个单词有三个embedding，叠加之后就形成了Bert的输入:

**位置信息embedding**，这是因为NLP中单词顺序是很重要的特征，需要对位置信息进行编码；

**单词embedding**,这个就是我们之前一直提到的单词embedding；

**句子embedding**，前面提到训练数据都是由两个句子构成的，每个句子有个整体的embedding项对应给每个单词。

**GTP 2.0**

GPT 2.0仍然选择单向语言模型，首先把Transformer模型参数扩容，常规的Transformer Big包含24个叠加的Block，用更多的训练数据来做预训练，更大的模型，更多的参数，意味着更高的模型容量。

使用更大数量的无监督训练数据，这样训练出来的语言模型，通用性好，覆盖几乎任何领域的内容，这意味着它可以用于任意领域的下游任务

没有做fine tuning，而是无监督地去做下游任务

说明了第一个阶段的预训练过程，如果采用更高质量的数据，采用更宽泛的数据，采用更大量的数据，Transformer采用更复杂的模型，那么在Transformer里能学会更多更好的NLP的通用知识。如果第二阶段仍然采取Finetuning，对下游任务的提升效果是可以期待的。

**tf.Dataset**

**读取数据**

tf.data.Dataset.from\_tensor\_slice() 从ndarray中读取

tf.data.TextLineDataset() 输入文件名，读取文件中的每一行

tf.data.FixedLengthRecordDataset() 读取二进制文件，输入文件名和每次读取的字节大小

**读取模式**

非 eager 模式 每次从iterator中读取一个tensor，需要sess.run()

eager 模式 创建iterator方式不同，迭代时直接取出，不需要sess.run()

**函数**

dataset.map 将Dataset中的每个元素输入函数，将返回值作为新的Datset

dataset **=** dataset**.**map(**lambda** x: x **+** 1)

dataset.batch 将多个元素组合成batch

dataset **=** dataset**.**batch(32)

dataset.shuffle 打乱dataset中的元素，参数为buffer\_size，达到buffer\_size后打乱

dataset **=** dataset**.**shuffle(buffer\_size**=**10000)

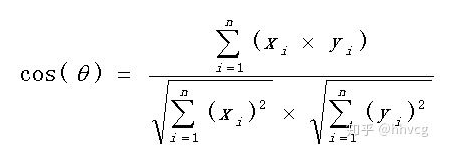
dataset.repeat 将整个序列重复多次，处理机器学习中的epoch，例子中是5个epoch

dataset **=** dataset**.**repeat(5)

**文本相似度的度量**

两句子间语义相似度最简单的方法就是求句子中所有单词词嵌入的平均值，然后计算两句子词嵌入之间的余弦相似性。

\* 求一句话中词嵌入的平均值似乎给与不相关的单词太多权重了



词移距离使用两文本间的词嵌入，测量其中一文本中的单词在语义空间中移动到另一文本单

词所需要的最短距离。